MATHEMATICS FOR COMPUTER SCIENCE



Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer
Google and Massachusetts Institute of Technology

LibreTexts™

Mathematics for Computer Science (Lehman, Leighton, and Meyer)

This text is disseminated via the Open Education Resource (OER) LibreTexts Project (https://LibreTexts.org) and like the thousands of other texts available within this powerful platform, it is freely available for reading, printing, and "consuming."

The LibreTexts mission is to bring together students, faculty, and scholars in a collaborative effort to provide an accessible, and comprehensive platform that empowers our community to develop, curate, adapt, and adopt openly licensed resources and technologies; through these efforts we can reduce the financial burden born from traditional educational resource costs, ensuring education is more accessible for students and communities worldwide.

Most, but not all, pages in the library have licenses that may allow individuals to make changes, save, and print this book. Carefully consult the applicable license(s) before pursuing such effects. Instructors can adopt existing LibreTexts texts or Remix them to quickly build course-specific resources to meet the needs of their students. Unlike traditional textbooks, LibreTexts' web based origins allow powerful integration of advanced features and new technologies to support learning.



LibreTexts is the adaptable, user-friendly non-profit open education resource platform that educators trust for creating, customizing, and sharing accessible, interactive textbooks, adaptive homework, and ancillary materials. We collaborate with individuals and organizations to champion open education initiatives, support institutional publishing programs, drive curriculum development projects, and more.

The LibreTexts libraries are Powered by NICE CXone Expert and was supported by the Department of Education Open Textbook Pilot Project, the California Education Learning Lab, the UC Davis Office of the Provost, the UC Davis Library, the California State University Affordable Learning Solutions Program, and Merlot. This material is based upon work supported by the National Science Foundation under Grant No. 1246120, 1525057, and 1413739.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation nor the US Department of Education.

Have questions or comments? For information about adoptions or adaptions contact info@LibreTexts.org or visit our main website at https://LibreTexts.org.

This text was compiled on 12/16/2025

TABLE OF CONTENTS

Licensing

1: Proofs

- 1: Intro to Proofs
  - 1.1: Propositions
  - 1.2: Predicates
  - 1.3: The Axiomatic Method
  - 1.4: Our Axioms
  - 1.5: Proving an Implication
  - 1.6: Proving an “If and Only If”
  - 1.7: Proof by Cases
  - 1.8: Proof by Contradiction
  - 1.9: Good Proofs in Practice
- 2: Well Ordering Principle
  - 2.1: Well Ordering Proofs
  - 2.2: Template for Well Ordering Proofs
  - 2.3: Factoring into Primes
  - 2.4: Well Ordered Sets
- 3: Logical Formulas
  - 3.1: Propositions from Propositions
  - 3.2: Propositional Logic in Computer Programs
  - 3.3: Equivalence and Validity
  - 3.4: The Algebra of Propositions
  - 3.5: The SAT Problem
  - 3.6: Predicate Formulas
- 4: Mathematical Data Types
  - 4.1: Sets
  - 4.2: Sequences
  - 4.3: Functions
  - 4.4: Binary Relations
  - 4.5: Finite Cardinality
- 5: Induction
  - 5.1: Ordinary Induction
  - 5.2: Strong Induction
  - 5.3: Strong Induction vs. Induction vs. Well Ordering
  - 5.4: State Machines
- 6: Recursive Data Types
  - 6.1: Recursive Definitions and Structural Induction
  - 6.2: Strings of Matched Brackets
  - 6.3: Recursive Functions on Nonnegative Integers
  - 6.4: Arithmetic Expressions
  - 6.5: Induction in Computer Science
- 7: Infinite Sets
  - 7.1: Infinite Cardinality

- 7.2: The Halting Problem
- 7.3: The Logic of Sets
- 7.4: Does All This Really Work?

2: Structures

- 8: Number Theory
  - 8.1: Divisibility
  - 8.2: The Greatest Common Divisor
  - 8.3: Prime Mysteries
  - 8.4: The Fundamental Theorem of Arithmetic
  - 8.5: Alan Turing
  - 8.6: Modular Arithmetic
  - 8.7: Remainder Arithmetic
  - 8.8: Turing's Code (Version 2.0)
  - 8.9: Multiplicative Inverses and Cancelling
  - 8.10: Euler's Theorem
  - 8.11: RSA Public Key Encryption
  - 8.12: What has SAT got to do with it?
- 9: Directed graphs and Partial Orders
  - 9.1: Vertex Degrees
  - 9.2: Walks and Paths
  - 9.3: Adjacency Matrices
  - 9.4: Walk Relations
  - 9.5: Directed Acyclic Graphs and Scheduling
  - 9.6: Partial Orders
  - 9.7: Representing Partial Orders by Set Containment
  - 9.8: Linear Orders
  - 9.9: Product Orders
  - 9.10: Equivalence Relations
  - 9.11: Summary of Relational Properties
- 10: Communication Networks
  - 10.1: Complete Binary Tree
  - 10.2: Routing Problems
  - 10.3: Network Diameter
  - 10.4: Switch Count
  - 10.5: Network Latency
  - 10.6: Congestion
  - 10.7: 2-D Array
  - 10.8: Butterfly
  - 10.9: Beneš Network
- 11: Simple Graphs
  - 11.1: Vertex Adjacency and Degrees
  - 11.2: Sexual Demographics in America
  - 11.3: Some Common Graphs
  - 11.4: Isomorphism
  - 11.5: Bipartite Graphs and Matchings
  - 11.6: The Stable Marriage Problem
  - 11.7: Coloring

- 11.8: Simple Walks
- 11.9: Connectivity
- 11.10: Forests and Trees
- 12: Planar Graphs
  - 12.1: Drawing Graphs in the Plane
  - 12.2: Definitions of Planar Graphs
  - 12.3: Euler's Formula
  - 12.4: Bounding the Number of Edges in a Planar Graph
  - 12.5: Returning to K5 and K3,3
  - 12.6: Coloring Planar Graphs
  - 12.7: Classifying Polyhedra
  - 12.8: Another Characterization for Planar Graphs

3: Counting

- 13: Sums and Asymptotics
  - 13.1: The Value of an Annuity
  - 13.2: Sums of Powers
  - 13.3: Approximating Sums
  - 13.4: Hanging Out Over the Edge
  - 13.5: Products
  - 13.6: Double Trouble
  - 13.7: Asymptotic Notation
- 14: Cardinality Rules
  - 14.1: Counting One Thing by Counting Another
  - 14.2: Counting Sequences
  - 14.3: The Generalized Product Rule
  - 14.4: The Division Rule
  - 14.5: Counting Subsets
  - 14.6: Sequences with Repetitions
  - 14.7: Counting Practice - Poker Hands
  - 14.8: The Pigeonhole Principle
  - 14.9: Inclusion-Exclusion
  - 14.10: Combinatorial Proofs
- 15: Generating Functions
  - 15.1: Infinite Series
  - 15.2: Counting with Generating Functions
  - 15.3: Partial Fractions
  - 15.4: Solving Linear Recurrences
  - 15.5: Formal Power Series

4: Probability

- 16: Events and Probability Spaces
  - 16.1: Let's Make a Deal
  - 16.2: The Four Step Method
  - 16.3: Strange Dice
  - 16.4: The Birthday Principle
  - 16.5: Set Theory and Probability

- 17: Conditional Probability
  - 17.1: Monty Hall Confusion
  - 17.2: Definition and Notation
  - 17.3: The Four-Step Method for Conditional Probability
  - 17.4: Why Tree Diagrams Work
  - 17.5: The Law of Total Probability
  - 17.6: Simpson's Paradox
  - 17.7: Independence
  - 17.8: Mutual Independence
- 18: Random Variables
  - 18.1: Random Variable Examples
  - 18.2: Independence
  - 18.3: Distribution Functions
  - 18.4: Great Expectations
  - 18.5: Linearity of Expectation
- 19: Deviation from the Mean
  - 19.1: Markov's Theorem
  - 19.2: Chebyshev's Theorem
  - 19.3: Properties of Variance
  - 19.4: Estimation by Random Sampling
  - 19.5: Confidence versus Probability
  - 19.6: Sums of Random Variables
  - 19.7: Really Great Expectations
- 20: Random Walks
  - 20.1: Gambler's Ruin
  - 20.2: Random Walks on Graphs

5: Recurrences

- 21: Recurrences
  - 21.1: The Towers of Hanoi
  - 21.2: Merge Sort
  - 21.3: Linear Recurrences
  - 21.4: Divide-and-Conquer Recurrences
  - 21.5: A Feel for Recurrences

Index

Glossary

Detailed Licensing

Licensing

*A detailed breakdown of this resource's licensing can be found in **Back Matter/Detailed Licensing**.*

SECTION OVERVIEW

1: Proofs

1: Intro to Proofs

1.1: Propositions
1.2: Predicates
1.3: The Axiomatic Method
1.4: Our Axioms
1.5: Proving an Implication
1.6: Proving an “If and Only If”
1.7: Proof by Cases
1.8: Proof by Contradiction
1.9: Good Proofs in Practice

2: Well Ordering Principle

2.1: Well Ordering Proofs
2.2: Template for Well Ordering Proofs
2.3: Factoring into Primes
2.4: Well Ordered Sets

3: Logical Formulas

3.1: Propositions from Propositions
3.2: Propositional Logic in Computer Programs
3.3: Equivalence and Validity
3.4: The Algebra of Propositions
3.5: The SAT Problem
3.6: Predicate Formulas

4: Mathematical Data Types

4.1: Sets
4.2: Sequences
4.3: Functions
4.4: Binary Relations
4.5: Finite Cardinality

5: Induction

5.1: Ordinary Induction
5.2: Strong Induction
5.3: Strong Induction vs. Induction vs. Well Ordering
5.4: State Machines

6: Recursive Data Types

6.1: Recursive Definitions and Structural Induction
6.2: Strings of Matched Brackets
6.3: Recursive Functions on Nonnegative Integers
6.4: Arithmetic Expressions
6.5: Induction in Computer Science

7: Infinite Sets

7.1: Infinite Cardinality
7.2: The Halting Problem
7.3: The Logic of Sets
7.4: Does All This Really Work?

Thumbnail: P. Oxy. 29, one of the oldest surviving fragments of Euclid's Elements, a textbook used for millennia to teach proof-writing techniques. The diagram accompanies Book II, Proposition 5. (Public Domain). Text from Oscar Levin's Discrete Mathematics text (CC BY-SA).

This page titled 1: Proofs is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

1: Intro to Proofs

1.1: Propositions
1.2: Predicates
1.3: The Axiomatic Method
1.4: Our Axioms
1.5: Proving an Implication
1.6: Proving an “If and Only If”
1.7: Proof by Cases
1.8: Proof by Contradiction
1.9: Good Proofs in Practice

This page titled 1: Intro to Proofs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

1.1: Propositions

Definition

A *proposition* is a statement (communication) that is either true or false.

For example, both of the following statements are propositions. The first is true, and the second is false.

Proposition 1.1.1. *2 + 3 = 5.*

Proposition 1.1.2. *1 + 1 = 3.*

Being true or false doesn't sound like much of a limitation, but it does exclude statements such as "Wherefore art thou Romeo?" and "Give me an A!" It also excludes statements whose truth varies with circumstance such as, "It's five o'clock," or "the stock market will rise tomorrow."

Unfortunately it is not always easy to decide if a proposition is true or false:

Proposition 1.1.3. *For every nonnegative integer, n, the value of $n^2 + n + 41$ is prime.*

(A *prime* is an integer greater than 1 that is not divisible by any other integer greater than 1. For example, 2, 3, 5, 7, 11, are the first five primes.) Let's try some numerical experimentation to check this proposition. Let

$$p(n) ::= n^2 + n + 41.^1 \tag{1.1.1}$$

We begin with $p(0) = 41$, which is prime; then

$$p(1) = 43, p(2) = 47, p(3) = 53, \ldots, p(20) = 461$$

are each prime. Hmmm, starts to look like a plausible claim. In fact we can keep checking through $n = 39$ and confirm that $p(39) = 1601$ is prime.

But $p(40) = 40^2 + 40 + 41 = 41 \cdot 41$, which is not prime. So it's not true that the expression is prime *for all* nonnegative integers. In fact, it's not hard to show that *no* polynomial with integer coefficients can map all nonnegative numbers into prime numbers, unless it's a constant (see Problem 1.17). But the real point of this example is to show that in general, you can't check a claim about an infinite set by checking a finite set of its elements, no matter how large the finite set.

By the way, propositions like this about all numbers or all items of some kind are so common that there is a special notation for them. With this notation, Proposition 1.1.3 would be

$$\forall n \in \mathbb{N}, p(n) \text{ is prime.} \tag{1.1.2}$$

Here the symbol $\forall$ is read "for all." The symbol $\mathbb{N}$ stands for the set of *nonnegative integers*: $0, 1, 2, 3, \ldots$(ask your instructor for the complete list). The symbol "$\in$" is read as "is a member of," or "belongs to," or simply as "is in." The period after the $\mathbb{N}$ is just a separator between phrases.

Here are two even more extreme examples:

Proposition 1.1.4. *[Euler's Conjecture] The equation*

$$a^4 + b^4 + c^4 = d^4$$

has no solution when a, b, c, d are positive integers.

Euler (pronounced "oiler") conjectured this in 1769. But the proposition was proved false 218 years later by Noam Elkies at a liberal arts school up Mass Ave. The solution he found was $a = 95800, b = 217519, c = 414560, d = 422481$.

In logical notation, Euler's Conjecture could be written,

$$\forall a \in \mathbb{Z}^+ \forall b \in \mathbb{Z}^+ \forall c \in \mathbb{Z}^+ \forall d \in \mathbb{Z}^+ . a^4 + b^4 + c^4 \neq d^4.$$

Here, $\mathbb{Z}^+$ is a symbol for the positive integers. Strings of $\forall$'s like this are usually abbreviated for easier reading:

$$\forall a, b, c, d \in \mathbb{Z}^+ . a^4 + b^4 + c^4 \neq d^4.$$

Proposition 1.1.5. $313(x^3 + y^3) = z^3$ *has no solution when* $x, y, z \in \mathbb{Z}^+$.

This proposition is also false, but the smallest counterexample has more than 1000 digits!

It's worth mentioning a couple of further famous propositions whose proofs were sought for centuries before finally being discovered:

Proposition 1.1.6 (Four Color Theorem). *Every map can be colored with 4 colors so that adjacent*[2] *regions have different colors.*

Several incorrect proofs of this theorem have been published, including one that stood for 10 years in the late 19th century before its mistake was found. A laborious proof was finally found in 1976 by mathematicians Appel and Haken, who used a complex computer program to categorize the four-colorable maps. The program left a few thousand maps uncategorized, which were checked by hand by Haken and his assistants—among them his 15-year-old daughter.

There was reason to doubt whether this was a legitimate proof: the proof was too big to be checked without a computer. No one could guarantee that the computer calculated correctly, nor was anyone enthusiastic about exerting the effort to recheck the four-colorings of thousands of maps that were done by hand. Two decades later a mostly intelligible proof of the Four Color Theorem was found, though a computer is still needed to check four-colorability of several hundred special maps.[3]

Proposition 1.1.7 (Fermat's Last Theorem). *There are no positive integers* x, y, *and* z *such that*

$$x^n + y^n = z^n$$

for some integer $n > 2$.

In a book he was reading around 1630, Fermat claimed to have a proof for this proposition, but not enough space in the margin to write it down. Over the years, the Theorem was proved to hold for all n up to 4,000,000, but we've seen that this shouldn't necessarily inspire confidence that it holds for *all* n. There is, after all, a clear resemblance between Fermat's Last Theorem and Euler's false Conjecture. Finally, in 1994, British mathematician Andrew Wiles gave a proof, after seven years of working in secrecy and isolation in his attic. His proof did not fit in any margin.[4]

Finally, let's mention another simply stated proposition whose truth remains unknown.

Proposition 1.1.8 *(Goldbach's Conjecture). Every even integer greater than 2 is the sum of two primes.*

Goldbach's Conjecture dates back to 1742. It is known to hold for all numbers up to 10^{18} but to this day, no one knows whether it's true or false.

For a computer scientist, some of the most important things to prove are the correctness of programs and systems—whether a program or system does what it's supposed to. Programs are notoriously buggy, and there's a growing community of researchers and practitioners trying to find ways to prove program correctness. These efforts have been successful enough in the case of CPU chips that they are now routinely used by leading chip manufacturers to prove chip correctness and avoid mistakes like the notorious Intel division bug in the 1990's. Developing mathematical methods to verify programs and systems remains an active research area. We'll illustrate some of these methods in Chapter 5.

[1]The symbol ::= means "equal by definition." It's always ok simply to write "=" instead of ::=, but reminding the reader that an equality holds by definition can be helpful.

[2]Two regions are adjacent only when they share a boundary segment of positive length. They are not considered to be adjacent if their boundaries meet only at a few points.

[3]The story of the proof of the Four Color Theorem is told in a well-reviewed popular (nontechnical) book: "Four Colors Suffice. How the Map Problem was Solved." Robin Wilson. Princeton Univ. Press, 2003, 276pp. ISBN 0-691-11533-8.

[4]In fact, Wiles' original proof was wrong, but he and several collaborators used his ideas to arrive at a correct proof a year later. This story is the subject of the popular book, Fermat's Enigma by Simon Singh, Walker & Company, November, 1997.

This page titled 1.1: Propositions is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

1.2: Predicates

A *predicate* can be understood as a proposition whose truth depends on the value of one or more variables. So "n is a perfect square" describes a predicate, since you can't say if it's true or false until you know what the value of the variable n happens to be. Once you know, for example, that n equals 4, the predicate becomes the true proposition "4 is a perfect square". Remember, nothing says that the proposition has to be true: if the value of n were 5, you would get the false proposition "5 is a perfect square."

Like other propositions, predicates are often named with a letter. Furthermore, a function-like notation is used to denote a predicate supplied with specific variable values. For example, we might use the name "P" for predicate above:

$$P(n) ::= \text{“}n \text{ is a perfect square”},$$

and repeat the remarks above by asserting that $P(4)$ is true, and $P(5)$ is false.

This notation for predicates is confusingly similar to ordinary function notation. If P is a predicate, then $P(n)$ is either *true or false*, depending on the value of n. On the other hand, if p is an ordinary function, like $n^2 + 1$, then $p(n)$ is a *numerical quantity*. **Don't confuse these two!**

This page titled 1.2: Predicates is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

1.3: The Axiomatic Method

The standard procedure for establishing truth in mathematics was invented by Euclid, a mathematician working in Alexandria, Egypt around 300 BC. His idea was to begin with five *assumptions* about geometry, which seemed undeniable based on direct experience. (For example, “There is a straight line segment between every pair of points”.) Propositions like these that are simply accepted as true are called *axioms*.

Starting from these axioms, Euclid established the truth of many additional propositions by providing “proofs.” A *proof* is a sequence of logical deductions from axioms and previously proved statements that concludes with the proposition in question. You probably wrote many proofs in high school geometry class, and you’ll see a lot more in this text.

There are several common terms for a proposition that has been proved. The different terms hint at the role of the proposition within a larger body of work.

- Important true propositions are called *theorems*.
- A *lemma* is a preliminary proposition useful for proving later propositions.
- A *corollary* is a proposition that follows in just a few logical steps from a theorem.

These definitions are not precise. In fact, sometimes a good lemma turns out to be far more important than the theorem it was originally used to prove.

Euclid’s axiom-and-proof approach, now called the *axiomatic method*, remains the foundation for mathematics today. In fact, just a handful of axioms, called the Zermelo-Fraenkel with Choice axioms(ZFC), together with a few logical deduction rules, appear to be sufficient to derive essentially all of mathematics. We’ll examine these in Chapter 7.

This page titled 1.3: The Axiomatic Method is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

1.4: Our Axioms

The ZFC axioms are important in studying and justifying the foundations of mathematics, but for practical purposes, they are much too primitive. Proving theorems in ZFC is a little like writing programs in byte code instead of a full-fledged programming language—by one reckoning, a formal proof in ZFC that $2+2=4$ requires more than 20,000 steps! So instead of starting with ZFC, we're going to take a *huge* set of axioms as our foundation: we'll accept all familiar facts from high school math.

This will give us a quick launch, but you may find this imprecise specification of the axioms troubling at times. For example, in the midst of a proof, you may start to wonder, "Must I prove this little fact or can I take it as an axiom?" There really is no absolute answer, since what's reasonable to assume and what requires proof depends on the circumstances and the audience. A good general guideline is simply to be up front about what you're assuming.

Logical Deductions

Logical deductions, or *inference rules*, are used to prove new propositions using previously proved ones.

A fundamental inference rule is *modus ponens*. This rule says that a proof of P together with a proof that P IMPLIES Q is a proof of Q.

Inference rules are sometimes written in a funny notation. For example, *modus ponens* is written:

Rule.

$$\frac{P, \quad P \text{ IMPLIES } Q}{Q}$$

When the statements above the line, called the *antecedents*, are proved, then we can consider the statement below the line, called the *conclusion* or *consequent*, to also be proved.

A key requirement of an inference rule is that it must be *sound*: an assignment of truth values to the letters, $P, Q, \ldots$, that makes all the antecedents true must also make the consequent true. So if we start off with true axioms and apply sound inference rules, everything we prove will also be true.

There are many other natural, sound inference rules, for example:

Rule.

$$\frac{P \text{ IMPLIES } Q, \quad Q \text{ IMPLIES } R}{P \text{ IMPLIES } R}$$

Rule.

$$\frac{\text{NOT}(P) \text{ IMPLIES NOT}(Q)}{Q \text{ IMPLIES } P}$$

On the other hand,

Non-Rule.

$$\frac{\text{NOT}(P) \text{ IMPLIES NOT}(Q)}{P \text{ IMPLIES } Q}$$

is not sound: if P is assigned **T** and Q is assigned **F**, then the antecedent is true and the consequent is not.

As with axioms, we will not be too formal about the set of legal inference rules. Each step in a proof should be clear and "logical"; in particular, you should state what previously proved facts are used to derive each new conclusion.

Patterns of Proof

In principle, a proof can be *any* sequence of logical deductions from axioms and previously proved statements that concludes with the proposition in question. This freedom in constructing a proof can seem overwhelming at first. How do you even *start* a proof?

Here’s the good news: many proofs follow one of a handful of standard templates. Each proof has it own details, of course, but these templates at least provide you with an outline to fill in. We’ll go through several of these standard patterns, pointing out the basic idea and common pitfalls and giving some examples. Many of these templates fit together; one may give you a top-level outline while others help you at the next level of detail. And we’ll show you other, more sophisticated proof techniques later on.

The recipes below are very specific at times, telling you exactly which words to write down on your piece of paper. You’re certainly free to say things your own way instead; we’re just giving you something you *could* say so that you’re never at a complete loss.

This page titled 1.4: Our Axioms is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

1.5: Proving an Implication

Propositions of the form "If P, then Q" are called *implications*. This implication is often rephrased as "P IMPLIES Q."

Here are some examples:

- (Quadratic Formula) If $ax^2 + bx + c = 0$ and $a \neq 0$, then
$$x = \left(-b \pm \sqrt{b^2 - 4ac}\right)/2a.$$
- (Goldbach's Conjecture 1.1.8 rephrased) If n is an even integer greater than 2, then n is a sum of two primes.
- If $0 \leq x \leq 2$, then $-x^3 + 4x + 1 > 0$.

There are a couple of standard methods for proving an implication.

Method #1

In order to prove that P IMPLIES Q:

1. Write, "Assume P."

2. Show that Q logically follows.

Example

Theorem 1.5.1

If $0 \leq x \leq 2$, then $-x^3 + 4x + 1 > 0$.

Before we write a proof of this theorem, we have to do some scratchwork to figure out why it is true.

The inequality certainly holds for $x = 0$; then the left side is equal to 1 and $1 > 0$. As x grows, the $4x$ term (which is positive) initially seems to have greater magnitude than $-x^3$ (which is negative). For example, when $x = 1$, we have $4x = 4$, but $-x^3 = -1$ only. In fact, it looks like $-x^3$ doesn't begin to dominate until $x > 2$. So it seems the $-x^3 + 4x$ part should be nonnegative for all x between 0 and 2, which would imply that $-x^3 + 4x + 1$ is positive.

So far, so good. But we still have to replace all those "seems like" phrases with solid, logical arguments. We can get a better handle on the critical $-x^3 + 4x$ part by factoring it, which is not too hard:

$$-x^3 + 4x = x(2 - x)(2 + x)$$

Aha! For x between 0 and 2, all of the terms on the right side are nonnegative. And a product of nonnegative terms is also nonnegative. Let's organize this blizzard of observations into a clean proof.

Proof. Assume $0 \leq x \leq 2$. Then $x, 2 - x$, and $2 + x$ are all nonnegative. Therefore, the product of these terms is also nonnegative. Adding 1 to this product gives a positive number, so:

$$x(2 - x)(2 + x) + 1 > 0$$

Multiplying out on the left side proves that

$$-x^3 + 4x + 1 > 0$$

as claimed. ■

There are a couple points here that apply to all proofs:

- You'll often need to do some scratchwork while you're trying to figure out the logical steps of a proof. Your scratchwork can be as disorganized as you like—full of dead-ends, strange diagrams, obscene words, whatever. But keep your scratchwork separate from your final proof, which should be clear and concise.
- Proofs typically begin with the word "Proof" and end with some sort of delimiter like □ or "QED." The only purpose for these conventions is to clarify where proofs begin and end.

Method #2 - Prove the Contrapositive

An implication ("P IMPLIES Q") is logically equivalent to its *contrapositive*

$$\text{NOT}(Q) \text{ IMPLIES } \text{NOT}(P).$$

Proving one is as good as proving the other, and proving the contrapositive is sometimes easier than proving the original statement. If so, then you can proceed as follows:

1. Write, "We prove the contrapositive:" and then state the contrapositive.
2. Proceed as in Method #1.

Example

Theorem 1.5.2

If r is irrational, then $\sqrt{r}$ is also irrational.

A number is *rational* when it equals a quotient of integers — that is, if it equals m/n for some integers m and n. If it's not rational, then it's called *irrational*. So we must show that if r is *not* a ratio of integers, then $\sqrt{r}$ is also *not* a ratio of integers. That's pretty convoluted! We can eliminate both *not*'s and simplify the proof by using the contrapositive instead.

Proof. We prove the contrapositive: if $\sqrt{r}$ is rational, then r is rational.

Assume that $\sqrt{r}$ is rational. Then there exist integers m and n such that:

$$\sqrt{r} = \frac{m}{n}$$

Squaring both sides gives:

$$r = \frac{m^2}{n^2}$$

Since m^2 and n^2 are integers, r is also rational. ■

This page titled 1.5: Proving an Implication is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

1.6: Proving an "If and Only If"

Many mathematical theorems assert that two statements are logically equivalent; that is, one holds if and only if the other does. Here is an example that has been known for several thousand years:

> Two triangles have the same side lengths if and only if two side lengths and the angle between those sides are the same.

The phrase "if and only if" comes up so often that it is often abbreviated "iff."

Method #1: Prove Each Statement Implies the Other

The statement "P **IFF** Q" is equivalent to the two statements "P **IMPLIES** Q" and "Q **IMPLIES** P." So you can prove an "iff" by proving two implications:

1. Write, "We prove P implies Q and vice-versa."
2. Write, "First, we show P implies Q." Do this by one of the methods in Section 1.5.
3. Write, "Now, we show Q implies P." Again, do this by one of the methods in Section 1.5.

Method #2: Construct a Chain of Iffs

In order to prove that P is true iff Q is true:

1. Write, "We construct a chain of if-and-only-if implications."
2. Prove P is equivalent to a second statement which is equivalent to a third statement and so forth until you reach Q.

This method sometimes requires more ingenuity than the first, but the result can be a short, elegant proof.

Example

The *standard deviation* of a sequence of values $x_1, x_2, \ldots, x_n$ is defined to be:

$$\sqrt{\frac{(x_1-\mu)^2+(x_2-\mu)^2+\cdots+(x_n-\mu)^2}{n}} \tag{1.6.1}$$

where μ is the average or mean of the values:

$$\mu ::= \frac{x_1+x_2+\cdots+x_n}{n}$$

Theorem 1.6.1

The standard deviation of a sequence of values $x_1, x_2, \ldots, x_n$ is zero iff all the values are equal to the mean.

For example, the standard deviation of test scores is zero if and only if everyone scored exactly the class average.

Proof. We construct a chain of "iff" implications, starting with the statement that the standard deviation (1.6.1) is zero:

$$\sqrt{\frac{(x_1-\mu)^2+(x_2-\mu)^2+\cdots+(x_n-\mu)^2}{n}} = 0. \tag{1.6.2}$$

Now since zero is the only number whose square root is zero, equation (1.6.2) holds iff

$$(x_1-\mu)^2+(x_2-\mu)^2+\cdots+(x_n-\mu)^2 = 0. \tag{1.6.3}$$

Squares of real numbers are always nonnegative, so every term on the left hand side of equation (1.6.3) is nonnegative. This means that (1.6.3) holds iff

$$\textbf{Every term on the left hand side of (1.6.3) is zero.} \tag{1.6.4}$$

But a term $(x_i-\mu)^2$ is zero iff $x_i = \mu$, so (1.6.4) is true iff

$$\textbf{Every } x_i \textbf{ equals the mean.} \quad \blacksquare$$

This page titled 1.6: Proving an “If and Only If” is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

1.7: Proof by Cases

Breaking a complicated proof into cases and proving each case separately is a common, useful proof strategy. Here's an amusing example.

Let's agree that given any two people, either they have met or not. If every pair of people in a group has met, we'll call the group a *club*. If every pair of people in a group has not met, we'll call it a group of *strangers*.

Theorem

Every collection of 6 people includes a club of 3 people or a group of 3 strangers.

Proof

The proof is by case analysis[5]. Let x denote one of the six people. There are two cases:

1. Among 5 other people besides x, at least 3 have met x.
2. Among the 5 other people, at least 3 have not met x.

Now, we have to be sure that at least one of these two cases must hold,[6] but that's easy: we've split the 5 people into two groups, those who have shaken hands with x and those who have not, so one of the groups must have at least half the people.

Case 1: Suppose that at least 3 people did meet x.

This case splits into two subcases:

> **Case 1.1**: No pair among those people met each other. Then these people are a group of at least 3 strangers. The theorem holds in this subcase.
>
> **Case 1.2**: Some pair among those people have met each other. Then that pair, together with x, form a club of 3 people. So the theorem holds in this subcase.

This implies that the theorem holds in Case 1.

Case 2: Suppose that at least 3 people did not meet x.

This case also splits into two subcases:

> **Case 2.1**: Every pair among those people met each other. Then these people are a club of at least 3 people. So the theorem holds in this subcase.
>
> **Case 2.2**: Some pair among those people have not met each other. Then that pair, together with x, form a group of at least 3 strangers. So the theorem holds in this subcase.

This implies that the theorem also holds in Case 2, and therefore holds in all cases. ■

[5]Describing your approach at the outset helps orient the reader.

[6]Part of a case analysis argument is showing that you've covered all the cases. This is often obvious, because the two cases are of the form "P" and "not P." However, the situation above is not stated quite so simply.

This page titled 1.7: Proof by Cases is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

1.8: Proof by Contradiction

In a *proof by contradiction*, or *indirect proof*, you show that if a proposition were false, then some false fact would be true. Since a false fact by definition can't be true, the proposition must be true.

Proof by contradiction is *always* a viable approach. However, as the name suggests, indirect proofs can be a little convoluted, so direct proofs are generally preferable when they are available.

Method: In order to prove a proposition P by contradiction:

1. Write, "We use proof by contradiction."
2. Write, "Suppose P is false."
3. Deduce something known to be false (a logical contradiction).
4. Write, "This is a contradiction. Therefore, P must be true."

Example

We'll prove by contradiction that $\sqrt{2}$ is irrational. Remember that a number is *rational* if it is equal to a ratio of integers—for example, $3.5 = 7/2$ and $0.1111\cdots = 1/9$ are rational numbers.

Theorem 1.8.1

$\sqrt{2}$ *is irrational.*

Proof

We use proof by contradiction. Suppose the claim is false, and $\sqrt{2}$ is rational. Then we can write $\sqrt{2}$ as a fraction n/d in *lowest terms*.

Squaring both sides gives $2 = n^2/d^2$and so $2d^2 = n^2$. This implies that n is a multiple of 2 (see Problems 1.10 and 1.11). Therefore n^2 must be a multiple of 4. But since $2d^2 = n^2$, we know $2d^2$ is a multiple of 4 and so d^2 is a multiple of 2. This implies that d is a multiple of 2.

So, the numerator and denominator have 2 as a common factor, which contradicts the fact that n/d is in lowest terms. Thus, $\sqrt{2}$ must be irrational. ■

This page titled 1.8: Proof by Contradiction is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

1.9: Good Proofs in Practice

One purpose of a proof is to establish the truth of an assertion with absolute certainty, and mechanically checkable proofs of enormous length or complexity can accomplish this. But humanly intelligible proofs are the only ones that help someone understand the subject. Mathematicians generally agree that important mathematical results can't be fully understood until their proofs are understood. That is why proofs are an important part of the curriculum.

To be understandable and helpful, more is required of a proof than just logical correctness: a good proof must also be clear. Correctness and clarity usually go together; a well-written proof is more likely to be a correct proof, since mistakes are harder to hide.

In practice, the notion of proof is a moving target. Proofs in a professional research journal are generally unintelligible to all but a few experts who know all the terminology and prior results used in the proof. Conversely, proofs in the first weeks of a beginning course like 6.042 would be regarded as tediously long-winded by a professional mathematician. In fact, what we accept as a good proof later in the term will be different from what we consider good proofs in the first couple of weeks of 6.042. But even so, we can offer some general tips on writing good proofs:

State your game plan. A good proof begins by explaining the general line of reasoning, for example, "We use case analysis" or "We argue by contradiction."

Keep a linear flow. Sometimes proofs are written like mathematical mosaics, with juicy tidbits of independent reasoning sprinkled throughout. This is not good. The steps of an argument should follow one another in an intelligible order.

A proof is an essay, not a calculation. Many students initially write proofs the way they compute integrals. The result is a long sequence of expressions without explanation, making it very hard to follow. This is bad. A good proof usually looks like an essay with some equations thrown in. Use complete sentences.

Avoid excessive symbolism. Your reader is probably good at understanding words, but much less skilled at reading arcane mathematical symbols. Use words where you reasonably can.

Revise and simplify. Your readers will be grateful.

Introduce notation thoughtfully. Sometimes an argument can be greatly simplified by introducing a variable, devising a special notation, or defining a new term. But do this sparingly, since you're requiring the reader to remember all that new stuff. And remember to actually define the meanings of new variables, terms, or notations; don't just start using them!

Structure long proofs. Long programs are usually broken into a hierarchy of smaller procedures. Long proofs are much the same. When your proof needed facts that are easily stated, but not readily proved, those fact are best pulled out as preliminary lemmas. Also, if you are repeating essentially the same argument over and over, try to capture that argument in a general lemma, which you can cite repeatedly instead.

Be wary of the "obvious." When familiar or truly obvious facts are needed in a proof, it's OK to label them as such and to not prove them. But remember that what's obvious to you may not be—and typically is not—obvious to your reader.

Most especially, don't use phrases like "clearly" or "obviously" in an attempt to bully the reader into accepting something you're having trouble proving. Also, go on the alert whenever you see one of these phrases in someone else's proof.

Finish. At some point in a proof, you'll have established all the essential facts you need. Resist the temptation to quit and leave the reader to draw the "obvious" conclusion. Instead, tie everything together yourself and explain why the original claim follows.

Creating a good proof is a lot like creating a beautiful work of art. In fact, mathematicians often refer to really good proofs as being "elegant" or "beautiful." It takes a practice and experience to write proofs that merit such praises, but to get you started in the right direction, we will provide templates for the most useful proof techniques.

Throughout the text there are also examples of *bogus proofs*—arguments that look like proofs but aren't. Sometimes a bogus proof can reach false conclusions because of missteps or mistaken assumptions. More subtle bogus proofs reach correct conclusions, but do so in improper ways such as circular reasoning, leaping to unjustified conclusions, or saying that the hard part of the proof is "left to the reader." Learning to spot the flaws in improper proofs will hone your skills at seeing how each proof step follows logically from prior steps. It will also enable you to spot flaws in your own proofs.

The analogy between good proofs and good programs extends beyond structure. The same rigorous thinking needed for proofs is essential in the design of critical computer systems. When algorithms and protocols only "mostly work" due to reliance on hand-waving arguments, the results can range from problematic to catastrophic. An early example was the Therac 25, a machine that provided radiation therapy to cancer victims, but occasionally killed them with massive overdoses due to a software race condition. A more recent (August 2004) example involved a single faulty command to a computer system used by United and American Airlines that grounded the entire fleet of both companies—and all their passengers! It is a certainty that we'll all one day be at the mercy of critical computer systems designed by you and your classmates. So we really hope that you'll develop the ability to formulate rock-solid logical arguments that a system actually does what you think it does!

This page titled 1.9: Good Proofs in Practice is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

2: Well Ordering Principle

Every nonempty set of nonnegative integers has a *smallest* element.

This statement is known as The *Well Ordering Principle*. Do you believe it? Seems sort of obvious, right? But notice how tight it is: it requires a nonempty set—it's false for the empty set which has no smallest element because it has no elements at all. And it requires a set of *nonnegative* integers—it's false for the set of *negative* integers and also false for some sets of nonnegative *rationals*—for example, the set of positive rationals. So, the Well Ordering Principle captures something special about the nonnegative integers.

While the Well Ordering Principle may seem obvious, it's hard to see offhand why it is useful. But in fact, it provides one of the most important proof rules in discrete mathematics. In this chapter, we'll illustrate the power of this proof method with a few simple examples.

2.1: Well Ordering Proofs

2.2: Template for Well Ordering Proofs

2.3: Factoring into Primes

2.4: Well Ordered Sets

This page titled 2: Well Ordering Principle is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

2.1: Well Ordering Proofs

We actually have already taken the Well Ordering Principle for granted in proving that $\sqrt{2}$ is irrational. That proof assumed that for any positive integers m and n, the fraction m/n can be written in *lowest terms*, that is, in the form m'/n' where m' and n' are positive integers with no common prime factors. How do we know this is always possible?

Suppose to the contrary that there are positive integers m and n such that the fraction m/n cannot be written in lowest terms. Now let C be the set of positive integers that are numerators of such fractions. Then $m \in C$, so C is nonempty. Therefore, by Well Ordering, there must be a smallest integer, $m' \in C$. So by definition of C, there is an integer $n_0 > 0$ such that

$$\textbf{the fraction } m_0/n_0 \textbf{ cannot be written in lowest terms.}$$

This means that m_0 and n_0 must have a common prime factor, $p > 1$. But

$$\frac{m_0/p}{n_0/p} = \frac{m_0}{n_0},$$

so any way of expressing the left hand fraction in lowest terms would also work for m_0/n_0, which implies

$$\textbf{the fraction } \frac{m_0/p}{n_0/p} \textbf{ cannot be in written in lowest terms either.}$$

So by definition of C, the numerator, m_0/p, is in C. But $m_0/p < m_0$, which contradicts the fact that m_0 is the smallest element of C.

Since the assumption that C is nonempty leads to a contradiction, it follows that C must be empty. That is, that there are no numerators of fractions that can't be written in lowest terms, and hence there are no such fractions at all.

We've been using the Well Ordering Principle on the sly from early on!

This page titled 2.1: Well Ordering Proofs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

2.2: Template for Well Ordering Proofs

More generally, there is a standard way to use Well Ordering to prove that some property, $P(n)$ holds for every nonnegative integer, n. Here is a standard way to organize such a well ordering proof:

To prove that "$P(n)$ is true for all $n \in \mathbb{N}$" using the Well Ordering Principle:

- Define the set, C, of *counterexamples* to P being true. Specifically, define

$$C ::= \{n \in \mathbb{N} \mid \text{NOT}(P(n)) \text{ is true}\}.$$

The notation $\{n \mid Q(n)\}$ means "the set of all elements n for which $Q(n)$ is true." See Section 4.1.4.

- Assume for proof by contradiction that C is nonempty.
- By the Well Ordering Principle, there will be a smallest element, n, in C.
- Reach a contradiction somehow—often by showing that $P(n)$ is actually true or by showing that there is another member of C that is smaller than n. This is the open-ended part of the proof task.
- Conclude that C must be empty, that is, no counterexamples exist. ■

Summing the Integers

Let's use this template to prove

Theorem 2.2.1

$$1 + 2 + 3 + \cdots + n = n(n+1)/2 \tag{2.2.1}$$

for all nonnegative integers, n.

First, we'd better address a couple of ambiguous special cases before they trip us up:

- If $n = 1$, then there is only one term in the summation, and so $1 + 2 + 3 + \cdots + n$ is just the term 1. Don't be misled by the appearance of 2 and 3 or by the suggestion that 1 and n are distinct terms!
- If $n = 0$, then there are no terms at all in the summation. By convention, the sum in this case is 0.

So, while the three dots notation, which is called an *ellipsis*, is convenient, you have to watch out for these special cases where the notation is misleading. In fact, whenever you see an ellipsis, you should be on the lookout to be sure you understand the pattern, watching out for the beginning and the end.

We could have eliminated the need for guessing by rewriting the left side of (2.2.1) with *summation notation*:

$$\sum_{i=1}^{n} i \quad \text{or} \quad \sum_{1 \le i \le n} i.$$

Both of these expressions denote the sum of all values taken by the expression to the right of the sigma as the variable, i, ranges from 1 to n. Both expressions make it clear what (2.2.1) means when $n = 1$. The second expression makes it clear that when $n = 0$, there are no terms in the sum, though you still have to know the convention that a sum of no numbers equals 0 (the *product* of no numbers is 1, by the way).

OK, back to the proof:

Proof. By contradiction. Assume that Theorem 2.2.1 is *false*. Then, some nonnegative integers serve as *counterexamples* to it. Let's collect them in a set:

$$C ::= \{n \in \mathbb{N} \mid 1 + 2 + 3 + \cdots + n \neq \frac{n(n+1)}{2}\}.$$

Assuming there are counterexamples, C is a nonempty set of nonnegative integers. So, by the Well Ordering Principle, C has a minimum element, which we'll call c. That is, among the nonnegative integers, c is the *smallest counterexample* to equation (2.2.1).

Since c is the smallest counterexample, we know that (2.2.1) is false for $n = c$ but true for all nonnegative integers $n < c$. But (2.2.1) is true for $n = 0$, so $c > 0$. This means $c = 1$ is a nonnegative integer, and since it is less than c, equation (2.2.1) is true for $c - 1$. That is,

$$1 + 2 + 3 + \cdots + (c-1) = \frac{(c-1)c}{2}.$$

But then, adding c to both sides, we get

$$1 + 2 + 3 + \cdots + (c-1) + c = \frac{(c-1)c}{2} + c = \frac{c^2 - c + 2c}{2} = \frac{c(c+1)}{2}.$$

which means that (2.2.1) does hold for c, after all! This is a contradiction, and we are done. ■

This page titled 2.2: Template for Well Ordering Proofs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

2.3: Factoring into Primes

We've previously taken for granted the *Prime Factorization Theorem*, also known as the *Unique Factorization Theorem* and the *Fundamental Theorem of Arithmetic*, which states that every integer greater than one has a unique[1] expression as a product of prime numbers. This is another of those familiar mathematical facts which are taken for granted but are not really obvious on closer inspection. We'll prove the uniqueness of prime factorization in a later chapter, but well ordering gives an easy proof that every integer greater than one can be expressed as *some* product of primes.

Theorem 2.3.1

Every positive integer greater than one can be factored as a product of primes.

Proof

The proof is by well ordering.

Let C be the set of all integers greater than one that cannot be factored as a product of primes. We assume C is not empty and derive a contradiction.

If C is not empty, there is a least element, $n \in C$, by well ordering. The n can't be prime, because a prime by itself is considered a (length one) product of primes and no such products are in C.

So n must be a product of two integers a and b where $1 < a, b < n$. Since a and b are smaller than the smallest element in C, we know that $a, b \notin C$. In other words, a can be written as a product of primes $p_1 p_2 \cdots p_k$ and b as a product of primes $q_1 \cdots q_l$. Therefore, $n = p_1 \cdots p_k q_1 \cdots q_l$ can be written as a product of primes, contradicting the claim that $n \in C$. Our assumption that C is not empty must therefore be false. ■

[1]. . . unique up to the order in which the prime factors appear

This page titled 2.3: Factoring into Primes is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

2.4: Well Ordered Sets

A set of numbers is *well ordered* when each of its nonempty subsets has a minimum element. The Well Ordering Principle says, of course, that the set of nonnegative integers is well ordered, but so are lots of other sets, such as every finite set, or the sets $r\mathbb{N}$ of numbers of the form rn, where r is a positive real number and $n \in \mathbb{N}$.

Well ordering commonly comes up in computer science as a method for proving that computations won't run forever. The idea is to assign a value to the successive steps of a computation so that the values get smaller at every step. If the values are all from a well ordered set, then the computation can't run forever, because if it did, the values assigned to its successive steps would define a subset with no minimum element. You'll see several examples of this technique applied in Section 5.4 to prove that various state machines will eventually terminate.

Notice that a set may have a minimum element but not be well ordered. The set of nonnegative rational numbers is an example: it has a minimum element, zero, but it also has nonempty subsets that don't have minimum elements—the *positive* rationals, for example.

The following theorem is a tiny generalization of the Well Ordering Principle.

Theorem 2.4.1

For any nonnegative integer, n, the set of integers greater than or equal to $-n$ is well ordered.

This theorem is just as obvious as the Well Ordering Principle, and it would be harmless to accept it as another axiom. But repeatedly introducing axioms gets worrisome after a while, and it's worth noticing when a potential axiom can actually be proved. We can easily prove Theorem 2.4.1 using the Well Ordering Principle:

Proof

Let S be any nonempty set of integers $\geq -n$. Now add n to each of the elements in S; let's call this new set $S + n$. Now $S + n$ is a nonempty set of *nonnegative* integers, and so by the Well Ordering Principle, it has a minimum element, m. But then it's easy to see that $m - n$ is the minimum element of S. ■

The definition of well ordering states that *every* subset of a well ordered set is well ordered, and this yields two convenient, immediate corollaries of Theorem 2.4.1:

Definition 2.4.2

A lower bound (respectively, *upper bound*) for a set, S, of real numbers is a number, b, such that $b \leq s$ (respectively, $b \geq s$) for every $s \in S$.

Note that a lower or upper bound of set S is not required to be in the set.

Corollary 2.4.3. *Any set of integers with a lower bound is well ordered.*

Proof. A set of integers with a lower bound $b \in \mathbb{R}$ will also have the integer $n = \lfloor b \rfloor$ as a lower bound, where $\lfloor b \rfloor$, called the floor of b, is gotten by rounding down b to the nearest integer. So Theorem 2.4.1 implies the set is well ordered. ■

Corollary 2.4.4. *Any nonempty set of integers with an upper bound has a maximum element.*

Proof. Suppose a set, S, of integers has an upper bound $b \in \mathbb{R}$. Now multiply each element of S by -1; let's call this new set of elements $-S$. Now, of course, $-b$ is a lower bound of $-S$. So $-S$ has a minimum element $-m$ by Corollary 2.4.3. But then it's easy to see that m is the maximum element of S. ■

A Different Well Ordered Set (Optional)

Another example of a well ordered set of numbers is the set $\mathbb{F}$ of fractions that can be expressed in the form $n/(n+1)$:

$$\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \cdots, \frac{n}{n+1}, \cdots$$

The minimum element of any nonempty subset of $\mathbb{F}$ is simply the one with the minimum numerator when expressed in the form $n/(n+1)$.

Now we can define a very different well ordered set by adding nonnegative integers to numbers in $\mathbb{F}$. That is, we take all the numbers of the form $n+f$ where n is a nonnegative integer and f is a number in $\mathbb{F}$. Let's call this set of numbers—you guessed it—$\mathbb{N}+\mathbb{F}$. There is a simple recipe for finding the minimum number in any nonempty subset of $\mathbb{N}+\mathbb{F}$, which explains why this set is well ordered:

Lemma 2.4.5. $\mathbb{N}+\mathbb{F}$ *is well ordered.*

Proof. Given any nonempty subset, S, of $\mathbb{N}+\mathbb{F}$, look at all the nonnegative integers, n, such that $n+f$ is in S for some $f \in \mathbb{F}$. This is a nonempty set nonnegative integers, so by the WOP, there is a minimum one; call it n_s.

By definition of n_s, there is some $f \in \mathbb{F}$ such that $n_S + f$ is in the set S. So the set all fractions f such that $n_S + f \in S$ is a nonempty subset of $\mathbb{F}$, and since $\mathbb{F}$ is well ordered, this nonempty set contains a minimum element; call it f_S. Now it easy to verify that $n_S + f_S$ is the minimum element of S (Problem 2.14). ■

The set $\mathbb{N}+\mathbb{F}$ is different from the earlier examples. In all the earlier examples, each element was greater than only a finite number of other elements. In $\mathbb{N}+\mathbb{F}$, every element greater than or equal to 1 can be the first element in strictly decreasing sequences of elements of arbitrary finite length. For example, the following decreasing sequences of elements in $\mathbb{N}+\mathbb{F}$ all start with 1:

$$\begin{array}{l} 1, 0. \\ 1, \frac{1}{2}, 0. \\ 1, \frac{2}{3}, \frac{1}{2}, 0. \\ 1, \frac{3}{4}, \frac{2}{3}, \frac{1}{2}, 0. \\ \vdots \end{array} \tag{2.4.1}$$

Nevertheless, since $\mathbb{N}+\mathbb{F}$ is well ordered, it is impossible to find an infinite decreasing sequence of elements in $\mathbb{N}+\mathbb{F}$, because the set of elements in such a sequence would have no minimum.

2.4: Well Ordered Sets is shared under a not declared license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

3: Logical Formulas

It is amazing that people manage to cope with all the ambiguities in the English language. Here are some sentences that illustrate the issue:

- "You may have cake, or you may have ice cream."
- "If pigs can fly, then you can understand the Chebyshev bound."
- "If you can solve any problem we come up with, then you get an ***A*** for the course."
- "Every American has a dream."

What *precisely* do these sentences mean? Can you have both cake and ice cream or must you choose just one dessert? Pigs can't fly, so does the second sentence say anything about your understanding the Chebyshev bound? If you can solve some problems we come up with, can you get an ***A*** for the course? And if you can't solve a single one of the problems, does it mean you can't get an ***A***? Finally, does the last sentence imply that all Americans have the same dream—say of owning a house—or might different Americans have different dreams—say, Eric dreams of designing a killer software application, Tom of being a tennis champion, Albert of being able to sing?

Some uncertainty is tolerable in normal conversation. But when we need to formulate ideas precisely—as in mathematics and programming—the ambiguities inherent in everyday language can be a real problem. We can't hope to make an exact argument if we're not sure exactly what the statements mean. So before we start into mathematics, we need to investigate the problem of how to talk about mathematics.

To get around the ambiguity of English, mathematicians have devised a special language for talking about logical relationships. This language mostly uses ordinary English words and phrases such as "or," "implies," and "for all." But mathematicians give these words precise and unambiguous definitions.

Surprisingly, in the midst of learning the language of logic, we'll come across the most important open problem in computer science—a problem whose solution could change the world.

3.1: Propositions from Propositions

3.2: Propositional Logic in Computer Programs

3.3: Equivalence and Validity

3.4: The Algebra of Propositions

3.5: The SAT Problem

3.6: Predicate Formulas

This page titled 3: Logical Formulas is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

3.1: Propositions from Propositions

In English, we can modify, combine, and relate propositions with words such as “not,” “and,” “or,” “implies,” and “if-then.” For example, we can combine three propositions into one like this:

If all humans are mortal **and** all Greeks are human, **then** all Greeks are mortal.

For the next while, we won’t be much concerned with the internals of propositions— whether they involve mathematics or Greek mortality—but rather with how propositions are combined and related. So, we’ll frequently use variables such as P and Q in place of specific propositions such as “All humans are mortal” and “2 + 3 = 5.” The understanding is that these *propositional variables*, like propositions, can take on only the values **T** (true) and **F** (false). Propositional variables are also called *Boolean variables* after their inventor, the nineteenth century mathematician George—you guessed it—Boole.

NOT, AND, and OR

Mathematicians use the words **NOT**, **AND**, and **OR** for operations that change or combine propositions. The precise mathematical meaning of these special words can be specified by *truth tables*. For example, if P is a proposition, then so is “NOT(P)” and the truth value of the proposition “NOT(P)” is determined by the truth value of P according to the following truth table:

P	NOT(P)
T	F
F	T

The first row of the table indicates that when proposition P is true, the proposition “NOT(P)” is false. The second line indicates that when P is false, “NOT(P)” is true. This is probably what you would expect.

In general, a truth table indicates the true/false value of a proposition for each possible set of truth values for the variables. For example, the truth table for the proposition “P AND Q” has four lines, since there are four settings of truth values for the two variables:

P	Q	P AND Q
T	T	T
T	F	F
F	T	F
F	F	F

According to this table, the proposition “P AND Q” is true only when P and Q are both true. This is probably the way you ordinarily think about the word “and.”

There is a subtlety in the truth table for “P OR Q”:

P	Q	P OR Q
T	T	T
T	F	T
F	T	T
F	F	F

The first row of this table says that “P OR Q” is true even if *both* P and Q are true. This isn’t always the intended meaning of “or” in everyday speech, but this is the standard definition in mathematical writing. So if a mathematician says, “You may have cake, or you may have ice cream,” he means that you *could* have both.

If you want to exclude the possibility of having both cake *and* ice cream, you should combine them with the *exclusive-or* operation, $textXOR$:

P	Q	P XOR Q
T	T	F
T	F	T
F	T	T
F	F	F

IMPLIES

The combining operation with the least intuitive technical meaning is "implies." Here is its truth table, with the lines labeled so we can refer to them later.

P	Q	P IMPLIES Q	
T	T	T	(tt)
T	F	F	(tf)
F	T	T	(ft)
F	F	T	(ff)

The truth table for implications can be summarized in words as follows:

An implication is true exactly when the if-part is false or the then-part is true.

This sentence is worth remembering; a large fraction of all mathematical statements are of the if-then form!

Let's experiment with this definition. For example, is the following proposition true or false?

"If Goldbach's Conjecture is true, then $x^2 \geq 0$ for every real number x."

Now, we already mentioned that no one knows whether Goldbach's Conjecture, Proposition 1.1.8, is true or false. But that doesn't prevent you from answering the question! This proposition has the form P IMPLIES Q where the *hypothesis*, P, is "Goldbach's Conjecture is true" and the *conclusion*, Q, is "$x^2 \geq 0$ for every real number x." Since the conclusion is definitely true, we're on either line (tt) or line (ft) of the truth table. Either way, the proposition as a whole is *true*!

One of our original examples demonstrates an even stranger side of implications.

"If pigs fly, then you can understand the Chebyshev bound."

Don't take this as an insult; we just need to figure out whether this proposition is true or false. Curiously, the answer has *nothing* to do with whether or not you can understand the Chebyshev bound. Pigs do not fly, so we're on either line (ft) or line (ff) of the truth table. In both cases, the proposition is *true*!

In contrast, here's an example of a false implication:

"If the moon shines white, then the moon is made of white cheddar."

Yes, the moon shines white. But, no, the moon is not made of white cheddar cheese. So we're on line (tf) of the truth table, and the proposition is false.

False Hypotheses

It often bothers people when they first learn that implications which have false hypotheses are considered to be true. But implications with false hypotheses hardly ever come up in ordinary settings, so there's not much reason to be bothered by whatever truth assignment logicians and mathematicians choose to give them.

There are, of course, good reasons for the mathematical convention that implications are true when their hypotheses are false. An illustrative example is a system specification (see Problem 3.12) which consisted of a series of, say, a dozen rules,

if C_i : the system sensors are in condition i, then A_i : the system takes action i,

or more concisely,

$$C_i \text{ IMPLIES } A_i$$

for $1 \leq i \leq 12$. Then the fact that the system obeys the specification would be expressed by saying that the AND

$$[C_1 \text{ IMPLIES } A_1] \text{ AND } [C_2 \text{ IMPLIES } A_2] \text{ AND} \cdots \text{ AND } [C_{12} \text{ IMPLIES } A_{12}] \tag{3.1.1}$$

of these rules was always true.

For example, suppose only conditions C_2 and C_5 are true, and the system indeed takes the specified actions A_2 and A_5. This means that in this case the system is behaving according to specification, and accordingly we want the formula (3.1.1) to come out true. Now the implications C_2 IMPLIES A_2 and C_5 IMPLIES A_5 are both true because both their hypotheses and their conclusions are

true. But in order for (3.1.1) to be true, we need all the other implications with the false hypotheses C_i for $i \neq 2, 5$ to be true. This is exactly what the rule for implications with false hypotheses accomplishes.

If and Only If

Mathematicians commonly join propositions in one additional way that doesn't arise in ordinary speech. The proposition "P if and only if Q" asserts that P and Q have the same truth value. Either both are true or both are false.

P	Q	P IFF Q
T	T	T
T	F	F
F	T	F
F	F	T

For example, the following if-and-only-if statement is true for every real number x:

$$x^2 - 4 \geq 0 \text{ IFF } |x| \geq 2.$$

For some values of x, *both* inequalities are true. For other values of x, *neither* inequality is true. In every case, however, the IFF proposition as a whole is true.

This page titled 3.1: Propositions from Propositions is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

3.2: Propositional Logic in Computer Programs

Propositions and logical connectives arise all the time in computer programs. For example, consider the following snippet, which could be either C, C++, or Java:

Misplaced '&'

Java uses the symbol || for "**OR**," and the symbol && for "**AND**." The *further instructions* are carried out only if the proposition following the word *if* is true. On closer inspection, this big expression is built from two simpler propositions.

Let A be the proposition that $x > 0$, and let B be the proposition that $y > 100$. Then we can rewrite the condition as

$$A \text{ OR } (\text{NOT}(A) \text{ AND } B) \tag{3.2.1}$$

Truth Table Calculation

A truth table calculation reveals that the more complicated expression 3.2 always has the same truth value as

$$A \text{ OR } B. \tag{3.2.2}$$

We begin with a table with just the truth values of A and B:

A	B	A	OR	(NOT(A)	AND	B)	A OR B
T	T						
T	F						
F	T						
F	F						

These values are enough to fill in two more columns:

A	B	A	OR	(NOT(A)	AND	B)	A OR B
T	T			F			T
T	F			F			T
F	T			T			T
F	F			T			F

Now we have the values needed to fill in the **AND** column:

A	B	A	OR	(NOT(A)	AND	B)	A OR B
T	T			F	F		T
T	F			F	F		T
F	T			T	T		T
F	F			T	F		F

and this provides the values needed to fill in the remaining column for the first **OR**:

A	B	A	OR	(NOT(A)	AND	B)	A OR B
T	T		T	F	F		T
T	F		T	F	F		T
F	T		T	T	T		T
F	F		F	T	F		F

Expressions whose truth values always match are called *equivalent*. Since the two emphasized columns of truth values of the two expressions are the same, they are equivalent. So we can simplify the code snippet without changing the program's behavior by replacing the complicated expression with an equivalent simpler one:

$$\begin{array}{l} \textbf{if } (x > 0 || y > 100) \\ \quad \vdots \\ \quad \textit{(further instructions)} \end{array} \tag{3.2.3}$$

The equivalence of (3.2.1) and (3.2.2) can also be confirmed reasoning by cases:

A is **T**. An expression of the form (**T OR** anything) is equivalent to **T**. Since A is **T** both (3.2.1) and (3.2.2) in this case are of this form, so they have the same truth value, namely, **T**.

A is **F**. An expression of the form (**F OR** *anything*) will have same truth value as *anything*. Since A is **F**, (3.2.2) has the same truth value as B.

An expression of the form (**T AND** *anything*) is equivalent to *anything*, as is any expression of the form **F OR** *anything*. So in this case A **OR** (**NOT**(A) **AND** B) is equivalent to (**NOT**(A) **AND** B), which in turn is equivalent to B.

Therefore both (3.2.1) and (3.2.2) will have the same truth value in this case, namely, the value of B.

Simplifying logical expressions has real practical importance in computer science. Expression simplification in programs like the one above can make a program easier to read and understand. Simplified programs may also run faster, since they require fewer operations. In hardware, simplifying expressions can decrease the number of logic gates on a chip because digital circuits can be described by logical formulas (see Problems 3.5 and 3.6). Minimizing the logical formulas corresponds to reducing the number of gates in the circuit. The payoff of gate minimization is potentially enormous: a chip with fewer gates is smaller, consumes less power, has a lower defect rate, and is cheaper to manufacture.

Cryptic Notation

Java uses symbols like “&&” and “||” in place of **AND** and **OR**. Circuit designers use "·" and "+," and actually refer to **AND** as a product and **OR** as a sum. Mathematicians use still other symbols, given in the table below.

English	**Symbolic Notation**
NOT(P)	$\neg P$ (**alternatively**, $\overline{P}$)
P **AND** Q	$P \wedge Q$
P **OR** Q	$P \vee Q$
P **IMPLIES** Q	$P \longrightarrow Q$
If P **then** Q	$P \longrightarrow Q$
P **IFF** Q	$P \longleftrightarrow Q$
P **XOR** Q	$P \oplus Q$

For example, using this notation, “If P **AND NOT** (Q), then R” would be written:

$$P \wedge \overline{Q} \rightarrow R.$$

The mathematical notation is concise but cryptic. Words such as “**AND**” and “**OR**” are easier to remember and won’t get confused with operations on numbers. We will often use $\overline{P}$ as an abbreviation for **NOT**(P), but aside from that, we mostly stick to the words —except when formulas would otherwise run off the page.

This page titled 3.2: Propositional Logic in Computer Programs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

3.3: Equivalence and Validity

Implications and Contrapositives

Do these two sentences say the same thing?

If I am hungry, then I am grumpy.

If I am not grumpy, then I am not hungry.

We can settle the issue by recasting both sentences in terms of propositional logic. Let P be the proposition "I am hungry" and Q be "I am grumpy." The first sentence says "P IMPLIES Q" and the second says "NOT(Q) IMPLIES NOT(P)." Once more, we can compare these two statements in a truth table:

P	Q	(P IMPLIES Q)	(NOT(Q)	IMPLIES	NOT(P))
T	T	T	F	T	F
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

Sure enough, the highlighted columns showing the truth values of these two statements are the same. A statement of the form "NOT(Q) IMPLIES NOT(P)" is called the *contrapositive* of the implication "P IMPLIES Q." The truth table shows that an implication and its contrapositive are equivalent—they are just different ways of saying the same thing.

In contrast, the *converse* of "P IMPLIES Q" is the statement "Q IMPLIES P." The converse to our example is:

If I am grumpy, then I am hungry.

This sounds like a rather different contention, and a truth table confirms this suspicion:

P	Q	P IMPLIES Q	Q IMPLIES P
T	T	T	T
T	F	F	T
F	T	T	F
F	F	T	T

Now the highlighted columns differ in the second and third row, confirming that an implication is generally *not* equivalent to its converse.

One final relationship: an implication and its converse together are equivalent to an iff statement, specifically, to these two statements together. For example,

If I am grumpy then I am hungry, and if I am hungry then I am grumpy.

are equivalent to the single statement:

I am grumpy iff I am hungry.

Once again, we can verify this with a truth table.

P	Q	(P IMPLIES Q)	AND	(Q IMPLIES P)	(P IFF Q)
T	T	T	T	T	T
T	F	F	F	T	F
F	T	T	F	F	F
F	F	T	T	T	T

The fourth column giving the truth values of

(P IMPLIES Q) AND (Q IMPLIES P)

is the same as the sixth column giving the truth values of P IFF Q, which confirms that the AND of the implications is equivalent to the IFF statement.

Validity and Satisfiability

A *valid* formula is one which is *always* true, no matter what truth values its variables may have. The simplest example is

$$P \text{ OR NOT } (P).$$

You can think about valid formulas as capturing fundamental logical truths. For example, a property of implication that we take for granted is that if one statement implies a second one, and the second one implies a third, then the first implies the third. The following valid formula confirms the truth of this property of implication.

$$[(P \text{ IMPLIES } Q) \text{ AND } (Q \text{ IMPLIES } R)] \text{ IMPLIES } (P \text{ IMPLIES } R).$$

Equivalence of formulas is really a special case of validity. Namely, statements F and G are equivalent precisely when the statement (F IFF G) is valid. For example, the equivalence of the expressions (3.2.1) and (3.2.2) means that

$$(A \text{ OR } B) \text{ IFF } (A \text{ OR NOT } (A) \text{ AND } B))$$

is valid. Of course, validity can also be viewed as an aspect of equivalence. Namely, a formula is valid iff it is equivalent to **T**.

A *satisfiable* formula is one which can *sometimes* be true—that is, there is some assignment of truth values to its variables that makes it true. One way satisfiability comes up is when there are a collection of system specifications. The job of the system designer is to come up with a system that follows all the specs. This means that the **AND** of all the specs must be satisfiable or the designer's job will be impossible (see Problem 3.12).

There is also a close relationship between validity and satisfiability: a statement P is satisfiable iff its negation **NOT**(P) is *not* valid.

This page titled 3.3: Equivalence and Validity is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

3.4: The Algebra of Propositions

Propositions in Normal Form

Every propositional formula is equivalent to a "sum-of-products" or *disjunctive form*. More precisely, a disjunctive form is simply an OR of **AND**-terms, where each **AND**-terms is an **AND** of variables or negations of variables, for example,

$$(A \text{ AND } B) \text{ OR } (A \text{ AND } C). \tag{3.4.1}$$

You can read a disjunctive form for any propositional formula directly from its truth table. For example, the formula

$$A \text{ AND } (B \text{ OR } C) \tag{3.4.2}$$

has truth table:

A	B	C	A AND $(B$ OR $C)$
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

The formula (3.4.2) is true in the first row when $A, B,$ and C are all true, that is, where A AND B AND C is true. It is also true in the second row where A AND B AND $\overline{C}$ is true, and in the third row when A AND $\overline{B}$ AND C is true, and that's all. So (3.4.2) is true exactly when

$$(A \text{ AND } B \text{ AND } C) \text{ OR } (A \text{ AND } B \text{ AND } \overline{C}) \text{ OR } (A \text{ AND } \overline{B} \text{ AND } C) \tag{3.4.3}$$

is true.

Theorem 3.4.1

[Distributive Law of **AND** *over* **OR***]*

A AND $(B$ OR $C)$ *is equivalent to* $(A$ AND $B)$ OR $(A$ AND $C)$.

Theorem 3.4.1 is called a *distributive* law because of its resemblance to the distributivity of products over sums in arithmetic.

Similarly, we have (Problem 3.10):

Theorem 3.4.2

[Distributive Law of **OR** *over* **AND***]*

A OR $(B$ AND $C)$ *is equivalent to* $(A$ OR $B)$ AND $(A$ OR $C)$.

Note the contrast between Theorem 3.4.2 and arithmetic, where sums do not distribute over products.

The expression (3.4.3) is a disjunctive form where each **AND**-term is an **AND** of *every one* of the variables or their negations in turn. An expression of this form is called a *disjunctive normal form* (*DNF*). A DNF formula can often be simplified into a smaller disjunctive form. For example, the DNF (3.4.3) further simplifies to the equivalent disjunctive form (3.4.1) above.

Applying the same reasoning to the **F** entries of a truth table yields a *conjunctive form* for any formula—an **AND** of **OR**-terms in which the **OR**-terms are **OR**'s only of variables or their negations. For example, formula (3.4.2) is false in the fourth row of its truth table (3.4.1) where A is **T**, B is **F** and C is **F**. But this is exactly the one row where ($\overline{A}$ OR B OR C) is **F**! Likewise, the (3.4.2) is false in the fifth row which is exactly where (A OR $\overline{B}$ OR $\overline{C}$) is **F**. This means that (3.4.2) will be **F** whenever the **AND** of these two

OR-terms is false. Continuing in this way with the OR-terms corresponding to the remaining three rows where (3.4.2) is false, we get a *conjunctive normal form (CNF)* that is equivalent to (3.4.2), namely,

$$(\overline{A} \text{ OR } B \text{ OR } C) \text{ AND } (A \text{ OR } \overline{B} \text{ OR } \overline{C}) \text{ AND } (A \text{ OR } \overline{B} \text{ OR } C) \text{ AND } (A \text{ OR } B \text{ OR } \overline{C}) \text{ AND } (A \text{ OR } B \text{ OR } C)$$

The methods above can be applied to any truth table, which implies

Theorem 3.4.3

Every propositional formula is equivalent to both a disjunctive normal form and a conjunctive normal form.

Proving Equivalences

A check of equivalence or validity by truth table runs out of steam pretty quickly: a proposition with n variables has a truth table with 2^n lines, so the effort required to check a proposition grows exponentially with the number of variables. For a proposition with just 30 variables, that's already over a billion lines to check!

An alternative approach that *sometimes* helps is to use algebra to prove equivalence. A lot of different operators may appear in a propositional formula, so a useful first step is to get rid of all but three: AND, OR, and NOT. This is easy because each of the operators is equivalent to a simple formula using only these three. For example, A IMPLIES B is equivalent to NOT (A) OR B. Formulas using only AND, OR, and NOT for the remaining operators are left to Problem 3.13.

We list below a bunch of equivalence axioms with the symbol "$\longleftrightarrow$" between equivalent formulas. These axioms are important because they are all that's needed to prove every possible equivalence. We'll start with some equivalences for AND's that look like the familiar ones for multiplication of numbers:

$$A \text{ AND } B \longleftrightarrow B \text{ AND } A \qquad \text{(commutativity of AND)} \tag{3.4.4}$$

$$(A \text{ AND } B) \text{ AND } C \longleftrightarrow A \text{ AND } (B \text{ AND } C) \qquad \text{(associativity of AND)} \tag{3.4.5}$$

$$\mathbf{T} \text{ AND } A \longleftrightarrow A \qquad \text{(identity of AND)}$$

$$\mathbf{F} \text{ AND } A \longleftrightarrow \mathbf{F} \qquad \text{(zero for AND)}$$

Three axioms that don't directly correspond to number properties are

$$A \text{ AND } A \longleftrightarrow A \qquad \text{(idempotence for AND)}$$

$$A \text{ AND } \overline{A} \longleftrightarrow \mathbf{F} \qquad \text{(contradiction for AND)} \tag{3.4.6}$$

$$\text{NOT } (\overline{A}) \longleftrightarrow A \qquad \text{(double negation)} \tag{3.4.7}$$

It is associativity (3.4.5) that justifies writing A AND B AND C without specifying whether it is parenthesized as A AND (B AND C) or (A AND B) AND C. Both ways of inserting parentheses yield equivalent formulas.

There are a corresponding set of equivalences for OR which we won't bother to list, except for the OR rule corresponding to contradiction for AND (3.4.6):

$$A \text{ OR } \overline{A} \longleftrightarrow \mathbf{T} \qquad \text{(validity for OR)} \tag{3.4.8}$$

Finally, there are *DeMorgan's Laws* which explain how to distribute NOT's over AND's and OR's:

$$\text{NOT}(A \text{ AND } B) \longleftrightarrow \overline{A} \text{ OR } \overline{B} \qquad \text{(DeMorgan for AND)} \tag{3.4.9}$$

$$\text{NOT}(A \text{ OR } B) \longleftrightarrow \overline{A} \text{ AND } \overline{B} \qquad \text{(DeMorgan for OR)} \tag{3.4.10}$$

All of these axioms can be verified easily with truth tables.

These axioms are all that's needed to convert any formula to a disjunctive normal form. We can illustrate how they work by applying them to turn the negation of formula (3.4.2),

$$\text{NOT}((A \text{ AND } B) \text{ OR } (A \text{ AND } C)). \tag{3.4.11}$$

into disjunctive normal form.

We start by applying DeMorgan's Law for OR (3.4.10) to (3.4.11) in order to move the NOT deeper into the formula. This gives

$$\text{NOT}(A \text{ AND } B) \text{ AND NOT } (A \text{ AND } C).$$

Now applying Demorgan's Law for AND (3.4.9) to the two innermost AND-terms, gives

$$(\overline{A} \text{ OR } \overline{B}) \text{ AND } (\overline{A} \text{ OR } \overline{C}). \tag{3.4.12}$$

At this point NOT only applies to variables, and we won't need Demorgan's Laws any further.

Now we will repeatedly apply The Distributivity of AND over OR (Theorem 3.4.1) to turn (3.4.12) into a disjunctive form. To start, we'll distribute $(\overline{A} \text{ OR } \overline{B})$ over AND to get

$$((\overline{A} \text{ OR } \overline{B}) \text{ AND } \overline{A}) \text{ OR } ((\overline{A} \text{ OR } \overline{B}) \text{ AND } \overline{C}).$$

Using distributivity over both AND's we get

$$((\overline{A} \text{ AND } \overline{A}) \text{ OR } (\overline{B} \text{ AND } \overline{A})) \text{ OR } ((\overline{A} \text{ AND } \overline{C}) \text{ OR } (\overline{B} \text{ AND } \overline{C})).$$

By the way, we've implicitly used commutativity (3.7) here to justify distributing over an AND from the right. Now applying idempotence to remove the duplicate occurrence of A we get

$$(\overline{A} \text{ OR } (\overline{B} \text{ AND } \overline{A})) \text{ OR } ((\overline{A} \text{ AND } \overline{C}) \text{ OR } (\overline{B} \text{ AND } \overline{C})).$$

Associativity now allows dropping the parentheses around the terms being OR'd to yield the following disjunctive form for (3.4.11):

$$\overline{A} \text{ OR } (\overline{B} \text{ AND } \overline{A}) \text{ OR } (\overline{A} \text{ AND } \overline{C}) \text{ OR } (\overline{B} \text{ AND } \overline{C}). \tag{3.4.13}$$

The last step is to turn each of these AND-terms into a disjunctive normal form with all three variables $A, B,$ and C. We'll illustrate how to do this for the second AND-term $(\overline{B} \text{ AND } \overline{A})$. This term needs to mention C to be in normal form. To introduce C, we use validity for OR and identity for AND to conclude that

$$(\overline{B} \text{ AND } \overline{A}) \leftrightarrow (\overline{B} \text{ AND } \overline{A}) \text{ AND } (C \text{ OR } \overline{C}).$$

Now distributing $(\overline{B} \text{ AND } \overline{A})$ over the OR yields the disjunctive normal form

$$(\overline{B} \text{ AND } \overline{A} \text{ AND } C) \text{ OR } (\overline{B} \text{ AND } \overline{A} \text{ AND } \overline{C}).$$

Doing the same thing to the other AND-terms in (3.4.13) finally gives a disjunctive normal form for (3.4.2):

$$\begin{array}{l}(\overline{A} \text{ AND } B \text{ AND } C) \text{ OR } (\overline{A} \text{ AND } B \text{ AND } \overline{C}) \text{ OR} \\ (\overline{A} \text{ AND } \overline{B} \text{ AND } C) \text{ OR } (\overline{A} \text{ AND } \overline{B} \text{ AND } \overline{C}) \text{ OR} \\ (\overline{B} \text{ AND } \overline{A} \text{ AND } C) \text{ OR } (\overline{B} \text{ AND } \overline{A} \text{ AND } \overline{C}) \text{ OR} \\ (\overline{A} \text{ AND } \overline{C} \text{ AND } B) \text{ OR } (\overline{A} \text{ AND } \overline{C} \text{ AND } \overline{B}) \text{ OR} \\ (\overline{B} \text{ AND } \overline{C} \text{ AND } A) \text{ OR } (\overline{B} \text{ AND } \overline{C} \text{ AND } \overline{A}).\end{array} \tag{3.4.14}$$

Using commutativity to sort the term and OR-idempotence to remove duplicates, finally yields a unique sorted DNF:

$$\begin{array}{l}(A \text{ AND } \overline{B} \text{ AND } \overline{C}) \text{ OR} \\ (\overline{A} \text{ AND } B \text{ AND } C) \text{ OR} \\ (\overline{A} \text{ AND } B \text{ AND } \overline{C}) \text{ OR} \\ (\overline{A} \text{ AND } \overline{B} \text{ AND } C) \text{ OR} \\ (\overline{A} \text{ AND } \overline{B} \text{ AND } \overline{C}).\end{array} \tag{3.4.15}$$

This example illustrates a strategy for applying these equivalences to convert any formula into disjunctive normal form, and conversion to conjunctive normal form works similarly, which explains:

Theorem 3.4.4

Any propositional formula can be transformed into disjunctive normal form or a conjunctive normal form using the equivalences listed above.

What has this got to do with equivalence? That's easy: to prove that two formulas are equivalent, convert them both to disjunctive normal form over the set of variables that appear in the terms. Then use commutativity to sort the variables and **AND**-terms so they all appear in some standard order. We claim the formulas are equivalent iff they have the same sorted disjunctive normal form. This is obvious if they do have the same disjunctive normal form. But conversely, the way we read off a disjunctive normal form from a truth table shows that two different sorted **DNF**'s over the same set of variables correspond to different truth tables and hence to inequivalent formulas. This proves

Theorem 3.4.5

(Completeness of the propositional equivalence axioms). *Two propositional formula are equivalent iff they can be proved equivalent using the equivalence axioms listed above.*

The benefit of the axioms is that they leave room for ingeniously applying them to prove equivalences with less effort than the truth table method. Theorem 3.4.5 then adds the reassurance that the axioms are guaranteed to prove every equivalence, which is a great punchline for this section. But we don't want to mislead you: it's important to realize that using the strategy we gave for applying the axioms involves essentially the same effort it would take to construct truth tables, and there is no guarantee that applying the axioms will generally be any easier than using truth tables.

This page titled 3.4: The Algebra of Propositions is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

3.5: The SAT Problem

Determining whether or not a more complicated proposition is satisfiable is not so easy. How about this one?

$$(P \text{ OR } Q \text{ OR } R) \text{ AND } (\overline{P} \text{ OR } \overline{Q}) \text{ AND } (\overline{P} \text{ OR } \overline{R}) \text{ AND } (\overline{R} \text{ OR } \overline{Q})$$

The general problem of deciding whether a proposition is satisfiable is called *SAT*. One approach to SAT is to construct a truth table and check whether or not a **T** ever appears, but as with testing validity, this approach quickly bogs down for formulas with many variables because truth tables grow exponentially with the number of variables.

Is there a more efficient solution to SAT? In particular, is there some brilliant procedure that determines SAT in a number of steps that grows *polynomially* — like n^2 or n^{14} — instead of *exponentially* — 2^n — whether any given proposition of size n is satisfiable or not? No one knows. And an awful lot hangs on the answer.

The general definition of an “efficient” procedure is one that runs in *polynomial time*, that is, that runs in a number of basic steps bounded by a polynomial in s, where s is the size of an input. It turns out that an efficient solution to SAT would immediately imply efficient solutions to many other important problems involving scheduling, routing, resource allocation, and circuit verification across multiple disciplines including programming, algebra, finance, and political theory. This would be wonderful, but there would also be worldwide chaos. Decrypting coded messages would also become an easy task, so online financial transactions would be insecure and secret communications could be read by everyone. Why this would happen is explained in Section 8.12.

Of course, the situation is the same for validity checking, since you can check for validity by checking for satisfiability of a negated formula. This also explains why the simplification of formulas mentioned in Section 3.2 would be hard—validity testing is a special case of determining if a formula simplifies to (\textbf{T}\).

Recently there has been exciting progress on *SAT-solvers* for practical applications like digital circuit verification. These programs find satisfying assignments with amazing efficiency even for formulas with millions of variables. Unfortunately, it’s hard to predict which kind of formulas are amenable to SAT-solver methods, and for formulas that are unsatisfiable, SAT-solvers generally get nowhere.

So no one has a good idea how to solve SAT in polynomial time, or how to prove that it can’t be done—researchers are completely stuck. The problem of determining whether or not SAT has a polynomial time solution is known as the “(\textbf{P}\) vs. (\textbf{NP}\)” problem.[1] It is the outstanding unanswered question in theoretical computer science. It is also one of the seven Millenium Problems: the Clay Institute will award you $1,000,000 if you solve the (\textbf{P}\) vs. (\textbf{NP}\) problem.

[1] **P** stands for problems whose instances can be solved in time that grows polynomially with the size of the instance. **NP** stands for ***nondeterministtic polynomial*** time, but we’ll leave an explanation of what that is to texts on the theory of computational complexity

This page titled 3.5: The SAT Problem is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

3.6: Predicate Formulas

Quantifiers

The "for all" notation, $\forall$, has already made an early appearance in Section 1.1. For example, the predicate

$$\text{“}x^2 \geq 0\text{”}$$

is always true when x is a real number. That is,

$$\forall x \in \mathbb{R}.\, x^2 \geq 0$$

is a true statement. On the other hand, the predicate

$$\text{“}5x^2 - 7 = 0\text{”}$$

is only sometimes true; specifically, when $x = \pm\sqrt{7/5}$. There is a "there exists" notation, $\exists$, to indicate that a predicate is true for at least one, but not necessarily all objects. So

$$\exists x \in \mathbb{R} : 5x^2 - 7 = 0$$

is true, while

$$\forall x \in \mathbb{R} : 5x^2 - 7 = 0$$

is not true.

There are several ways to express the notions of "always true" and "sometimes true" in English. The table below gives some general formats on the left and specific examples using those formats on the right. You can expect to see such phrases hundreds of times in mathematical writing!

Always True

For all $x \in D, P(x)$ is true.	For all $x \in \mathbb{R}, x^2 \geq 0$.
$P(x)$ is true for every x in the set, D.	$x^2 \geq 0$ for every $x \in \mathbb{R}$.

Sometimes True

There is an $x \in D$ such that $P(x)$ is true.	There is an $x \in \mathbb{R}$ such that $5x^2 - 7 = 0$.
$P(x)$ is true for some x in the set, D.	$5x^2 - 7 = 0$ for some $x \in \mathbb{R}$.
$P(x)$ is true for at least one $x \in D$.	$5x^2 - 7 = 0$ for at least one $x \in \mathbb{R}$.

All these sentences "quantify" how often the predicate is true. Specifically, an assertion that a predicate is always true is called a *universal quantification*, and an assertion that a predicate is sometimes true is an *existential quantification*. Sometimes the English sentences are unclear with respect to quantification:

If you can solve any problem we come up with, then you get an A for the course. (3.6.1)

The phrase "you can solve any problem we can come up with" could reasonably be interpreted as either a universal or existential quantification:

you can solve *every* problem we come up with, (3.6.2)

or maybe

you can solve *at least one* problem we come up with. (3.6.3)

To be precise, let Probs be the set of problems we come up with, Solves(x) be the predicate "You can solve problem x," and G be the proposition, "You get an A for the course." Then the two different interpretations of (3.16) can be written as follows:

(**$\forall x \in$ Probs.Solves(x)) IMPLIES G**, for (3.6.1),

(**$\exists x \in$ Probs.Solves(x)) IMPLIES G**, for (3.6.2),

Mixing Quantifiers

Many mathematical statements involve several quantifiers. For example, we already described

Goldbach's Conjecture 1.1.8: Every even integer greater than 2 is the sum of two primes.

Let's write this out in more detail to be precise about the quantification:

For every even integer n greater than 2, there exist primes p and q such that $n = p + q$.

Let Evens be the set of even integers greater than 2, and let Primes be the set of primes. Then we can write Goldbach's Conjecture in logic notation as follows:

$$\underbrace{\forall n \in \text{Evens}}_{\text{for every even integer } n>2} \quad \underbrace{\exists p \in \text{Primes}\ \exists q \in \text{Primes}.}_{\text{there exists primes } p \text{ and } q \text{ such that}} \quad n = p + q.$$

Order of Quantifiers

Swapping the order of different kinds of quantifiers (existential or universal) usually changes the meaning of a proposition. For example, let's return to one of our initial, confusing statements:

"Every American has a dream."

This sentence is ambiguous because the order of quantifiers is unclear. Let A be the set of Americans, let D be the set of dreams, and define the predicate $H(a, d)$ to be "American a has dream d." Now the sentence could mean there is a single dream that every American shares—such as the dream of owning their own home:

$$\exists d \in D\ \forall a \in A.\, H(a, d)$$

Or it could mean that every American has a personal dream:

$$\exists a \in A\ \forall d \in D.\, H(a, d)$$

For example, some Americans may dream of a peaceful retirement, while others dream of continuing practicing their profession as long as they live, and still others may dream of being so rich they needn't think about work at all.

Swapping quantifiers in Goldbach's Conjecture creates a patently false statement that every even number ≥ 2 is the sum of *the same* two primes:

$$\underbrace{\exists p \in \text{Primes}\ \exists q \in \text{Primes}.}_{\text{there exists primes } p \text{ and } q \text{ such that}} \quad \underbrace{\forall n \in \text{Evens}}_{\text{for every even integer } n>2} \quad n = p + q.$$

Variables Over One Domain

When all the variables in a formula are understood to take values from the same nonempty set, D, it's conventional to omit mention of D. For example, instead of $\forall x \in D \exists y \in D.\, Q(x, y)$ we'd write $\forall x \exists y.\, Q(x, y)$. The unnamed nonempty set that x and y range over is called the *domain of discourse*, or just plain *domain*, of the formula.

It's easy to arrange for all the variables to range over one domain. For example, Goldbach's Conjecture could be expressed with all variables ranging over the domain $\mathbb{N}$ as

$$\forall n.\, n \in \text{ Evens IMPLIES } (\exists p\ \exists q.\, p \in \text{Primes AND } q \in \text{Primes AND } n = p + q).$$

Negating Quantifiers

There is a simple relationship between the two kinds of quantifiers. The following two sentences mean the same thing:

Not everyone likes ice cream.

There is someone who does not like ice cream.

The equivalence of these sentences is a instance of a general equivalence that holds between predicate formulas:

$$\text{NOT } (\forall x.\, P(x)) \text{ is equivalent to } \exists x.\ \text{NOT}(P(x)). \tag{3.6.4}$$

Similarly, these sentences mean the same thing:

There is no one who likes being mocked.

Everyone dislikes being mocked.

The corresponding predicate formula equivalence is

$$\text{NOT } (\exists x.\, P(x)) \text{ is equivalent to } \forall x.\ \text{NOT}(P(x)). \tag{3.6.5}$$

The general principle is that *moving a* **NOT** *across a quantifier changes the kind of quantifier*. Note that (3.6.5) follows from negating both sides of (3.6.4).

Validity for Predicate Formulas

The idea of validity extends to predicate formulas, but to be valid, a formula now must evaluate to true no matter what the domain of discourse may be, no matter what values its variables may take over the domain, and no matter what interpretations its predicate variables may be given. For example, the equivalence (3.6.4) that gives the rule for negating a universal quantifier means that the following formula is valid:

$$\text{NOT } (\forall x.\, P(x)) \text{ IFF } \exists x.\ \text{NOT}(P(x)). \tag{3.6.6}$$

Another useful example of a valid assertion is

$$\exists x \forall y.\, P(x, y) \text{ IMPLIES } \forall y\ \exists x.\, P(x, y). \tag{3.6.7}$$

Here's an explanation why this is valid: Let D be the domain for the variables and P_0 be some binary predicate[2] on D. We need to show that if

$$\exists x \in D.\, \forall y \in D.\, P_0(x, y) \tag{3.6.8}$$

holds under this interpretation, then so does

$$\forall y \in D\ \exists x \in D.\, P_0(x, y). \tag{3.6.9}$$

So suppose (3.6.8) is true. Then by definition of ∃, this means that some element $d_0 \in D$ has the property that

$$\forall y \in D.\, P_0(d_0, y).$$

By definition of 8, this means that

$$P_0(d_0, d)$$

is true for all $d \in D$. So given any $d \in D$, there is an element in D, namely, d_0, such that $P_0(d_0, d)$ is true. But that's exactly what (3.6.9) means, so we've proved that (3.6.9) holds under this interpretation, as required.

We hope this is helpful as an explanation, but we don't really want to call it a "proof." The problem is that with something as basic as (3.6.7), it's hard to see what more elementary axioms are ok to use in proving it. What the explanation above did was translate the logical formula (3.6.7) into English and then appeal to the meaning, in English, of "for all" and "there exists" as justification.

In contrast to (3.6.7), the formula

$$\forall y\ \exists x.\, P(x, y) \text{ IMPLIES } \forall x\ \exists y.\, P(x, y) \tag{3.6.10}$$

is *not* valid. We can prove this just by describing an interpretation where the hypothesis, $\forall y \exists x.\, P(x, y)$, is true but the conclusion, $\forall x \exists y.\, P(x, y)$, is not true. For example, let the domain be the integers and $P(x, y)$ mean $x > y$. Then the hypothesis would be true because, given a value, n, for y we could choose the value of x to be $n + 1$, for example. But under this interpretation the conclusion asserts that there is an integer that is bigger than all integers, which is certainly false. An interpretation like this that falsifies an assertion is called a *counter model* to that assertion.

[2]That is, a predicate that depends on two variables.

This page titled 3.6: Predicate Formulas is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

4: Mathematical Data Types

We have assumed that you've already been introduced to the concepts of sets, sequences, and functions, and we've used them informally several times in previous sections. In this chapter, we'll now take a more careful look at these mathematical data types. We'll quickly review the basic definitions, add a few more such as "images" and "inverse images" that may not be familiar, and end the chapter with some methods for comparing the sizes of sets.

4.1: Sets

4.2: Sequences

4.3: Functions

4.4: Binary Relations

4.5: Finite Cardinality

This page titled 4: Mathematical Data Types is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

4.1: Sets

Informally, a *set* is a bunch of objects, which are called the *elements* of the set. The elements of a set can be just about anything: numbers, points in space, or even other sets. The conventional way to write down a set is to list the elements inside curly-braces. For example, here are some sets:

$$\begin{array}{ll} A = \{\text{Alex, Tippy, Shells, Shadow}\} & \text{dead pets} \\ B = \{\text{red, blue, yellow}\} & \text{primary colors} \\ C = \{\{a,b\},\{a,c\},\{b,c\}\} & \text{a set of sets} \end{array} \tag{4.1.1}$$

This works fine for small finite sets. Other sets might be defined by indicating how to generate a list of them:

$$D ::= \{1,2,4,8,16,\ldots\} \qquad \text{the powers of 2} \tag{4.1.2}$$

The order of elements is not significant, so $\{x,y\}$ and $\{y,x\}$ are the same set written two different ways. Also, any object is, or is not, an element of a given set— there is no notion of an element appearing more than once in a set.[1] So, writing $\{x,x\}$ is just indicating the same thing twice: that x is in the set. In particular, $\{x,x\} = \{x\}$.

The expression $e \in S$ asserts that e is an element of set S. For example, $32 \in D$ and blue $\in B$, but Tailspin $\notin A$—yet.

Sets are simple, flexible, and everywhere. You'll find some set mentioned in nearly every section of this text.

Some Popular Sets

Mathematicians have devised special symbols to represent some common sets.

Illegal pream-token (&)

A superscript "+" restricts a set to its positive elements; for example, $\mathbb{R}^+$ denotes the set of positive real numbers. Similarly, $\mathbb{Z}^-$ denotes the set of negative integers.

Comparing and Combining Sets

The expression $S \subseteq T$ indicates that set S is a *subset* of set T, which means that every element of S is also an element of T. For example, $\mathbb{N} \subseteq \mathbb{Z}$ because every nonnegative integer is an integer; $\mathbb{Q} \subseteq \mathbb{R}$ because every rational number is a real number, but $\mathbb{C} \not\subseteq \mathbb{R}$ because not every complex number is a real number.

As a memory trick, think of the "$\subseteq$" symbol as like the "$\leq$" sign with the smaller set or number on the left hand side. Notice that just as $n \leq n$ for any number n, also $S \subseteq S$ for any set S.

There is also a relation, $\subset$, on sets like the "less than" relation $<$ on numbers. $S \subset T$ means that S is a subset of T, but the two are *not* equal. So just as $n \not< n$ for every number n, also $A \not\subset A$, for every set A. "$S \subset T$" is read as "S is a *strict subset* of T."

There are several basic ways to combine sets. For example, suppose

$$\begin{array}{l} X ::= \{1,2,3\}, \\ Y ::= \{2,3,4\}. \end{array} \tag{4.1.3}$$

Definition 4.1.1

- The *union* of sets A and B, denoted $A \cup B$, includes exactly the elements appearing in A or B or both. That is,

$$x \in A \cup B \text{ IFF } x \in A \text{ OR } x \in B.$$

So $X \cup Y = \{1,2,3,4\}$.

- The *intersection* of sets A and B, denoted $A \cap B$, consists of all elements that appear in *both* A and B or both. That is,

$$x \in A \cap B \text{ IFF } x \in A \text{ AND } x \in B.$$

So $X \cap Y = \{2,3\}$.

- The *set difference* of sets A and B, denoted $A - B$, consists of all elements that are in A, but not in B or both. That is,

$$x \in A - B \text{ IFF } x \in A \text{ AND } x \notin B.$$

So $X - Y = \{1\}$ and $Y - X = \{4\}$.

Often all the sets being considered are subsets of a known domain of discourse, D. Then for any subset, A, of D, we define $\overline{A}$ to be the set of all elements of D *not* in A. That is,

$$\overline{A} ::= D - A.$$

The set $\overline{A}$ is called the *complement* of A. So

$$\overline{A} = \emptyset \text{ IFF } A = D.$$

For example, if the domain we're working with is the integers, the complement of the nonnegative integers is the set of negative integers:

$$\overline{\mathbb{N}} = \mathbb{Z}^-.$$

We can use complement to rephrase subset in terms of equality

$$A \subseteq B \text{ is equivalent to } A \cap \overline{B} = \emptyset.$$

Power Set

The set of all the subsets of a set, A, is called the *power set*, pow(A), of A. So

$$B \in \text{pow}(A) \text{ IFF } B \subseteq A.$$

For example, the elements of pow($\{1, 2\}$) are $\emptyset, \{1\}, \{2\}, \text{ and } \{1, 2\}$.

More generally, if A has n elements, then there are 2^n sets in pow(A)—see Theorem 4.5.5. For this reason, some authors use the notation 2^A instead of pow(A).

Set Builder Notation

An important use of predicates is in *set builder notation*. We'll often want to talk about sets that cannot be described very well by listing the elements explicitly or by taking unions, intersections, etc., of easily described sets. Set builder notation often comes to the rescue. The idea is to define a set using a predicate; in particular, the set consists of all values that make the predicate true. Here are some examples of set builder notation:

$$\begin{aligned} A &::= \{n \in \mathbb{N} \mid n \text{ is a prime and } n = 4k + 1 \text{ for some integer } k\} \\ B &::= \{x \in \mathbb{R} \mid x^3 - 3x + 1 > 0\} \\ C &::= \{a + bi \in \mathbb{C} \mid a^2 + 2b^2 \leq 1\} \end{aligned} \tag{4.1.4}$$

The set A consists of all nonnegative integers n for which the predicate

$$\text{"}n \text{ is a prime and } n = 4k + 1 \text{ for some integer } k\text{"}$$

is true. Thus, the smallest elements of A are:

$$5, 13, 17, 29, 37, 41, 53, 61, 73, \ldots$$

Trying to indicate the set A by listing these first few elements wouldn't work very well; even after ten terms, the pattern is not obvious! Similarly, the set B consists of all real numbers x for which the predicate

$$x^3 - 3x + 1 > 0$$

is true. In this case, an explicit description of the set B in terms of intervals would require solving a cubic equation. Finally, set C consists of all complex numbers a C bi such that:

$$a^2 + 2b^2 \leq 1$$

This is an oval-shaped region around the origin in the complex plane.

Providing Set Equalities

Two sets are defined to be equal if they have exactly the same elements. That is, $X = Y$ means that $z \in X$ if and only if $z \in Y$, for all elements, z.[2] So, set equalities can be formulated and proved as "iff" theorems. For example:

Theorem 4.1.2

[Distributive Law for Sets] Let A, B, and C be sets. Then:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \tag{4.1.5}$$

Proof

The equality (4.1.5) is equivalent to the assertion that

$$z \in A \cap (B \cup C) \text{ iff } z \in (A \cap B) \cup (A \cap C) \tag{4.1.6}$$

for all z. Now we'll prove (4.1.6) by a chain of iff's.

Now we have

$$\begin{array}{lll} z \in A \cap (B \cup C) & & \\ \text{iff } (z \in A) \text{ AND } (z \in B \cup C) & \text{(def of } \cap) & \\ \text{iff } (z \in A) \text{ AND } (z \in B \text{ OR } z \in C) & \text{(def of } \cup) & \\ \text{iff } (z \in A \text{ AND } z \in B) \text{ OR } (z \in A \text{ AND } z \in C) & \text{(AND distributivity Thm 3.4.1)} & (4.1.7) \\ \text{iff } (z \in A \cap B) \text{ OR } (z \in A \cap C) & \text{(def of } \cap) & \\ \text{iff } z \in (A \cap B) \cup (A \cap C) & \text{(def of } \cup) & \end{array}$$

■

Although the basic set operations and propositional connectives are similar, it's important not to confuse one with the other. For example, $\cap$ resembles *textOR*, and in fact was defined directly in terms of *textOR*:

$$x \in A \cup B \text{ is equivalent to } (x \in A \text{ OR } x \in B).$$

Similarly, $\cap$ resembles **AND**, and complement resembles **NOT**.

But if A and B are sets, writing A **AND** B is a type-error, since **AND** is an operation on truth-values, not sets. Similarly, if P and Q are propositional variables, writing $P \cup Q$ is another type-error.

The proof of Theorem 4.1.2 illustrates a general method for proving a set equality involving the basic set operations by checking that a corresponding propositional formula is valid. As a further example, from De Morgan's Law (3.4.8) for propositions

$$\text{NOT}(P \text{ AND } Q) \text{ is equivalent to } \overline{P} \text{ OR } \overline{Q}$$

we can derive (Problem 4.5) a corresponding De Morgan's Law for set equality:

$$\overline{A \cap B} = \overline{A} \cup \overline{B}. \tag{4.1.8}$$

Despite this correspondence between two kinds of operations, it's important not to confuse propositional operations with set operations. For example, if X and Y are sets, then it is wrong to write "X **AND** Y" instead of "$X \cap Y$." Applying **AND** to sets will cause your compiler—or your grader—to throw a type error, because an operation that is only supposed to be applied to truth values has been applied to sets. Likewise, if P and Q are propositions, then it is a type error to write "$P \cup Q$" instead of "P **OR** Q."

[1]It's not hard to develop a notion of *multisets* in which elements can occur more than once, but multisets are not ordinary sets and are not covered in this text.

[2]This is actually the first of the ZFC axioms for set theory mentioned at the end of Section 1.3 and discussed further in Section 7.3.2.

This page titled 4.1: Sets is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

4.2: Sequences

Sets provide one way to group a collection of objects. Another way is in a *sequence*, which is a list of objects called *terms* or *components*. Short sequences are commonly described by listing the elements between parentheses; for example, (a, b, c) is a sequence with three terms.

While both sets and sequences perform a gathering role, there are several differences.

- The elements of a set are required to be distinct, but terms in a sequence can be the same. Thus, (a, b, a) is a valid sequence of length three, but $\{a, b, a\}$ is a set with two elements, not three.
- The terms in a sequence have a specified order, but the elements of a set do not. For example, $(a, b, c)$and (a, c, b) are different sequences, but $\{a, b, c\}$ and $\{a, c, b\}$ are the same set.
- Texts differ on notation for the *empty sequence*; we use λ for the empty sequence.

The product operation is one link between sets and sequences. A *Cartesian product* of sets, $S_1 \times S_2 \times \cdots \times S_n$, is a new set consisting of all sequences where the first component is drawn from S_1, the second from S_2, and so forth. Length two sequences are called *pairs*.[3] For example, $\mathbb{N} \times \{a, b\}$ is the set of all pairs whose first element is a nonnegative integer and whose second element is an a or a b:

$$\mathbb{N} \times \{a, b\} = \{(0, a), (0, b), (1, a), (1, b), (2, a), (2, b), ...\}$$

A product of n copies of a set S is denoted S^n. For example, $\{0, 1\}^3$ is the set of all 3-bit sequences:

$$\{0, 1\}^3 = \{(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)\}$$

[3]Some texts call them *ordered* pairs.

This page titled 4.2: Sequences is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

4.3: Functions

Domains and Images

A *function* assigns an element of one set, called the *domain*, to an element of another set, called the *codomain*. The notation

$$f : A \to B$$

indicates that f is a function with domain, A, and codomain, B. The familiar notation "$f(a) = b$" indicates that f assigns the element $b \in B$ to a. Here b would be called the *value* of f at *argument* a.

Functions are often defined by formulas, as in:

$$f_1(x) ::= \frac{1}{x^2}$$

where x is a real-valued variable, or

$$f_2(y, z) ::= y10yz$$

where y and z range over binary strings, or

$$f_3(x, n) ::= \text{the length } n \text{ sequence } \underbrace{(x, \ldots, x)}_{n\ x\text{'s}}$$

where n ranges over the nonnegative integers.

A function with a finite domain could be specified by a table that shows the value of the function at each element of the domain. For example, a function $f_4(P, Q)$ where P and Q are propositional variables is specified by:

P	Q	$f_4(P, Q)$
T	T	T
T	F	F
F	T	T
F	F	T

Notice that f_4 could also have been described by a formula:

$$f_4(P, Q) ::= [P \text{ IMPLIES } Q].$$

A function might also be defined by a procedure for computing its value at any element of its domain, or by some other kind of specification. For example, define $f_5(y)$ to be the length of a left to right search of the bits in the binary string y until a 1 appears, so

$$\begin{aligned} f_5(0010) &= 3, \\ f_5(100) &= 1, \\ f_5(0000) &\text{ is undefined.} \end{aligned} \tag{4.3.1}$$

Notice that f_5 does not assign a value to any string of just 0's. This illustrates an important fact about functions: they need not assign a value to every element in the domain. In fact this came up in our first example $f_1(x) = 1/x^2$, which does not assign a value to 0. So in general, functions may be *partial functions*, meaning that there may be domain elements for which the function is not defined. If a function is defined on every element of its domain, it is called a *total function*.

It's often useful to find the set of values a function takes when applied to the elements in a *set* of arguments. So if $f : A \to B$, and S is a subset of A, we define $f(S)$ to be the set of all the values that f takes when it is applied to elements of S. That is,

$$f(S) ::= \{b \in B \mid f(s) = b \text{ for some } s \in S\}$$

For example, if we let $[r, s]$ denote set of numbers in the interval from r to s on the real line, then $f_1([1, 2]) = [1/4, 1]$.

For another example, let's take the "search for a 1" function, f_5. If we let X be the set of binary words which start with an even number of 0's followed by a 1, then $f_5(X)$ would be the odd nonnegative integers.

Applying f to a set, S, of arguments is referred to as "applying f *pointwise* to S", and the set $f(S)$ is referred to as the *image* of S under f.[4] The set of values that arise from applying f to all possible arguments is called the *range* of f. That is,

$$\mathbf{range}(f) ::= f(\mathbf{domain}(f)).$$

Some authors refer to the codomain as the range of a function, but they shouldn't. The distinction between the range and codomain will be important later in Sections 4.5 when we relate sizes of sets to properties of functions between them.

Function Composition

Doing things step by step is a universal idea. Taking a walk is a literal example, but so is cooking from a recipe, executing a computer program, evaluating a formula, and recovering from substance abuse.

Abstractly, taking a step amounts to applying a function, and going step by step corresponds to applying functions one after the other. This is captured by the operation of *composing* functions. Composing the functions f and g means that first f is applied to some argument, x, to produce $f(x)$, and then g is applied to that result to produce $g(f(x))$.

Definition 4.3.1

For functions $f : A \to B$ and $g : B \to C$, the *composition*, $g \circ f$, of g with f is defined to be the function from A to C defined by the rule:

$$(g \circ f)(x) ::= g(f(x)),$$

for all $x \in A$.

Function composition is familiar as a basic concept from elementary calculus, and it plays an equally basic role in discrete mathematics.

[4]There is a picky distinction between the function f which applies to elements of A and the function which applies f pointwise to subsets of A, because the domain of f is A, while the domain of pointwise-f is $\text{pow}(A)$. It is usually clear from context whether f or pointwise-f is meant, so there is no harm in overloading the symbol f in this way.

This page titled 4.3: Functions is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

4.4: Binary Relations

Binary relations define relations between two objects. For example, "less-than" on the real numbers relates every real number, a, to a real number, b, precisely when $a < b$. Similarly, the subset relation relates a set, A, to another set, B, precisely when $A \subseteq B$. A function $f : A \to B$ is a special case of binary relation in which an element $a \in A$ is related to an element $b \in B$ precisely when $b = f(a)$.

In this section we'll define some basic vocabulary and properties of binary relations.

Definition 4.4.1

A *binary relation*, R, consists of a set, A, called the *domain* of R, a set, B, called the *codomain* of R, and a subset of $A \times B$ called the *graph of* R.

A relation whose domain is A and codomain is B is said to be "between A and B", or "from A to B." As with functions, we write $fR : A \to B$ to indicate that R is a relation from A to B. When the domain and codomain are the same set, A, we simply say the relation is "on A." It's common to use "$a\ R\ b$" to mean that the pair (a, b) is in the graph of R. [5]

Notice that Definition 4.4.1 is exactly the same as the definition in Section 4.3 of a *function*, except that it doesn't require the functional condition that, for each domain element, a, there is *at most* one pair in the graph whose first coordinate is a. As we said, a function is a special case of a binary relation.

The "in-charge of" relation, *Chrg*, for MIT in Spring '10 subjects and instructors is a handy example of a binary relation. Its domain, Fac, is the names of all the MIT faculty and instructional staff, and its codomain is the set, SubNums, of subject numbers in the Fall '09–Spring '10 MIT subject listing. The graph of *Chrg* contains precisely the pairs of the form

$$(\langle \textbf{instructor-name} \rangle, \langle \textbf{subject-num} \rangle)$$

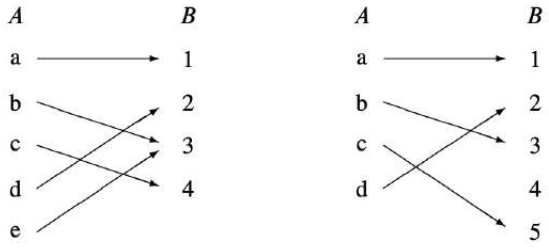
such that the faculty member named $\langle \textbf{instructor-name} \rangle$ is in charge of the subject with number $\langle \textbf{subject-num} \rangle$) that was offered in Spring '10. So graph(*Chrg*) contains pairs like

Illegal pream-token (&)

Some subjects in the codomain, SubNums, do not appear among this list of pairs—that is, they are not in range(*Chrg*). These are the Fall term-only subjects. Similarly, there are instructors in the domain, Fac, who do not appear in the list because they are not in charge of any Spring term subjects.

Relation Diagrams

Some standard properties of a relation can be visualized in terms of a diagram. The diagram for a binary relation, R, has points corresponding to the elements of the domain appearing in one column (a very long column if domain(R) is infinite). All the elements of the codomain appear in another column which we'll usually picture as being to the right of the domain column. There is an arrow going from a point, a, in the lefthand, domain column to a point, b, in the righthand, codomain column, precisely when the corresponding elements are related by R. For example, here are diagrams for two functions:



Being a function is certainly an important property of a binary relation. What it means is that every point in the domain column has *at most one arrow coming out of it.* So we can describe being a function as the "≤ 1 arrow out" property. There are four more standard properties of relations that come up all the time. Here are all five properties defined in terms of arrows:

Definition 4.4.2

A binary relation, R, is:

- a *function* when it has the [≤ 1 arrow **out**] property.
- *surjective* when it has the [≥ 1 arrows **in**] property. That is, every point in the righthand, codomain column has at least one arrow pointing to it.
- *total* when it has the [≥ 1 arrows **out**] property.
- *injective* when it has the [≤ 1 arrow **in**] property.
- *bijective* when it has the [$= 1$ arrow **out**] and the [$= 1$ arrow **in**] property.

From here on, we'll stop mentioning the arrows in these properties and for example, just write [≤ 1 in] instead of [≤ 1 arrows in].

So in the diagrams above, the relation on the left has the [= 1 out] and [≥ 1 in] properties, which means it is a total, surjective function. But it does not have the [≤ 1 in] property because element 3 has two arrows going into it; it is not injective.

The relation on the right has the [= 1 out] and [≤ 1 in] properties, which means it is a total, injective function. But it does not have the [≥ 1 in] property because element 4 has no arrow going into it; it is not surjective.

The arrows in a diagram for R correspond, of course, exactly to the pairs in the graph of R. Notice that the arrows alone are not enough to determine, for example, if R has the [≥ 1 out], total, property. If all we knew were the arrows, we wouldn't know about any points in the domain column that had no arrows out. In other words, graph(R) alone does not determine whether R is total: we also need to know what domain(R) is.

Example 4.4.3

The function defined by the formula $1/x^2$ has the [≥ 1 out] property if its domain is $\mathbb{R}^+$, but not if its domain is some set of real numbers including 0. It has the [= 1 in] and [= 1 out] property if its domain and codomain are both $\mathbb{R}^+$, but it has neither the [≤ 1 in] nor the [≥ 1 out] property if its domain and codomain are both $\mathbb{R}$.

Relational Images

The idea of the image of a set under a function extends directly to relations.

Definition 4.4.4

The *image* of a set, Y, under a relation, R, written $R(Y)$, is the set of elements of the codomain, B, of R that are related to some element in Y. In terms of the relation diagram, $R(Y)$ is the set of points with an arrow coming in that starts from some point in Y.

For example, the set of subject numbers that Meyer is in charge of in Spring '10 is exactly *Chrg*(A. Meyer). To figure out what this is, we look for all the arrows in the *Chrg* diagram that start at "A. Meyer," and see which subject-numbers are at the other end of these arrows. Looking at the list (4.4) of pairs in graph(*Chrg*), we see that these subject-numbers are {6.042, 18.062, 6.844}. Similarly, to find the subject numbers that either Freeman or Eng are in charge of, we can collect all the arrows that start at either "G. Freeman," or "T. Eng" and, again, see which subjectnumbers are at the other end of these arrows. This is *Chrg* ({G. Freeman, T. Eng}). Looking again at the list (4.4), we see that

$$Chrg\ (\{\text{G. Freeman, T. Engg}\}) = \{6.011, 6.881, 6.882, 6.UAT\}$$

Finally, Fac is the set of all in-charge instructors, so *Chrg*(Fac) is the set of all the subjects listed for Spring '10.

Inverse Relations and Images

Definition 4.4.5

The inverse, R^{-1} of a relation $R : A \to B$ is the relation from B to A defined by the rule

$$bR^{-1}a \text{ IFF } aRb$$

In other words, R^{-1} is the relation you get by reversing the direction of the arrows in the diagram of R.

Definition 4.4.6

The image of a set under the relation, R^{-1}, is called the *inverse image* of the set. That is, the inverse image of a set, X, under the relation, R, is defined to be $R^{-1}(X)$.

Continuing with the in-charge example above, the set of instructors in charge of 6.UAT in Spring '10 is exactly the inverse image of {6.UAT} under the *Chrg* relation. From the list (???), we see that Eng and Freeman are both in charge of 6.UAT, that is,

$$\{\text{T. Eng, D. Freeman}\} \subseteq Chrg^{-1}(\{\text{6.UAT}\}).$$

We can't assert equality here because there may be additional pairs further down the list showing that additional instructors are co-incharge of 6.UAT.

Now let Intro be the set of introductory course 6 subject numbers. These are the subject numbers that start with "6.0." So the set of names of the instructors who were in-charge of introductory course 6 subjects in Spring '10, is $Chrg^{-1}$(Intro). From the part of the *Chrg* list shown in (???), we see that Meyer, Leighton, Freeman, and Guttag were among the instructors in charge of introductory subjects in Spring '10. That is,

$$\{\text{Meyer, Leighton, Freeman, Guttag}\} \subseteq Chrg^{-1}(\text{Intro}).$$

Finally, $Chrg^{-1}$(SubNums), is the set of all instructors who were in charge of a subject listed for Spring '10.

[5]Writing the relation or operator symbol between its arguments is called *infix* notation. Infix expressions like "$m < n$" or "$m + n$" are the usual notation used for things like the less-then relation or the addition operation rather than prefix notation like "$<(m, n)$" or "$+(m, n)$."

This page titled 4.4: Binary Relations is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

4.5: Finite Cardinality

A finite set is one that has only a finite number of elements. This number of elements is the "size" or *cardinality* of the set:

Definition 4.5.1

If A is a finite set, the *cardinality* of A, written $|A|$, is the number of elements in A.

A finite set may have no elements (the empty set), or one element, or two elements, ... , so the cardinality of finite sets is always a nonnegative integer.

Now suppose $R : A \to B$ is a function. This means that every element of A contributes at most one arrow to the diagram for R, so the number of arrows is at most the number of elements in A. That is, if R is a function, then

Misplaced '#'

If R is also surjective, then every element of B has an arrow into it, so there must be at least as many arrows in the diagram as the size of B. That is,

Misplaced '#'

Combining these inequalities implies that if R is a surjective function, then $|A| \geq |B|$.

In short, if we write A surj B to mean that there is a surjective function from A to B, then we've just proved a lemma: if A surj B for finite sets A, B, then $|A| \geq |B|$. The following definition and lemma lists this statement and three similar rules relating domain and codomain size to relational properties.

Definition 4.5.2

Let A, B be (not necessarily finite) sets. Then

1. A surj B iff there is a surjective *function* from A to B.
2. A inj B iff there is a injective *total* from A to B.
3. A bij B iff there is a bijective from A to B.

Lemma 4.5.3. *For finite sets A, B:*

1. *If A surj B, then $|A| \geq |B|$.*
2. *If A inj B, then $|A| \leq |B|$.*
3. *If A bij B, then $|A| = |B|$.*

Proof. We've already given an "arrow" proof of implication 1. Implication 2. follows immediately from the fact that if R has the [≤ 1 out], function property, and the [≥ 1 in], surjective property, then R^{-1} is total and injective, so A surj B iff B inj A. Finally, since a bijection is both a surjective function and a total injective relation, implication 3. is an immediate consequence of the first two. ■

Lemma 4.5.3.1. has a converse: if the size of a finite set, A, is greater than or equal to the size of another finite set, B, then it's always possible to define a surjective function from A to B. In fact, the surjection can be a total function. To see how this works, suppose for example that

$$\begin{aligned} A &= \{a_0, a_1, a_2, a_3, a_4, a_5\} \\ B &= \{b_0, b_1, b_2, b_3.\} \end{aligned} \tag{4.5.1}$$

Then define a total function f W A ! B by the rules $f : A \to B$ by the rules

$$f(a_0) ::= b_0, f(a_1) ::= b_1, f(a_2) ::= b_2, f(a_3) = f(a_4) = f(a_5) ::= b_3$$

More concisely,

$$f(a_i) ::= b_{min(i,3)},$$

for $0 \le i \le 5$. Since $5 \ge 3$, this f is a surjection.

So we have figured out that if A and B are finite sets, then $|A| \ge |B|$ *if and only if* A surj B. All told, this argument wraps up the proof of a theorem that summarizes the whole finite cardinality story:

Theorem 4.5.4

[Mapping Rules] For finite *sets*, A, B,

$$|A| \ge |B| \text{ iff } A \text{ surj } B, \tag{4.5.2}$$

$$|A| \le |B| \text{ iff } A \text{ inj } B, \tag{4.5.3}$$

$$|A| = |B| \text{ iff } A \text{ bij } B. \tag{4.5.4}$$

How Many Subsets of a Finite Set?

As an application of the bijection mapping rule (4.5.4), we can give an easy proof of:

Theorem 4.5.5. There are 2n subsets of an n-element set. That is,

Theorem 4.5.5

There are 2^n *subsets of an n-element set. That is,*

$$|A| = n \text{ implies } |\text{pow}(A)| = 2^n.$$

For example, the three-element set $\{a_1, a_2, a_3\}$ has eight different subsets:

$$\begin{array}{cccc} \emptyset & \{a_1\} & \{a_2\} & \{a_1, a_2\} \\ \{a_3\} & \{a_1, a_3\} & \{a_2, a_3\} & \{a_1, a_2, a_3\} \end{array}$$

Theorem 4.5.5 follows from the fact that there is a simple bijection from subsets of A to $\{01\}^n$, the n-bit sequences. Namely, let $a_1, a_2, \ldots, a_n$ be the elements of A. The bijection maps each subset of $S \subseteq A$ to the bit sequence $(b_1, b_2, \ldots, b_n)$ defined by the rule that

$$b_i = 1 \text{ iff } a_i \in S.$$

For example, if $n = 10$, then the subset $\{a_2, a_3, a_5, a_7, a_{10}\}$ maps to a 10-bit sequence as follows:

subset: { a_2, a_3, a_5, a_7, a_{10}}

sequence: (0, 1, 1, 0, 1, 0, 1, 0, 0, 1)

Now by bijection case of the Mapping Rules 4.5.4.(4.5.4),

$$|\text{pow}(A)| = |\{0,1\}^n|.$$

But every computer scientist knows6 that there are 2^n n-bit sequences! So we've proved Theorem 4.5.5!

[6]In case you're someone who doesn't know how many n-bit sequences there are, you'll find the 2^n explained in Section 14.2.2.

This page titled 4.5: Finite Cardinality is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

5: Induction

Induction is a powerful method for showing a property is true for all nonnegative integers. Induction plays a central role in discrete mathematics and computer science. In fact, its use is a defining characteristic of *discrete*—as opposed to *continuous*—mathematics. This chapter introduces two versions of induction, Ordinary and Strong, and explains why they work and how to use them in proofs. It also introduces the Invariant Principle, which is a version of induction specially adapted for reasoning about step-by-step processes.

5.1: Ordinary Induction

5.2: Strong Induction

5.3: Strong Induction vs. Induction vs. Well Ordering

5.4: State Machines

This page titled 5: Induction is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

5.1: Ordinary Induction

To understand how induction works, suppose there is a professor who brings a bottomless bag of assorted miniature candy bars to her large class. She offers to share the candy in the following way. First, she lines the students up in order. Next she states two rules:

1. The student at the beginning of the line gets a candy bar.
2. If a student gets a candy bar, then the following student in line also gets a candy bar.

Let's number the students by their order in line, starting the count with 0, as usual in computer science. Now we can understand the second rule as a short description of a whole sequence of statements:

- If student 0 gets a candy bar, then student 1 also gets one.
- If student 1 gets a candy bar, then student 2 also gets one.
- If student 2 gets a candy bar, then student 3 also gets one.

:

Of course, this sequence has a more concise mathematical description:

> If student n gets a candy bar, then student $n+1$ gets a candy bar, for all nonnegative integers n.

So suppose you are student 17. By these rules, are you entitled to a miniature candy bar? Well, student 0 gets a candy bar by the first rule. Therefore, by the second rule, student 1 also gets one, which means student 2 gets one, which means student 3 gets one as well, and so on. By 17 applications of the professor's second rule, you get your candy bar! Of course the rules really guarantee a candy bar to every student, no matter how far back in line they may be.

A Rule for Ordinary Induction

The reasoning that led us to conclude that every student gets a candy bar is essentially all there is to induction.

The Induction Principle.

Let P be a predicate on nonnegative integers. If

- $P(0)$ is true, and
- $P(n)$ IMPLIES $P(n+1)$ for all nonnegative integers, n,

then

- $P(m)$ is true for all nonnegative integers, m.

Since we're going to consider several useful variants of induction in later sections, we'll refer to the induction method described above as *ordinary induction* when we need to distinguish it. Formulated as a proof rule as in Section 1.4.1, this would be

Rule. Induction Rule

$$\frac{P(0), \quad \forall n \in \mathbb{N}.\, P(n) \text{ IMPLIES } P(n+1)}{\forall m \in \mathbb{N}.\, P(m)}$$

This Induction Rule works for the same intuitive reason that all the students get candy bars, and we hope the explanation using candy bars makes it clear why the soundness of ordinary induction can be taken for granted. In fact, the rule is so obvious that it's hard to see what more basic principle could be used to justify it.[1] What's not so obvious is how much mileage we get by using it.

A Familiar Example

Below is the formula (5.1.1) for the sum of the nonnegative integers up to n. The formula holds for all nonnegative integers, so it is the kind of statement to which induction applies directly. We've already proved this formula using the Well Ordering Principle (Theorem 2.2.1), but now we'll prove it *by induction*, that is, using the Induction Principle.

Theorem 5.1.1

For all $n \in \mathbb{N}$,

$$1+2+3+\cdots+n=\frac{n(n+1)}{2} \tag{5.1.1}$$

To prove the theorem by induction, define predicate $P(n)$ to be the equation (5.1.1). Now the theorem can be restated as the claim that $P(n)$ is true for all $n \in \mathbb{N}$. This is great, because the Induction Principle lets us reach precisely that conclusion, provided we establish two simpler facts:

- $P(0)$ is true.
- For all $n \in \mathbb{N}$, $P(n)$ IMPLIES $P(n+1)$.

So now our job is reduced to proving these two statements.

The first statement follows because of the convention that a sum of zero terms is equal to 0. So $P(0)$ is the true assertion that a sum of zero terms is equal to 0(0+1) / 2=0.

The second statement is more complicated. But remember the basic plan from Section 1.5 for proving the validity of any implication: *assume* the statement on the left and then *prove* the statement on the right. In this case, we assume $P(n)$— namely, equation (5.1.1)—in order to prove $P(n+1)$, which is the equation

$$1+2+3+\cdots+n+(n+1)=\frac{(n+1)(n+2)}{2} \tag{5.1.2}$$

These two equations are quite similar; in fact, adding $(n+1)$ to both sides of equation (5.1.1) and simplifying the right side gives the equation (5.1.2):

$$\begin{aligned} 1+2+3+\cdots+n+(n+1) &= \frac{n(n+1)}{2}+(n+1) \\ &= \frac{(n+2)(n+1)}{2} \end{aligned} \tag{5.1.3}$$

Thus, if $P(n)$ is true, then so is $P(n+1)$. This argument is valid for every nonnegative integer n, so this establishes the second fact required by the induction proof. Therefore, the Induction Principle says that the predicate $P(m)$ is true for all nonnegative integers, m. The theorem is proved.

A Template for Induction Proofs

The proof of equation (5.1.1) was relatively simple, but even the most complicated induction proof follows exactly the same template. There are five components:

1. **State that the proof uses induction**. This immediately conveys the overall structure of the proof, which helps your reader follow your argument.
2. **Define an appropriate predicate** The predicate $P(n)$ is called the *induction hypothesis*. The eventual conclusion of the induction argument will be that $P(n)$ is true for all nonnegative n. A clearly stated induction hypothesis is often the most important part of an induction proof, and its omission is the largest source of confused proofs by students. In the simplest cases, the induction hypothesis can be lifted straight from the proposition you are trying to prove, as we did with equation (5.1.1). Sometimes the induction hypothesis will involve several variables, in which case you should indicate which variable serves as n.
3. **Prove that $P(0)$ is true**. This is usually easy, as in the example above. This part of the proof is called the *base case or basis step*.
4. **Prove that $P(n)$ implies $P(n+1)$ for every nonnegative integer n.** This is called the *inductive step*. The basic plan is always the same: assume that $P(n)$ is true and then use this assumption to prove that $P(n+1)$ is true. These two statements should be fairly similar, but bridging the gap may require some ingenuity. Whatever argument you give must be valid for every nonnegative integer n, since the goal is to prove that *all* the following implications are true:

$$P(0) \to P(1), P(1) \to P(2), P(2) \to P(3), \ldots$$

5. **Invoke induction**. Given these facts, the induction principle allows you to conclude that $P(n)$ is true for all nonnegative n. This is the logical capstone to the whole argument, but it is so standard that it's usual not to mention it explicitly

Always be sure to explicitly label the *base case* and the *inductive step*. Doing so will make your proofs clearer and will decrease the chance that you forget a key step—like checking the base case.

A Clean Writeup

The proof of Theorem 5.1.1 given above is perfectly valid; however, it contains a lot of extraneous explanation that you won't usually see in induction proofs. The writeup below is closer to what you might see in print and should be prepared to produce yourself.

Revised proof of Theorem 5.1.1. We use induction. The induction hypothesis, $P(n)$, will be equation (5.1.1).

Base case: $P(0)$ is true, because both sides of equation (5.1.1) equal zero when $n = 0$.

Inductive step: Assume that $P(n)$ is true, that is equation (5.1.1) holds for some nonnegative integer n. Then adding $n+1$ to both sides of the equation implies that

$$\begin{aligned} 1+2+3+\cdots+n+(n+1) &= \frac{n(n+1)}{2}+(n+1) \\ &= \frac{(n+1)(n+2)}{2} \quad \text{(by simple algebra)} \end{aligned} \tag{5.1.4}$$

which proves $P(n+1)$.

So it follows by induction that $P(n)$ is true for all nonnegative n. ■

It probably bothers you that induction led to a proof of this summation formula but did not provide an intuitive way to understand it nor did it explain where the formula came from in the first place.[2] This is both a weakness and a strength. It is a weakness when a proof does not provide insight. But it is a strength that a proof can provide a reader with a reliable guarantee of correctness without *requiring* insight.

A More Challenging Example

During the development of MIT's famous Stata Center, as costs rose further and further beyond budget, some radical fundraising ideas were proposed. One rumored plan was to install a big square courtyard divided into unit squares. The big square would be 2^n units on a side for some undetermined nonnegative integer n, and one of the unit squares in the center[3] occupied by a statue of a wealthy potential donor—whom the fund raisers privately referred to as "Bill." The $n = 3$ case is shown in Figure 5.1.

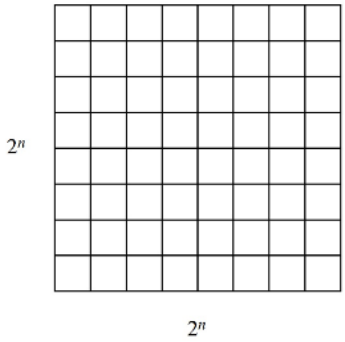


Figure 5.1 A $2^n \times 2^n$ courtyard for $n = 3$.

A complication was that the building's unconventional architect, Frank Gehry, was alleged to require that only special L-shaped tiles (shown in Figure 5.2) be used for the courtyard. For $n = 2$, a courtyard meeting these constraints is shown in Figure 5.3. But what about for larger values of n? Is there a way to tile a $2^n \times 2^n$ courtyard with L-shaped tiles around a statue in the center? Let's try to prove that this is so.

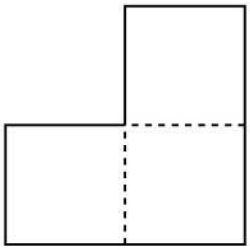


Figure 5.2 The special L-shaped tile.

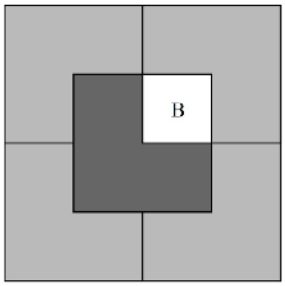


Figure 5.3 A tiling using L-shaped tiles for $n = 2$ with Bill in a center square.

Theorem 5.1.2

For all $n \geq 0$ there exists a tiling of a $2^n \times 2^n$ courtyard with Bill in a central square.

Proof

(doomed attempt) The proof is by induction. Let $P(n)$ be the proposition that there exists a tiling of a $2^n \times 2^n$ courtyard with Bill in the center.

Base case: $P(0)$ is true because Bill fills the whole courtyard.

Inductive step: Assume that there is a tiling of a $2^n \times 2^n$ courtyard with Bill in the center for some $n \geq 0$. We must prove that there is a way to tile a $2^{n+1} \times 2^{n+1}$ courtyard with Bill in the center ... ■

Now we're in trouble! The ability to tile a smaller courtyard with Bill in the center isn't much help in tiling a larger courtyard with Bill in the center. We haven't figured out how to bridge the gap between $P(n)$ and $P(n+1)$.

So if we're going to prove Theorem 5.1.2 by induction, we're going to need some other induction hypothesis than simply the statement about n that we're trying to prove.

When this happens, your first fallback should be to look for a *stronger* induction hypothesis; that is, one which implies your previous hypothesis. For example, we could make $P(n)$ the proposition that for *every* location of Bill in a $2^n \times 2^n$ courtyard, there exists a tiling of the remainder.

This advice may sound bizarre: "If you can't prove something, try to prove something grander!" But for induction arguments, this makes sense. In the inductive step, where you have to prove $P(n)$ IMPLIES $P(n+1)$, you're in better shape because you can *assume* $P(n)$, which is now a more powerful statement. Let's see how this plays out in the case of courtyard tiling.

Proof (successful attempt). The proof is by induction. Let $P(n)$ be the proposition that for every location of Bill in a $2^n \times 2^n$ courtyard, there exists a tiling of the remainder.

Base case: $P(0)$ is true because Bill fills the whole courtyard.

Inductive step: Assume that $P(n)$ is true for some $n \geq 0$; that is, for every location of Bill in a $2^n \times 2^n$ courtyard, there exists a tiling of the remainder. Divide the (2^{n+1} \times 2^{n+1}\) courtyard into four quadrants, each $2^n \times 2^n$. One quadrant contains Bill (**B** in the diagram below). Place a temporary Bill (**X** in the diagram) in each of the three central squares lying outside this quadrant as shown in Figure 5.4.

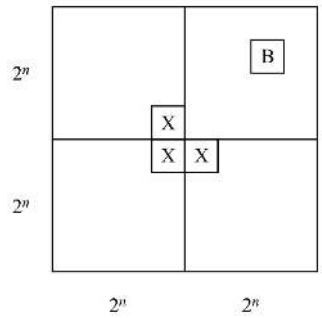


Figure 5.4 Using a stronger inductive hypothesis to prove Theorem 5.1.2.

Now we can tile each of the four quadrants by the induction assumption. Replacing the three temporary Bills with a single L-shaped tile completes the job. This proves that $P(n)$ implies $P(n+1)$ for all $n \geq 0$. Thus $P(n)$ is true for all $m \in \mathbb{N}$, and the theorem follows as a special case where we put Bill in a central square. ■

This proof has two nice properties. First, not only does the argument guarantee that a tiling exists, but also it gives an algorithm for finding such a tiling. Second, we have a stronger result: if Bill wanted a statue on the edge of the courtyard, away from the pigeons, we could accommodate him!

Strengthening the induction hypothesis is often a good move when an induction proof won't go through. But keep in mind that the stronger assertion must actually be *true*; otherwise, there isn't much hope of constructing a valid proof. Sometimes finding just the right induction hypothesis requires trial, error, and insight. For example, mathematicians spent almost twenty years trying to prove or disprove the conjecture that every planar graph is 5-choosable.[4] Then, in 1994, Carsten Thomassen gave an induction proof simple enough to explain on a napkin. The key turned out to be finding an extremely clever induction hypothesis; with that in hand, completing the argument was easy!

A Faulty Induction Proof

If we have done a good job in writing this text, right about now you should be thinking, "Hey, this induction stuff isn't so hard after all—just show $P(0)$ is true and that $P(n)$ implies $P(n+1)$ for any number n." And, you would be right, although sometimes when you start doing induction proofs on your own, you can run into trouble. For example, we will now use induction to "prove" that all horses are the same color—just when you thought it was safe to skip class and work on your robot program instead. Sorry!

False Theorem. *All horses are the same color.*

Notice that no n is mentioned in this assertion, so we're going to have to reformulate it in a way that makes an n explicit. In particular, we'll (falsely) prove that

False Theorem 5.1.3. *In every set of $n \geq 1$ horses, all the horses are the same color.*

This is a statement about all integers $n \geq 1$ rather ≥ 0, so it's natural to use a slight variation on induction: prove $P(1)$ in the base case and then prove that $P(n)$ implies $P(n+1)$ for all $n \geq 1$ in the inductive step. This is a perfectly valid variant of induction and is *not* the problem with the proof below.

Bogus proof. The proof is by induction on n. The induction hypothesis, $P(n)$, will be

$$\text{In every set of } n \text{ horses, all are the same color.} \tag{5.1.5}$$

Base case: $n = 1$. $P(1)$ is true, because in a size-1 set of horses, there's only one horse, and this horse is definitely the same color as itself.

Inductive step: Assume that $P(n)$ is true for some $n \geq 1$. That is, assume that in every set of n horses, all are the same color. Now suppose we have a set of $n+1$ horses:

$$h_1, h_2, \ldots, h_n, h_{n+1}$$

We need to prove these $n+1$ horses are all the same color.

By our assumption, the first n horses are the same color:

$$\underbrace{h_1, h_2, \ldots, h_n}_{\text{same color}}, h_{n+1}$$

Also by our assumption, the last n horses are the same color:

$$h_1, \underbrace{h_2, \ldots, h_n, h_{n+1}}_{\text{same color}}$$

So h_1 is the same color as the remaining horses besides h_{n+1} —that is,$h_2, \ldots, h_n$. Likewise, h_{n+1} is the same color as the remaining horses besides h_1—that is, $h_2, \ldots, h_n$, again. Since h_1 and h_{n+1} are the same color as $h_2, \ldots, h_n$, all $n+1$ horses must be the same color, and so $P(n+1)$ is true. Thus, $P(n)$ implies $P(n+1)$.

By the principle of induction, $P(n)$ is true for all $n \geq 1$. ■

We've proved something false! Does this mean that math broken and we should all take up poetry instead? Of course not! It just means that this proof has a mistake.

The mistake in this argument is in the sentence that begins "So h_1 is the same color as the remaining horses besides h_{n+1}—that is $h_2, \ldots, h_n$ ...." The ellipis notation ("...") in the expression "$h_1, h_2, \ldots, h_n, h_{n+1}$" creates the impression that there are some remaining horses—namely $h_2, \ldots, h_n$ —besides h_1 and h_{n+1} However, this is not true when $n = 1$. In that case, $h_1, h_2, \ldots, h_n, h_{n+1}$ is just h_1, h_2 and *there are no "remaining" horses* for h_1 to share a color with. And of course, in this case h_1 and h_2 really don't need to be the same color.

This mistake knocks a critical link out of our induction argument. We proved $P(1)$ and we *correctly* proved $P(2) \to P(3), P(3) \to P(4)$, etc. But we failed to prove $P(1) \to P(2)$, and so everything falls apart: we cannot conclude that $P(2)$, $P(3)$, etc., are true. And naturally, these propositions are all false; there are sets of n horses of different colors for all $n \geq 2$.

Students sometimes explain that the mistake in the proof is because $P(n)$ is false for $n \geq 2$, and the proof assumes something false, $P(n)$, in order to prove $P(n+1)$. You should think about how to help such a student understand why this explanation would get no credit on a Math for Computer Science exam.

[1]But see Section 5.3.

[2]Methods for finding such formulas are covered in Part III of the text.

[3]In the special case $n = 0$, the whole courtyard consists of a single central square; otherwise, there are four central squares.

[4]5-choosability is a slight generalization of 5-colorability. Although every planar graph is 4- colorable and therefore 5-colorable, not every planar graph is 4-choosable. If this all sounds like nonsense, don't panic. We'll discuss graphs, planarity, and coloring in Part II of the text.

This page titled 5.1: Ordinary Induction is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

5.2: Strong Induction

A useful variant of induction is called *strong induction*. Strong induction and ordinary induction are used for exactly the same thing: proving that a predicate is true for all nonnegative integers. Strong induction is useful when a simple proof that the predicate holds for $n+1$ does not follow just from the fact that it holds at n, but from the fact that it holds for other values $\leq n$.

A Rule for Strong Induction

Principle of Strong Induction.

Let P be a predicate on nonnegative integers. If

- $P(0)$ is true, and
- for all $n \in \mathbb{N}$, $P(0)$, $P(1)$, ..., $P(n)$ *together* imply $P(n+1)$,

then $P(m)$ is true for all $m \in \mathbb{N}$.

The only change from the ordinary induction principle is that strong induction allows you make more assumptions in the inductive step of your proof! In an ordinary induction argument, you assume that $P(n)$ is true and try to prove that $P(n+1)$ is also true. In a strong induction argument, you may assume that $P(0)$, $P(1)$, ..., and $P(n)$ are *all* true when you go to prove $P(n+1)$. So you can assume a *stronger* set of hypotheses which can make your job easier.

Formulated as a proof rule, strong induction is

Rule. ***Strong Induction Rule***

$$\frac{P(0), \quad \forall n \in \mathbb{N}.\, (P(0) \text{ AND } P(1) \text{ AND } \ldots \text{ AND } P(n)) \text{ IMPLIES } P(n+1)}{\forall m \in \mathbb{N}.\, P(m)}$$

Stated more succintly, the rule is

Rule.

$$\frac{P(0), \quad [\forall k \leq n \in \mathbb{N}.\, P(k)] \text{ IMPLIES } P(n+1)}{\forall m \in \mathbb{N}.\, P(m)}$$

The template for strong induction proofs is identical to the template given in Section 5.1.3 for ordinary induction except for two things:

- you should state that your proof is by strong induction, and
- you can assume that $P(0), P(1), \ldots,$ and $P(n)$ are all true instead of only $P(n)$ during the inductive step.

Products of Primes

As a first example, we'll use strong induction to re-prove Theorem 2.3.1 which we previously proved using Well Ordering.

Theorem

Every integer greater than 1 is a product of primes.

Proof

We will prove the Theorem by strong induction, letting the induction hypothesis, $P(n)$, be

$$n \text{ is a product of primes.}$$

So the Theorem will follow if we prove that $P(n)$ holds for all $n \geq 2$.

Base Case: (n = 2): $P(2)$ is true because 2 is prime, so it is a length one product of primes by convention.

Inductive step: Suppose that $n \geq 2$ and that every number from 2 to n is a product of primes. We must show that $P(n+1)$ holds, namely, that $n+1$ is also a product of primes. We argue by cases:

If $n+1$ is itself prime, then it is a length one product of primes by convention, and so $P(n+1)$ holds in this case.

Otherwise, $n+1$ is not prime, which by definition means $n+1 = k \cdot m$ for some integers k, m between 2 and n. Now by the strong induction hypothesis, we know that both k and m are products of primes. By multiplying these products, it follows immediately that $k \cdot m = n+1$ is also a product of primes. Therefore, $P(n+1)$ holds in this case as well.

So $P(n+1)$ holds in any case, which completes the proof by strong induction that $P(n)$ holds for all $n \geq 2$. ■

Making Change

The country Inductia, whose unit of currency is the Strong, has coins worth 3Sg (3 Strongs) and 5Sg. Although the Inductians have some trouble making small change like 4Sg or 7Sg, it turns out that they can collect coins to make change for any number that is at least 8 Strongs.

Strong induction makes this easy to prove for $n+1 \geq 11$, because then $(n+1)-3 \geq 8$, so by strong induction the Inductians can make change for exactly $(n+1)-3$ Strongs, and then they can add a 3Sg coin to get $(n+1)$ Sg. So the only thing to do is check that they can make change for all the amounts from 8 to 10Sg, which is not too hard to do.

Here's a detailed writeup using the official format:

Proof. We prove by strong induction that the Inductians can make change for any amount of at least 8Sg. The induction hypothesis, $P(n)$ will be:

There is a collection of coins whose value is $n+8$ Strongs.

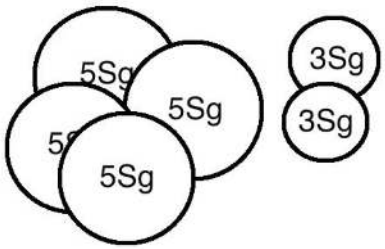


Figure 5.5 One way to make 26 Sg using Strongian currency

We now proceed with the induction proof:

Base case: $P(0)$ is true because a 3Sg coin together with a 5Sg coin makes 8Sg.

Inductive step: We assume $P(k)$ holds for all $k \leq n$, and prove that $P(n+1)$ holds. We argue by cases:

Case $(n+1=1)$: We have to make $n+1)+8=$ 9Sg. We can do this using three 3Sg coins.

Case $(n+1=2)$: We have to make $n+1)+8=$ 10Sg. Use two 5Sg coins.

Case $(n+1 \geq 3)$: Then $0 \leq n-2 \leq n$, so by the strong induction hypothesis, the Inductians can make change for $(n-2)+8$Sg. Now by adding a 3Sg coin, they can make change for $(n+1)+8$Sg, so $P(n+1)$ holds in this case.

Since $n \geq 0$, we know that $n+1 \geq 1$ and thus that the three cases cover every possibility. Since $P(n+1)$ is true in every case, we can conclude by strong induction that for all $n \geq 0$, the Inductians can make change for $n+8$ Strong. That is, they can make change for any number of eight or more Strong. ■

The Stacking Game

Here is another exciting game that's surely about to sweep the nation! You begin with a stack of n boxes. Then you make a sequence of moves. In each move, you divide one stack of boxes into two nonempty stacks. The game ends when you have n stacks, each containing a single box. You earn points for each move; in particular, if you divide one stack of height $a+b$ into two stacks with heights a and b, then you score ab points for that move. Your overall score is the sum of the points that you earn for each move. What strategy should you use to maximize your total score?

Stack Heights										Score
$\underline{10}$										
5	$\underline{5}$									25 points
$\underline{5}$	3	2								6
$\underline{4}$	3	2	1							4
2	$\underline{3}$	2	1	2						4
$\underline{2}$	2	2	1	2	1					2
1	$\underline{2}$	2	1	2	1	1				1
1	1	$\underline{2}$	1	2	1	1	1			1
1	1	1	1	$\underline{2}$	1	1	1	1		1
1	1	1	1	1	1	1	1	1	1	1
									Total Score	= 45 points

Figure 5.6 An example of the stacking game with $n = 10$ boxes. On each line, the underlined stack is divided in the next step.

As an example, suppose that we begin with a stack of $n = 10$ boxes. Then the game might proceed as shown in Figure 5.6. Can you find a better strategy?

Analyzing the Game

Let's use strong induction to analyze the unstacking game. We'll prove that your score is determined entirely by the number of boxes—your strategy is irrelevant!

Theorem 5.2.1

Every way of unstacking n blocks gives a score of $n(n-1)/2$ points.

There are a couple technical points to notice in the proof:

- The template for a strong induction proof mirrors the one for ordinary induction.
- As with ordinary induction, we have some freedom to adjust indices. In this case, we prove $P(1)$ in the base case and prove that $P(1), \ldots, P(n)$ imply $P(n+1)$ for all $n \geq 1$ in the inductive step.

Proof

The proof is by strong induction. Let $P(n)$ be the proposition that every way of unstacking n blocks gives a score of $n(n-1)/2$.

Base case: If $n = 1$, then there is only one block. No moves are possible, and so the total score for the game is $1(1-1)/2 = 0$. Therefore, $P(1)$ is true.

Inductive step: Now we must show that $P(1), \ldots, P(n)$ imply $P(n+1)$ for all $n \geq 1$. So assume that $P(1), \ldots, P(n)$ are all true and that we have a stack of $n+1$ blocks. The first move must split this stack into substacks with positive sizes a and b where $a + b = n + 1$ and $0 < a, b \leq n$. Now the total score for the game is the sum of points for this first move plus points obtained by unstacking the two resulting substacks:

$$\begin{aligned} \text{total score} &= (\text{ score for 1 st move }) \\ &\quad + (\text{ score for unstacking } a \text{ blocks }) \\ &\quad + (\text{ score for unstacking } b \text{ blocks }) \\ &= ab + \frac{a(a-1)}{2} + \frac{b(b-1)}{2} \quad \text{by } P(a) \text{ and } P(b) \\ &= \frac{(a+b)^2 - (a+b)}{2} = \frac{(a+b)((a+b)-1)}{2} \\ &= \frac{(n+1)n}{2} \end{aligned} \tag{5.2.1}$$

This shows that $P(1), \ldots, P(n)$ imply $P(n+1)$.

Therefore, the claim is true by strong induction. ■

This page titled 5.2: Strong Induction is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

5.3: Strong Induction vs. Induction vs. Well Ordering

Strong induction looks genuinely “stronger” than ordinary induction —after all, you can assume a lot more when proving the induction step. Since ordinary induction is a special case of strong induction, you might wonder why anyone would bother with the ordinary induction.

But strong induction really isn’t any stronger, because a simple text manipulation program can automatically reformat any proof using strong induction into a proof using ordinary induction—just by decorating the induction hypothesis with a universal quantifier in a standard way. Still, it’s worth distinguishing these two kinds of induction, since which you use will signal whether the inductive step for $n+1$ follows directly from the case for n or requires cases smaller than n, and that is generally good for your reader to know.

The template for the two kinds of induction rules looks nothing like the one for the Well Ordering Principle, but this chapter included a couple of examples where induction was used to prove something already proved using well ordering. In fact, this can always be done. As the examples may suggest, any well ordering proof can automatically be reformatted into an induction proof. So theoretically, no one need bother with the Well Ordering Principle either.

But it’s equally easy to go the other way, and automatically reformat any strong induction proof into a Well Ordering proof. The three proof methods—well ordering, induction, and strong induction—are simply different formats for presenting the same mathematical reasoning!

So why three methods? Well, sometimes induction proofs are clearer because they don’t require proof by contradiction. Also, induction proofs often provide recursive procedures that reduce large inputs to smaller ones. On the other hand, well ordering can come out slightly shorter and sometimes seem more natural and less worrisome to beginners.

So which method should you use? There is no simple recipe. Sometimes the only way to decide is to write up a proof using more than one method and compare how they come out. But whichever method you choose, be sure to state the method up front to help a reader follow your proof.

This page titled 5.3: Strong Induction vs. Induction vs. Well Ordering is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

5.4: State Machines

State machines are a simple, abstract model of step-by-step processes. Since computer programs can be understood as defining step-by-step computational processes, it's not surprising that state machines come up regularly in computer science. They also come up in many other settings such as designing digital circuits and modeling probabilistic processes. This section introduces *Floyd's Invariant Principle* which is a version of induction tailored specifically for proving properties of state machines.

One of the most important uses of induction in computer science involves proving one or more desirable properties continues to hold at every step in a process. A property that is preserved through a series of operations or steps is known as a *preserved invariant* . Examples of desirable invariants include properties such as a variable never exceeding a certain value, the altitude of a plane never dropping below 1,000 feet without the wingflaps being deployed, and the temperature of a nuclear reactor never exceeding the threshold for a meltdown.

States and Transitions

Formally, a *state machine* is nothing more than a binary relation on a set, except that the elements of the set are called "states," the relation is called the transition relation, and an arrow in the graph of the transition relation is called a *transition*. A transition from state q to state r will be written $q \longrightarrow r$. The transition relation is also called the *state graph* of the machine. A state machine also comes equipped with a designated *start state*.

A simple example is a bounded counter, which counts from 0 to 99 and overflows at 100. This state machine is pictured in Figure 5.7, with states pictured as circles, transitions by arrows, and with start state 0 indicated by the double circle. To be precise, what the picture tells us is that this bounded counter machine has

$$\begin{aligned} \text{states} &::= \{0, 1, \ldots, 99, \text{ overflow}\}, \\ \text{start state} &::= 0, \\ \text{transitions} &::= \{n \longrightarrow n+1 \mid 0 \leq n < 99\} \\ &\quad \cup \{99 \longrightarrow \text{overflow}, \text{overflow} \longrightarrow \text{overflow}\} \end{aligned} \tag{5.4.1}$$

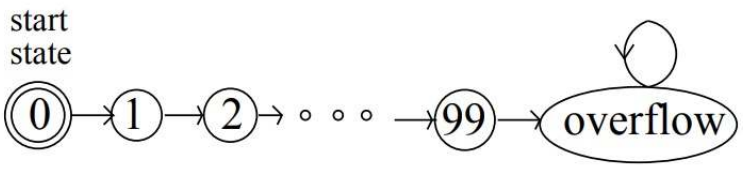


Figure 5.7 *State transitions for the 99-bounded counter.*

This machine isn't much use once it overflows, since it has no way to get out of its overflow state.

State machines for digital circuits and string pattern matching algorithms, for instance, usually have only a finite number of states. Machines that model continuing computations typically have an infinite number of states. For example, instead of the 99-bounded counter, we could easily define an "unbounded" counter that just keeps counting up without overflowing. The unbounded counter has an infinite state set, the nonnegative integers, which makes its state diagram harder to draw.

State machines are often defined with labels on states and/or transitions to indicate such things as input or output values, costs, capacities, or probabilities. Our state machines don't include any such labels because they aren't needed for our purposes. We do name states, as in Figure 5.7, so we can talk about them, but the names aren't part of the state machine.

Invariant for a Diagonally-Moving Robot

Suppose we have a robot that starts at the origin and moves on an infinite 2- dimensional integer grid. The *state* of the robot at any time can be specified by the integer coordinates x, y of the robot's current position. So the *start state* is (0, 0). At each step, the robot may move to a diagonally adjacent grid point, as illustrated in Figure 5.8.

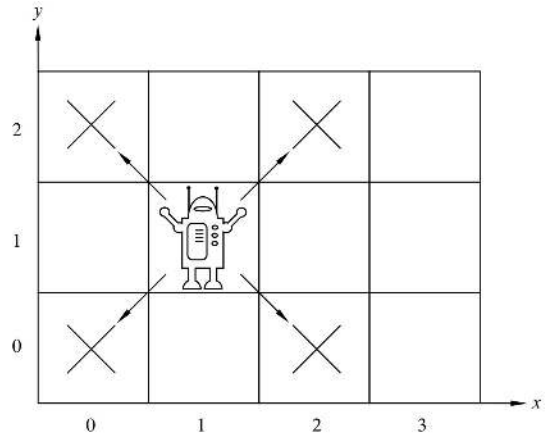


Figure 5.8 *The Diagonally Moving Robot.*

To be precise, the robot's transitions are:

$$\{(m,n) \longrightarrow (m \pm 1, n \pm 1) \mid m,n \in \mathbb{Z}\}$$

For example, after the first step, the robot could be in states (1, 1), (1, -1), (-1, 1), (-1, -1). After two steps, there are 9 possible states for the robot, including (0, 0). The question is, can the robot ever reach position (1, 0)?

If you play around with the robot a bit, you'll probably notice that the robot can only reach positions m,n for which $m+n$ is even, which of course means that it can't reach (1, 0). This follows because the evenness of the sum of the coordinates is preserved by transitions.

This once, let's go through this preserved-property argument, carefully highlighting where induction comes in. Specifically, define the even-sum property of states to be:

$$\textbf{Even-sum}\ ((m,n)) ::= [m+n \text{ is even }].$$

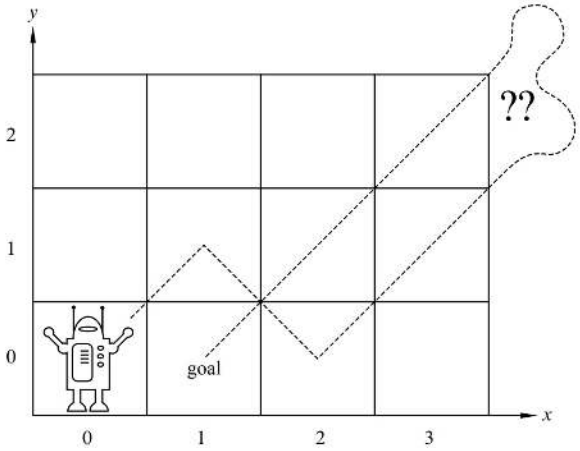


Figure 5.9 *Can the Robot get to (1, 0)?*

Lemma 5.4.1. *For any transition, $q \longrightarrow r$, of the diagonally-moving robot, if Even-sum(q), then Even-sum(r).*

This lemma follows immediately from the definition of the robot's transitions: $(m,n) \longrightarrow (m \pm 1, n \pm 1)$. After a transition, the sum of coordinates changes by $(\pm 1) + (\pm 1)$, that is, by 0, 2, or -2. Of course, adding 0, 2 or -2 to an even number gives an even number. So by a trivial induction on the number of transitions, we can prove:

Theorem 5.4.2

The sum of the coordinates of any state reachable by the diagonallymoving robot is even.

Proof

The proof is induction on the number of transitions the robot has made. The induction hypothesis is

$P(n) ::=$ **if q is a state reachable in n transitions, then Even-sum (q)**

Base case: $P(0)$ is true since the only state reachable in 0 transitions is the start state (0, 0), and 0 + 0 is even.

Inductive step: Assume that $P(n)$ is true, and let r be any state reachable in $n+1$ transitions. We need to prove that Even-sum(r) holds.

Since r is reachable in $n+1$ transitions, there must be a state, q, reachable in n transitions such that $q \longrightarrow r$. Since $P(n)$ is assumed to be true, Even-sum(q) holds, and so by Lemma 5.4.1, Even-sum(r) also holds. This proves that $P(n)$ **IMPLIES** $P(n+1)$ as required, completing the proof of the inductive step.

We conclude by induction that for all $n \geq 0$, if q is reachable in n transitions, then Even-sum(q). This implies that every reachable state has the Even-sum property. ■

Corollary 5.4.3. *The robot can never reach position (1, 0).*

Proof. By Theorem 5.4.2, we know the robot can only reach positions with coordinates that sum to an even number, and thus it cannot reach position (1, 0). ■

The Invariant Principle

Using the Even-sum invariant to understand the diagonally-moving robot is a simple example of a basic proof method called The Invariant Principle. The Principle summarizes how induction on the number of steps to reach a state applies to invariants.

A state machine *execution* describes a possible sequence of steps a machine might take.

Definition 5.4.4

An *execution* of the state machine is a (possibly infinite) sequence of states with the property that it begins with the start state, and

- it begins with the start state, and
- if q and r are consecutive states in the sequence, then $q \longrightarrow r$.

A state is called *reachable* if it appears in some execution.

Definition 5.4.5

A *preserved invariant* of a state machine is a predicate, P, on states, such that whenever $P(q)$ is true of a state, q, and $q \longrightarrow r$ for some state, r, then $P(r)$ holds.

The Invariant Principle

If a preserved invariant of a state machine is true for the start state, then it is true for all reachable states.

The Invariant Principle is nothing more than the Induction Principle reformulated in a convenient form for state machines. Showing that a predicate is true in the start state is the base case of the induction, and showing that a predicate is a preserved invariant corresponds to the inductive step.[5]

Robert W. Floyd



The Invariant Principle was formulated by Robert W. Floyd at Carnegie Tech in 1967. (Carnegie Tech was renamed Carnegie-Mellon University the following year.) Floyd was already famous for work on the formal grammars that transformed the field of programming language parsing; that was how he got to be a professor even though he never got a Ph.D. (He had been admitted to a PhD program as a teenage prodigy, but flunked out and never went back.)

In that same year, Albert R. Meyer was appointed Assistant Professor in the Carnegie Tech Computer Science Department, where he first met Floyd. Floyd and Meyer were the only theoreticians in the department, and they were both delighted to talk about their shared interests. After just a few conversations, Floyd's new junior colleague decided that Floyd was the smartest person he had ever met.

Naturally, one of the first things Floyd wanted to tell Meyer about was his new, as yet unpublished, Invariant Principle. Floyd explained the result to Meyer, and Meyer wondered (privately) how someone as brilliant as Floyd could be excited by such a trivial observation. Floyd had to show Meyer a bunch of examples before Meyer understood Floyd's excitement —not at the truth of the utterly obvious Invariant Principle, but rather at the insight that such a simple method could be so widely and easily applied in verifying programs.

Floyd left for Stanford the following year. He won the Turing award—the "Nobel prize" of computer science—in the late 1970's, in recognition of his work on grammars and on the foundations of program verification. He remained at Stanford from 1968 until his death in September, 2001. You can learn more about Floyd's life and work by reading the eulogy at

http://oldwww.acm.org/pubs/membernet...ries/floyd.pdf

written by his closest colleague, Don Knuth.

The Die Hard Example

The movie *Die Hard 3: With a Vengeance* includes an amusing example of a state machine. The lead characters played by Samuel L. Jackson and Bruce Willis have to disarm a bomb planted by the diabolical Simon Gruber:

Simon: On the fountain, there should be 2 jugs, do you see them? A 5- gallon and a 3-gallon. Fill one of the jugs with exactly 4 gallons of water and place it on the scale and the timer will stop. You must be precise; one ounce more or less will result in detonation. If you're still alive in 5 minutes, we'll speak.

Bruce: Wait, wait a second. I don't get it. Do you get it?

Samuel: No.

Bruce: Get the jugs. Obviously, we can't fill the 3-gallon jug with 4 gallons of water.

Samuel: Obviously.

Bruce: All right. I know, here we go. We fill the 3-gallon jug exactly to the top, right?

Samuel: Uh-huh.

Bruce: Okay, now we pour this 3 gallons into the 5-gallon jug, giving us exactly 3 gallons in the 5-gallon jug, right?

Samuel: Right, then what?

Bruce: All right. We take the 3-gallon jug and fill it a third of the way...

Samuel: No! He said, “Be precise.” Exactly 4 gallons.

Bruce: Sh - -. Every cop within 50 miles is running his a - - off and I’m out here playing kids games in the park.

Samuel: Hey, you want to focus on the problem at hand?

Fortunately, they find a solution in the nick of time. You can work out how.

The Die Hard 3 State Machine

The jug-filling scenario can be modeled with a state machine that keeps track of the amount, b, of water in the big jug, and the amount, l, in the little jug. With the 3 and 5 gallon water jugs, the states formally will be pairs, b, l, of real numbers such that $0 \leq b \leq 5, 0 \leq l \leq 3$. (We can prove that the reachable values of b and l will be nonnegative integers, but we won’t assume this.) The start state is (0, 0), since both jugs start empty.

Since the amount of water in the jug must be known exactly, we will only consider moves in which a jug gets completely filled or completely emptied. There are several kinds of transitions:

1. Fill the little jug: $(b, l) \longrightarrow (b, 3)$ for $l < 3$.
2. Fill the big jug: $(b, l) \longrightarrow (5, l)$ for $b < 5$.
3. Empty the little jug: $(b, l) \longrightarrow (b, 0)$ for $l > 0$.
4. Empty the big jug: $(b, l) \longrightarrow (0, l)$ for $b > 0$.
5. Pour from the little jug into the big jug: for $l > 0$,

$$(b, l) \longrightarrow \begin{cases} (b + l, 0) & \text{if } b + l \leq 5 \\ (5, l - (5 - b)) & \text{otherwise} \end{cases}$$

6. Pour form big jug into little jug: for $b > 0$,

$$(b, l) \longrightarrow \begin{cases} (0, b + l) & \text{if } b + l \leq 3 \\ (b - (3 - l), 3) & \text{otherwise} \end{cases}$$

Note that in contrast to the 99-counter state machine, there is more than one possible transition out of states in the Die Hard machine. Machines like the 99-counter with at most one transition out of each state are called *deterministic*. The Die Hard machine is *nondeterministic* because some states have transitions to several different states.

The Die Hard 3 bomb gets disarmed successfully because the state (4,3) is reachable.

Die Hard Once and For All

The *Die Hard* series is getting tired, so we propose a final *Die Hard Once and For All*. Here, Simon’s brother returns to avenge him, posing the same challenge, but with the 5 gallon jug replaced by a 9 gallon one. The state machine has the same specification as the Die Hard 3 version, except all occurrences of “5” are replaced by “9.”

Now, reaching any state of the form $(4, l)$ is impossible. We prove this using the Invariant Principle. Specifically, we define the preserved invariant predicate, $P((b, l))$, to be that b and l are nonnegative integer multiples of 3.

To prove that P is a preserved invariant of Die-Hard-Once-and-For-All machine, we assume $P(q)$ holds for some state $q ::= (b, l$ and that $q \longrightarrow r$. We have to show that $P(r)$ holds. The proof divides into cases, according to which transition rule is used.

One case is a “fill the little jug” transition. This means $r = (b, 3)$. But $P(q)$ implies that b is an integer multiple of 3, and of course 3 is an integer multiple of 3, so $P(r)$ still holds.

Another case is a “pour from big jug into little jug” transition. For the subcase when there isn’t enough room in the little jug to hold all the water, that is, when $b + l > 3$, we have $r = (b - (3 - l)), 3)$. But $P(q)$ implies that b and l are integer multiples of 3, which means $b - (3 - l)$ is too, so in this case too, $P(r)$ holds.

We won’t bother to crank out the remaining cases, which can all be checked just as easily. Now by the Invariant Principle, we conclude that every reachable state satisifies P. But since no state of the form $(4, l)$ satisifies P, we have proved rigorously that

Bruce dies once and for all!

By the way, notice that the state (1,0), which satisfies $\mathbf{NOT}(P)$, has a transition to (0,0), which satisfies P. So the negation of a preserved invariant may not be a preserved invariant.

Fast Exponentiation

Partial Correctness & Termination

Floyd distinguished two required properties to verify a program. The first property is called *partial correctness*; this is the property that the final results, if any, of the process must satisfy system requirements.

You might suppose that if a result was only partially correct, then it might also be partially incorrect, but that's not what Floyd meant. The word "partial" comes from viewing a process that might not terminate as computing a partial relation. Partial correctness means that *when there is a result*, it is correct, but the process might not always produce a result, perhaps because it gets stuck in a loop.

The second correctness property, called *termination*, is that the process does always produce some final value.

Partial correctness can commonly be proved using the Invariant Principle. Termination can commonly be proved using the Well Ordering Principle. We'll illustrate this by verifying a Fast Exponentiation procedure.

Exponentiating

The most straightforward way to compute the bth power of a number, a, is to multiply a by itself $b-1$ times. But the solution can be found in considerably fewer multiplications by using a technique called *Fast Exponentiation*. The register machine program below defines the fast exponentiation algorithm. The letters x, y, z, r denote registers that hold numbers. An *assignment statement* has the form "$z ::= a$" and has the effect of setting the number in register z to be the number a.

A Fast Exponentiation Program

Given inputs $a \in \mathbb{R}, b \in \mathbb{N}$, initialize registers x, y, z to $a, 1, b$ respectively, and repeat the following sequence of steps until termination:

- if $z = 0$ **return** y and terminate
- $r :=$ remainder $(z, 2)$
- $z :=$ quotient $(z, 2)$
- if $r = 1$, then $y := xy$
- $x := x^2$

We claim this program always terminates and leaves $y = a^b$.

To begin, we'll model the behavior of the program with a state machine:

1. states ::= $\mathbb{R} \times \mathbb{R} \times \mathbb{N}$,
2. start state ::= $(a, 1, b)$,
3. transitions are defined by the rule

$$(x, y, z) \longrightarrow \begin{cases} (x^2, y, \text{ quotient } (z, 2)) & \text{if } z \text{ is nonzero and even,} \\ (x^2, xy, \text{ quotient } (z, 2)) & \text{if } z \text{ is nonzero and odd.} \end{cases}$$

The preserved invariant, $P((x, y, z))$, will be

$$z \in \mathbb{N} \text{ AND } yx^z = a^b. \tag{5.4.2}$$

To prove that P is preserved, assume $P((x, y, z))$ holds and that $(x, y, z) \longrightarrow (x_t, y_t, z_t)$. We must prove that $P((x_t, y_t, z_t))$ holds, that is,

$$z_t \in \mathbb{N} \text{ AND } y_t x_t^{z_t} = a^b. \tag{5.4.3}$$

Since there is a transition from (x, y, z), we have $z \neq 0$, and since $z \in \mathbb{N}$ by (5.4), we can consider just two cases:

If z is even, then we have that $x_t = x^2$, $y_t = y$, $z_t = z/2$. Therefore, $z_t \in \mathbb{N}$ and

$$\begin{aligned} y_t x_t^{z_t} &= y\left(x^2\right)^{z/2} \\ &= yx^{2\cdot z/2} \\ &= yx^z \\ &= a^b \quad \text{(by (5.4.1))} \end{aligned} \tag{5.4.4}$$

If z is odd, then we have that $x_t = x^2$, $y_t = xy$, $z_t = (z-1)/2$. Therefore, $z_t \in \mathbb{N}$ and

$$\begin{aligned} y_t x_t^{z_t} &= xy\left(x^2\right)^{(z-1)/2} \\ &= yx^{1+2\cdot(z-1)/2} \\ &= yx^{1+(z-1)} \\ &= yx^z \\ &= a^b \quad \text{(by (5.4.1))} \end{aligned} \tag{5.4.5}$$

So in both cases, (5.4.3) holds, proving that P is a preserved invariant.

Now it's easy to prove partial correctness: if the Fast Exponentiation program terminates, it does so with a^b in register y. This works because $1 - a^b = a^b$, which means that the start state, $(a, 1, b)$, satisifies P. By the Invariant Principle, P holds for all reachable states. But the program only stops when $z = 0$. If a terminated state $(x, y, 0)$ is reachable, then $y = yx^0 = a^b$ as required.

Ok, it's partially correct, but what's fast about it? The answer is that the number of multiplications it performs to compute a^b is roughly the length of the binary representation of b. That is, the Fast Exponentiation program uses roughly $\log b$[6] multiplications, compared to the naive approach of multiplying by a a total of $b-1$ times.

More precisely, it requires at most $2(\lceil \log b \rceil + 1)$ multiplications for the Fast Exponentiation algorithm to compute a^b for $b > 1$. The reason is that the number in register z is initially b, and gets at least halved with each transition. So it can't be halved more than $\lceil \log b \rceil + 1$ times before hitting zero and causing the program to terminate. Since each of the transitions involves at most two multiplications, the total number of multiplications until $z = 0$ is at most $2(\lceil \log b \rceil + 1)$ for $b > 0$ (see Problem 5.36).

Derived Variables

The preceding termination proof involved finding a nonnegative integer-valued measure to assign to states. We might call this measure the "size" of the state. We then showed that the size of a state decreased with every state transition. By the Well Ordering Principle, the size can't decrease indefinitely, so when a minimum size state is reached, there can't be any transitions possible: the process has terminated.

More generally, the technique of assigning values to states—not necessarily nonnegative integers and not necessarily decreasing under transitions—is often useful in the analysis of algorithms. *Potential functions* play a similar role in physics. In the context of computational processes, such value assignments for states are called *derived variables*.

For example, for the Die Hard machines we could have introduced a derived variable, $f : \text{states} \to \mathbb{R}$, for the amount of water in both buckets, by setting $f((a, b)) ::= a + b$. Similarly, in the robot problem, the position of the robot along the x-axis would be given by the derived variable x-coord, where $x\text{-coord}((i, j)) ::= i$.

There are a few standard properties of derived variables that are handy in analyzing state machines.

Definition 5.4.6

A derived variable $f : \text{states} \to \mathbb{R}$ is *strictly decreasing* iff

$$q \longrightarrow q' \text{ IMPLIES } f(q') < f(q)$$

It is *weakly decreasing* iff

$$q \longrightarrow q' \text{ IMPLIES } f(q') \leq f(q)$$

Strictly increasing and *weakly increasing* derived variables are defined similarly. [7]

We confirmed termination of the Fast Exponentiation procedure by noticing that the derived variable z was nonnegative-integer-valued and strictly decreasing. We can summarize this approach to proving termination as follows:

Theorem 5.4.7

If f is a strictly decreasing $\mathbb{N}$-valued derived variable of a state machine, then the length of any execution starting at state q is at most $f(q)$.

Of course, we could prove Theorem 5.4.7 by induction on the value of $f(q)$, but think about what it says: "If you start counting down at some nonnegative integer $f(q)$, then you can't count down more than $f(q)$ times." Put this way, it's obvious.

Theorem 5.4.7 generalizes straightforwardly to derived variables taking values in a well ordered set (Section 2.4.

Theorem 5.4.8

If there exists a strictly decreasing derived variable whose range is a well ordered set, then every execution terminates.

Theorem 5.4.8 follows immediately from the observation that a set of numbers is well ordered iff it has no infinite decreasing sequences (Problem 2.17).

Note that the existence of a *weakly* decreasing derived variable does not guarantee that every execution terminates. An infinite execution could proceed through states in which a weakly decreasing variable remained constant.

A Southeast Jumping Robot (Optional)

Here's a contrived, simple example of proving termination based on a variable that is strictly decreasing over a well ordered set. Let's think about a robot positioned at an integer lattice-point in the Northeast quadrant of the plane, that is, at $(x, y) \in \mathbb{N}^2$.

At every second when it is away from the origin, (0, 0), the robot must make a move, which may be

- a unit distance West when it is not at the boundary of the Northeast quadrant (that is, $(x, y) \longrightarrow (x-1, y)$ for $x > 0$), or
- a unit distance South combined with an arbitrary jump East (that is, $(x, y) \longrightarrow (z, y-1)$ for $z \geq x$).

Claim 5.4.9. *The robot will always get stuck at the origin.*

If we think of the robot as a nondeterministic state machine, then Claim 5.4.9 is a termination assertion. The Claim may seem obvious, but it really has a different character than termination based on nonnegative integer-valued variables. That's because, even knowing that the robot is at position (0, 1), for example, there is no way to bound the time it takes for the robot to get stuck. It can delay getting stuck for as many seconds as it wants by making its next move to a distant point in the Far East. This rules out proving termination using Theorem 5.4.7.

So does Claim 5.4.9 still seem obvious?

Well it is if you see the trick. Define a derived variable, v, mapping robot states to the numbers in the well ordered set $\mathbb{N} + \mathbb{F}$ of Lemma 2.4.5. In particular, define $v : \mathbb{N}^2 \to \mathbb{N} + \mathbb{F}$ as follows

$$v(x, y) ::= y + \frac{x}{x+1}.$$

Now it's easy to check that if $(x, y) \longrightarrow (x', y')$ is a legitimate robot move, then $v((x', y')) < v((x, y))$. In particular, v is a strictly decreasing derived variable, so Theorem 5.4.8. implies that the robot always get stuck- even though we can't say how many moves it will take until it does.

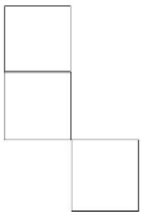


Figure 5.10 *Gehry's new tile.*

[5]Preserved invariants are commonly just called “invariants” in the literature on program correctness, but we decided to throw in the extra adjective to avoid confusion with other definitions. For example, other texts (as well as another subject at MIT) use “invariant” to mean “predicate true of all reachable states.” Let’s call this definition “invariant-2.” Now invariant-2 seems like a reasonable definition, since unreachable states by definition don’t matter, and all we want to show is that a desired property is invariant-2. But this confuses the *objective* of demonstrating that a property is invariant-2 with the *method* of finding a *preserved* invariant to show that it is invariant-2.

[6]As usual in computer science, $\log b$ means the base two logarithm, $\log_2 b$. We use, $\ln b$ for the natural logarithm $\log_e b$, and otherwise write the logarithm base explicitly, as in $\log_{10} b$.

[7]Weakly increasing variables are often also called *nondecreasing*. We will avoid this terminology to prevent confusion between nondecreasing variables and variables with the much weaker property of *not* being a decreasing variable.

This page titled 5.4: State Machines is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

6: Recursive Data Types

Recursive data types play a central role in programming, and induction is really all about them.

Recursive data types are specified by *recursive definitions*, which say how to construct new data elements from previous ones. Along with each recursive data type there are recursive definitions of properties or functions on the data type. Most importantly, based on a recursive definition, *there is a structural induction* method for proving that all data of the given type have some property.

This chapter examines a few examples of recursive data types and recursively defined functions on them:

- strings of characters,
- “balanced” strings of brackets,
- the nonnegative integers, and
- arithmetic expressions

6.1: Recursive Definitions and Structural Induction

6.2: Strings of Matched Brackets

6.3: Recursive Functions on Nonnegative Integers

6.4: Arithmetic Expressions

6.5: Induction in Computer Science

This page titled 6: Recursive Data Types is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

6.1: Recursive Definitions and Structural Induction

We'll start off illustrating recursive definitions and proofs using the example of character strings. Normally we'd take strings of characters for granted, but it's informative to treat them as a recursive data type. In particular, strings are a nice first example because you will see recursive definitions of things that are easy to understand or that you already know, so you can focus on how the definitions work without having to figure out what they are for.

Definitions of recursive data types have two parts:

- **Base case(s)** specifying that some known mathematical elements are in the data type, and
- **Constructor case(s)** that specify how to construct new data elements from previously constructed elements or from base elements.

The definition of strings over a given character set, A, follows this pattern:

Definition 6.1.1

Let A be a nonempty set called an *alphabet*, whose elements are referred to as *characters*, *letters*, or *symbols*. The recursive data type, A^*, of strings over alphabet, A, are defined as follows:

- **Base case**: the empty string, λ, is in A^*.
- **Constructor case**: If $a \in A$ and $s \in A^*$, then the pair $\langle a, s\rangle \in A^*$.

So $\{0,1\}^*$ are the binary strings.

The usual way to treat binary strings is as sequences of 0's and 1's. For example, we have identified the length-4 binary string 1011 as a sequence of bits, the 4-tuple (1, 0, 1, 1). But according to the recursive Definition 6.1.1, this string would be represented by nested pairs, namely

$$\langle 1, \langle 0, \langle 1, \langle 1, \lambda\rangle\rangle\rangle\rangle.$$

These nested pairs are definitely cumbersome and may also seem bizarre, but they actually reflect the way that such lists of characters would be represented in programming languages like Scheme or Python, where $\langle a, s\rangle$ would correspond to cons (a, s).

Notice that we haven't said exactly how the empty string is represented. It really doesn't matter, as long as we can recognize the empty string and not confuse it with any nonempty string.

Continuing the recursive approach, let's define the length of a string.

Definition 6.1.2

The length, $|s|$, of a string, s, is defined recursively based on the definition of $s \in A^*$:

Base case: $|\lambda| ::= 0$.

Constructor case: $|\langle a, s\rangle| ::= 1 + |s|$.

This definition of length follows a standard pattern: functions on recursive data types can be defined recursively using the same cases as the data type definition. Specifically, to define a function, f, on a recursive data type, define the value of f for the base cases of the data type definition, then define the value of f in each constructor case in terms of the values of f on the component data items.

Let's do another example: the *concatenation* $s \cdot t$ of the strings s and t is the string consisting of the letters of s followed by the letters of t. This is a perfectly clear mathematical definition of concatenation (except maybe for what to do with the empty string), and in terms of Scheme/Python lists, $s \cdot t$ would be the list append (s, t). Here's a recursive definition of concatenation.

Definition 6.1.3

The *concatenation* $s \cdot t$ of the strings $s, t \in A^*$ is defined recursively based on the definition of $s \in A^*$.

Base case:

$$\lambda \cdot t ::= t.$$

Constructor case:

$$\langle a, s\rangle \cdot t ::= \langle a, s \cdot t\rangle.$$

Structural Induction

Structural induction is a method for proving that all the elements of a recursively defined data type have some property. A structural induction proof has two parts corresponding to the recursive definition:

- Prove that each base case element has the property.
- Prove that each constructor case element has the property, when the constructor is applied to elements that have the property.

For example, we can verify the familiar fact that the length of the concatenation of two strings is the sum of their lengths using structural induction:

Theorem 6.1.4

For all $s, t \in A^*$,

$$|s \cdot t| = |s| + |t|.$$

Proof

By structural induction on the definition of $s \in A^*$. The induction hypothesis is

$$P(s) ::= \forall t \in A^*.\, |s \cdot t| = |s| + |t|.$$

Base case ($s = \lambda$):

$$\begin{aligned} |s \cdot t| &= |\lambda \cdot t| \\ &= |t| && (\text{def } \cdot,\ \text{base case}) \\ &= 0 + |t| \\ &= |s| + |t| && (\text{def length, base case}) \end{aligned} \tag{6.1.1}$$

Constructor case: Suppose $s ::= \langle a, r\rangle$ and assume the induction hypothesis, $P(r)$. We must show that $P(s)$ holds:

$$\begin{aligned} |s \cdot t| &= |\langle a, r\rangle \cdot t| \\ &= |\langle a, r \cdot t\rangle| && (\text{concat def, constructor case}) \\ &= 1 + |r \cdot t| && (\text{length def, constructor case}) \\ &= |1 + (|r| + |t|) && (\text{since } P(r) \text{ holds}) \\ &= (1 + |r|) + |t| \\ &= |\langle a, r\rangle| + |t| && (\text{length def, constructor case}) \\ &= |s| + |t|. \end{aligned} \tag{6.1.2}$$

This proves that $P(s)$ holds as required, completing the constructor case. By structural induction we conclude that $P(s)$ holds for all strings $s \in A^*$. ■

This proof illustrates the general principle:

The Principle of Structural Induction.

Let P be a predicate on a recursively defined data type R. If

- $P(b)$ is true for each base case element, $b \in R$, and
- for all two-argument constructors, $\mathbf{c}$,

$$|P(r) \text{ AND } P(s)| \text{ IMPLIES } P(\mathbf{c}(r,s))$$

for all $r, s \in R$,

and likewise for all constructors taking other numbers of arguments, then

$$P(r) \text{ is true for all } r \in R.$$

One More Thing

The number, $\#_c(s)$, of occurrences of the character $c \in A$ in the string s has a simple recursive definition based on the definition of $s \in A^*$:

Definition 6.1.5

Base case: $\#_c(\lambda) ::= 0$.

Constructor case:

$$\#_c(\langle a, s\rangle) ::= \begin{cases} \#_c(s) & \text{if } a \neq c \\ 1+\#_c(s) & \text{if } a = c \end{cases}$$

We'll need the following lemma in the next section:

$$\#_c(s \cdot t) = \#_c(s)+\#_c(t)$$

The easy proof by structural induction is an exercise (Problem 6.7).

This page titled 6.1: Recursive Definitions and Structural Induction is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

6.2: Strings of Matched Brackets

Let $\{[,]\}^*$ be the set of all strings of square brackets. For example, the following two strings are in $\{[,]\}^*$:

$$]][[[\quad \text{and} \quad [[]][] \tag{6.2.1}$$

A string, $s \in \{[,]\}^*$, is called a *matched string* if its brackets "match up" in the usual way. For example, the left hand string above is not matched because its second right bracket does not have a matching left bracket. The string on the right is matched.

We're going to examine several different ways to define and prove properties of matched strings using recursively defined sets and functions. These properties are pretty straightforward, and you might wonder whether they have any particular relevance in computer science. The honest answer is "not much relevance *any more*." The reason for this is one of the great successes of computer science, as explained in the text box below.

Expression Parsing

During the early development of computer science in the 1950's and 60's, creation of effective programming language compilers was a central concern. A key aspect in processing a program for compilation was expression parsing. One significant problem was to take an expression like

$$x + y * z^2 \div y + 7$$

and *put in* the brackets that determined how it should be evaluated—should it bes

$$[[x + y] * z^2 \div y] + 7, \text{ or}$$

$$x + [y * z^2 \div [y + 7]] \text{ , or,}$$

$$[x + [y * z^2]] \div [y + 7], \text{ or } ...?$$

The Turing award (the "Nobel Prize" of computer science) was ultimately bestowed on Robert W. Floyd, for, among other things, discovering simple procedures that would insert the brackets properly.

In the 70's and 80's, this parsing technology was packaged into high-level compiler-compilers that automatically generated parsers from expression grammars. This automation of parsing was so effective that the subject no longer demanded attention. It had largely disappeared from the computer science curriculum by the 1990's.

The matched strings can be nicely characterized as a recursive data type:

Definition 6.2.1

Recursively define the set, RecMatch, of strings as follows:

Base case: $\lambda \in \mathbf{RecMatch}$.

Constructor case: If $s, t \in \mathbf{RecMatch}$, then

$$[s]t \in \mathbf{RecMatch}.$$

Here $[s]t$ refers to the concatenation of strings which would be written in full as

$$[\cdot(s\cdot(] \cdot t)).$$

From now on, we'll usually omit the "$\cdot$'s."

Using this definition, $\lambda \in \mathbf{RecMatch}$ by the base case, so letting $s = t = \lambda$ in the constructor case implies

$$[\lambda]\lambda = [] \in \mathbf{RecMatch}.$$

Now,

$$\begin{array}{rl} [\lambda][] = [][] \in \text{RecMatch} & (\text{letting } s = \lambda, t = []) \\ [[]]\lambda = [[]] \in \text{RecMatch} & (\text{letting } s = [], t = \lambda) \\ [[]][] \in \text{RecMatch} & (\text{letting } s = [], t = []) \end{array} \tag{6.2.2}$$

are also strings in RecMatch by repeated applications of the constructor case; and so on.

It's pretty obvious that in order for brackets to match, there had better be an equal number of left and right ones. For further practice, let's carefully prove this from the recursive definitions.

Lemma. *Every string in RecMatch has an equal number of left and right brackets.*

Proof. The proof is by structural induction with induction hypothesis

$$P(s) ::= \#_{[}(s) = \#_{]}(s).$$

Base case: $P(\lambda)$ holds because

$$\#_{[}(\lambda) = 0 = \#_{]}(\lambda)$$

by the base case of Definition 6.1.5 of $\#_c()$.

Constructor case: By structural induction hypothesis, we assume $P(s)$ and $P(t)$ and must show $P([s]t)$:

$$\begin{array}{lll} \#_{[}([s]t) & = \#_{[}([) + \#_{[}(s) + \#_{[}(]) + \#_{[}(t) & (\text{Lemma 6.1.6}) \\ & = 1 + \#_{[}(s) + 0 + \#_{[}(t) & (\text{def } \#_{[}()) \\ & = 1 + \#_{]}(s) + 0 + \#_{]}(t) & (\text{by } P(s) \text{ and } P(t)) \\ & = 0 + \#_{]}(s) + 0 + \#_{]}(t) & \\ & = \#_{]}([) + \#_{]}(s) + \#_{]}(]) + \#_{]}(t) & (\text{def } \#_{]}()) \\ & = \#_{]}([s]t) & (\text{Lemma 6.1.6}) \end{array}$$

This completes the proof of the constructor case. We conclude by structural induction that $P(s)$ holds for all $s \in \text{RecMatch}$. ■

Warning: When a recursive definition of a data type allows the same element to be constructed in more than one way, the definition is said to be *ambiguous*. We were careful to choose an *un*ambiguous definition of RecMatch to ensure that functions defined recursively on its definition would always be well-defined. Recursively defining a function on an ambiguous data type definition usually will not work. To illustrate the problem, here's another definition of the matched strings.

Definition 6.2.2

Define the set, $\text{AmbRecMatch} \subseteq \{[], []\}^*$ recursively as follows:

Base case: $\lambda \in \text{AmbRecMatch}$,

Constructor cases: if $s, t \in \text{AmbRecMatch}$, then the strings $[s]$ and st are also in AmbRecMatch.

It's pretty easy to see that the definition of AmbRecMatch is just another way to define RecMatch, that is AmbRecMatch D RecMatch (see Problem 6.15). The definition of AmbRecMatch is arguably easier to understand, but we didn't use it because it's ambiguous, while the trickier definition of RecMatch is unambiguous. Here's why this matters. Let's define the number of operations, $f(s)$, to construct a matched string s recursively on the definition of $s \in \text{AmbRecMatch}$:

$$\begin{array}{ll} f(\lambda) ::= 0, & (f \text{ base case }) \\ f([s]) ::= 1 + f(s), & \\ f(st) := 1 + f(s) + f(t). & (f \text{ concat case }) \end{array} \tag{6.2.3}$$

This definition may seem ok, but it isn't: $f(\lambda)$ winds up with two values, and consequently:

$$\begin{array}{lll} 0 & = f(\lambda) & (f \text{ base case })) \\ & = f(\lambda \cdot \lambda) & (\text{concat def, base case}) \\ & = 1 + f(\lambda) + f(\lambda) & (f \text{ concat case}), \\ & = 1 + 0 + 0 = 1 & (f \text{ base case}). \end{array}$$

This is definitely not a situation we want to be in!

This page titled 6.2: Strings of Matched Brackets is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

6.3: Recursive Functions on Nonnegative Integers

The nonnegative integers can be understood as a recursive data type.

Definition 6.3.1

The set, \mathbb{N}, is a data type defined recursively as:

- $0 \in \mathbb{N}$.
- If $n \in \mathbb{N}$, then the *successor*, $n+1$, of n is in $\mathbb{N}$.

The point here is to make it clear that ordinary induction is simply the special case of structural induction on the recursive Definition 6.3.1. This also justifies the familiar recursive definitions of functions on the nonnegative integers.

Some Standard Recursive Functions on $\mathbb{N}$

Example 6.3.2. The factorial function. This function is often written "$n!$." You will see a lot of it in later chapters. Here, we'll use the notation $\mathbf{fac}(n)$:

Example 6.3.2 *The factorial function*

This function is often written "$n!$." You will see a lot of it in later chapters. Here, we'll use the notation fac(n):

- $\mathbf{fac}(0) ::= 1$.
- $\mathbf{fac}(n+1) ::= (n+1) \cdot \mathbf{fac}(n)$ for $n \geq 0$.

Example 6.3.3 *The Fibonacci numbers*

Fibonacci numbers arose out of an effort 800 years ago to model population growth. They have a continuing fan club of people captivated by their extraordinary properties (see Problems 5.8, 5.21, 5.26). The nth Fibonacci number, fib, can be defined recursively by:

$$\begin{aligned} F(0) &::= 0, \\ F(1) &::= 1, \\ F(n) &::= F(n-1) + F(n-2) \quad \text{for } n \geq 2. \end{aligned}$$

Here the recursive step starts at $n = 2$ with base cases for 0 and 1. This is needed since the recursion relies on two previous values.

What is $F(4)$? Well, $F(2) = F(1) + F(0) = 1, F(3) = F(2) + F(1) = 2$, so $F(4) = 3$. The sequence starts out $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

Example 6.3.4. *Summation notation*

Let "$S(n)$" abbreviate the expression "$\sum_{i=1}^{n} f(i)$." We can recursively define $S(n)$ with the rules

- $S(0) ::= 0$
- $S(n+1) ::= f(n+1) + S(n)$ for $n \geq 0$.

Ill-formed Function Definitions

There are some other blunders to watch out for when defining functions recursively. The main problems come when recursive definitions don't follow the recursive definition of the underlying data type. Below are some function specifications that resemble good definitions of functions on the nonnegative integers, but really aren't.

$$f_1(n) ::= 2 + f_1(n-1). \tag{6.3.1}$$

This "definition" has no base case. If some function, f_1, satisfied (6.3.1), so would a function obtained by adding a constant to the value of f_1. So equation (6.3.1) does not uniquely define an f_1.

$$f_2(n) ::= \begin{cases} 0, & \text{if } n = 0 \\ f_2(n+1) & \text{otherwise} \end{cases} \tag{6.3.2}$$

This "definition" has a base case, but still doesn't uniquely determine f_2. Any function that is 0 at 0 and constant everywhere else would satisfy the specification, so (6.3.2) also does not uniquely define anything.

In a typical programming language, evaluation of $f_2(1)$ would begin with a recursive call of $f_2(2)$, which would lead to a recursive call of $f_2(3), \ldots$ with recursive calls continuing without end. This "operational" approach interprets (6.3.2) as defining a *partial* function, f_2, that is undefined everywhere but 0 .

$$f_3(n) ::= \begin{cases} 0, & \text{if } n \text{ is divisible by 2} \\ 1, & \text{if } n \text{ is divisible by 3} \\ 2, & \text{otherwise} \end{cases} \tag{6.3.3}$$

This "definition" is inconsistent: it requires $f_3(6) = 0$ and $f_3(6) = 1$, so (6.3.3) doesn't define anything.

Mathematicians have been wondering about this function specification, known as the Collatz conjecture for a while:

$$f_4(n) ::= \begin{cases} 1, & \text{if } n \leq 1 \\ f_4(n/2) & \text{if } n > 1 \text{ is even} \\ f_4(3n+1) & \text{if } n > 1 \text{ is odd} \end{cases} \tag{6.3.4}$$

For example, $f_4(3) = 1$ because

$$f_4(3) ::= f_4(10) ::= f_4(5) ::= f_4(16) ::= f_4(8) ::= f_4(4) ::= f_4(2) ::= f_4(1) ::= 1.$$

The constant function equal to 1 will satisfy (6.3.4), but it's not known if another function does as well. The problem is that the third case specifies $f_4(n)$ in terms of f_4 at arguments larger than n, and so cannot be justified by induction on $\mathbb{N}$. It's known that any f_4 satisfying (6.3.4) equals 1 for all n up to over 10^{18}.

A final example is the Ackermann function, which is an extremely fast-growing function of two nonnegative arguments. Its inverse is correspondingly slow-growingit grows slower than $\log n, \log\log n, \log\log\log n, \ldots$, but it does grow unboundly. This inverse actually comes up analyzing a useful, highly efficient procedure known as the *Union-Find algorithm*. This algorithm was conjectured to run in a number of steps that grew linearly in the size of its input, but turned out to be "linear" but with a slow growing coefficient nearly equal to the inverse Ackermann function. This means that pragmatically, *Union-Find* is linear, since the theoretically growing coefficient is less than 5 for any input that could conceivably come up.

The Ackermann function can be defined recursively as the function, A, given by the following rules:

$$A(m,n) = 2n \qquad \text{if } m = 0 \text{ or } n \leq 1, \tag{6.3.5}$$

$$A(m,n) = A(m-1, A(m, n-1)), \qquad \text{otherwise.} \tag{6.3.6}$$

Now these rules are unusual because the definition of $A(m,n)$ involves an evaluation of A at arguments that may be a lot bigger than m and n. The definitions of f_2 above showed how definitions of function values at small argument values in terms of larger one can easily lead to nonterminating evaluations. The definition of the Ackermann function is actually ok, but proving this takes some ingenuity (see Problem 6.17).

This page titled 6.3: Recursive Functions on Nonnegative Integers is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

6.4: Arithmetic Expressions

Expression evaluation is a key feature of programming languages, and recognition of expressions as a recursive data type is a key to understanding how they can be processed.

To illustrate this approach we'll work with a toy example: arithmetic expressions like $3x^2 + 2x + 1$ involving only one variable, "x." We'll refer to the data type of such expressions as Aexp. Here is its definition:

Definition 6.4.1

- **Base cases**:
  - The variable, x, is in Aexp.
  - The arabic numeral, $\mathbf{k}$, for any nonnegative integer, k, is in Aexp.
- **Constructor cases**: If $e, f \in \mathbf{Aexp}$, then
  - $[e + f] \in \mathbf{Aexp}$. The expression $[e + f]$ is called a *sum*. The Aexp's e and f are called the *components* of the sum; they're also called the *summands*.
  - $[e * f] \in \mathbf{Aexp}$. The expression $[e * f]$ is called a *product*. The Aexp's e and f are called the *components* of the product; they're also called the *multiplier* and *multiplicand*.
  - $-[e] \in \mathbf{Aexp}$. The expression $-[e] \in \mathbf{Aexp}$ is called a *negative*.

- Notice that Aexp's are fully bracketed, and exponents aren't allowed. So the Aexp version of the polynomial expression $3x^2 + 2x + 1$ would officially be written as

$$[[3 * [x * x]] + [[2 * x] + 1]]. \tag{6.4.1}$$

These brackets and *'s clutter up examples, so we'll often use simpler expressions like "$3x^2 + 2x + 1$" instead of (6.8). But it's important to recognize that $3x^2 + 2x + 1$ is not an Aexp; it's an *abbreviation* for an Aexp.

Evaluation and Substitution with Aexp's

Evaluating Aexp's

Since the only variable in an Aexp is x, the value of an Aexp is determined by the value of x. For example, if the value of x is 3, then the value of $3x^2 + 2x + 1$ is 34. In general, given any Aexp, e, and an integer value, n, for the variable, x, we can evaluate e to finds its value, $\text{eval}(e, n)$. It's easy, and useful, to specify this evaluation process with a recursive definition.

Definition 6.4.2

The *evaluation function*, $\text{eval} : \mathbf{Aexp} \times \mathbb{Z} \to \mathbb{Z}$, is defined recursively on expressions, $e \in \mathbf{Aexp}$, as follows. Let n be any integer.

- **Base cases**:

$$\text{eval}(x, n) ::= n, \qquad (\text{value of variable } x \text{ is } n.) \tag{6.4.2}$$
$$\text{eval}(\mathbf{k}, n) ::= k, \qquad (\text{value of numeral } \mathbf{k} \text{ is } k, \text{ regardless of } x.) \tag{6.4.3}$$

- **Constructor cases**:

$$\text{eval}([e_1 + e_2], n) := \text{eval}(e_1, n) + \text{eval}(e_2, n) \tag{6.4.4}$$
$$\text{eval}([e_1 * e_2], n) := \text{eval}(e_1, n) \cdot \text{eval}(e_2, n) \tag{6.4.5}$$
$$\text{eval}(-[e_1], n) := -\text{eval}(e_1, n). \tag{6.4.6}$$

For example, here's how the recursive definition of eval would arrive at the value of $3 + x^2$ when x is 2:

$$\begin{aligned}
\text{eval}([3 + [x * x]], 2) &= \text{eval}(3, 2) + \text{eval}([x * x], 2) && (\text{by Def 6.4.2 6.4.4}) \\
&= 3 + \text{eval}([x * x], 2) && (\text{by Def 6.4.2 6.4.3}) \\
&= 3 + (\text{eval}(x, 2) \cdot \text{eval}(x, 2)) && (\text{by Def 6.4.2 6.4.5}) \\
&= 3 + (2 \cdot 2) && (\text{by Def 6.4.2 6.4.2}) \\
&= 3 + 4 = 7.
\end{aligned} \tag{6.4.7}$$

Substituting into Aexp's

Substituting expressions for variables is a standard operation used by compilers and algebra systems. For example, the result of substituting the expression $3x$ for x in the expression $x(x-1)$ would be $3x(3x-1)$. We'll use the general notation $\mathbf{subst}(f, e)$ for the result of substituting an $\mathbf{Aexp}$, f, for each of the x 's in an Aexp, e. So as we just explained,

$$\mathbf{subst}(3x, x(x-1)) = 3x(3x-1).$$

This substitution function has a simple recursive definition:

Definition 6.4.3

The *substitution function* from $\mathbf{Aexp} \times \mathbf{Aexp}$ to Aexp is defined recursively on expressions, $e \in \mathbf{Aexp}$, as follows. Let f be any Aexp.

- **Base cases**:

$$\mathbf{subst}(f, x) ::= f, \quad (\text{subbing } f \text{ for variable }, x, \text{ just gives } f) \tag{6.4.8}$$
$$\mathbf{subst}(f, \mathrm{k}) ::= \mathrm{k}, \quad (\text{subbing into a numeral does nothing.}) \tag{6.4.9}$$

- **Constructor cases**:

$$\mathbf{subst}(f, [e_1 + e_2]) := [\mathbf{subst}(f, e_1) + \mathbf{subst}(f, e_2)] \tag{6.4.10}$$
$$\mathbf{subst}(f, [e_1 * e_2]) := [\mathbf{subst}(f, e_1) * \mathbf{subst}(f, e_2)] \tag{6.4.11}$$
$$\mathbf{subst}(f, -[e_1]) := -[\mathbf{subst}(f, e_1)] \tag{6.4.12}$$

Here's how the recursive definition of the substitution function would find the result of substituting $3x$ for x in the $x(x-1)$:

Misplaced '&'

Now suppose we have to find the value of $\mathbf{subst}(3x, x(x-1))$ when $x = 2$. There are two approaches.

First, we could actually do the substitution above to get $3x(3x-1)$, and then we could evaluate $3x(3x-1)$ when $x = 2$, that is, we could recursively calculate $\mathbf{eval}(3x(3x-1), 2)$ to get the final value 30 . This approach is described by the expression

$$\mathbf{eval}(\mathbf{subst}(3x, x(x-1)), 2) \tag{6.4.13}$$

In programming jargon, this would be called evaluation using the *Substitution Model*. With this approach, the formula $3x$ appears twice after substitution, so the multiplication $3 \cdot 2$ that computes its value gets performed twice.

The other approach is called evaluation using the *Environment Model*. Namely, to compute the value of (6.4.13), we evaluate $3x$ when $x = 2$ using just 1 multiplication to get the value 6 . Then we evaluate $x(x-1)$ when x has this value 6 to arrive at the value $6 \cdot 5 = 30$. This approach is described by the expression

$$\mathbf{eval}(x(x-1), \mathbf{eval}(3x, 2)). \tag{6.4.14}$$

The Environment Model only computes the value of $3x$ once, and so it requires one fewer multiplication than the Substitution model to compute (6.4.14). This is a good place to stop and work this example out yourself (Problem 6.18).

But how do we know that these final values reached by these two approaches, that is, the final integer values of (6.4.13) and (6.4.14), agree? In fact, we can prove pretty easily that these two approaches *always* agree by structural induction on the definitions of the two approaches. More precisely, what we want to prove is

Theorem 6.4.4

For all expressions $e, f \in \mathbf{Aexp}$ *and* $n \in \mathbb{N}$,

$$\mathbf{eval}(\mathbf{subst}(f, e), n) = \mathbf{eval}(e, \mathbf{eval}(f, n)) \tag{6.4.15}$$

Proof

The proof is by structural induction on e.[1]

Base cases:

Case[x]

The left hand side of equation (6.4.15) equals $\mathbf{eval}(f, n)$ by this base case in Definition 6.4.3 of the substitution function, and the right hand side also equals $\mathbf{eval}(f, n)$ by this base case in Definition 6.4.2 of eval.

Case[$\mathbf{k}$]

The left hand side of equation (6.4.15) equals $\mathbf{k}$ by this base case in Definitions 6.4.3 and 6.4.2 of the substitution and evaluation functions. Likewise, the right hand side equals $\mathbf{k}$ by two applications of this base case in the Definition 6.4.2 of eval.

Constructor cases:

Case [$[e_1 + e_2]$]

By the structural induction hypothesis (6.4.15), we may assume that for all $f \in \mathbf{Aexp}$ and $n in \mathbb{Z}$,

$$\mathbf{eval}(\mathbf{subst}(f, e_i), n) = \mathbf{eval}(e_i, \mathbf{eval}(f, n)) \tag{6.4.16}$$

for $i = 1, 2$. We wish to prove that

$$\mathbf{eval}(\mathbf{subst}(f, [e_1 + e_2]), n) = \mathbf{eval}([e_1 + e_2], \mathbf{eval}(f, n)) \tag{6.4.17}$$

The left hand side of (6.4.17) equals

$$\mathbf{eval}([\mathbf{subst}(f, e_1)) + \mathbf{subst}(f, e_2)], n)$$

by Definition 6.4.3 6.4.10 of substitution into a sum expression. But this equals

$$\mathbf{eval}(\mathbf{subst}(f, e_1), n) + \mathbf{eval}(\mathbf{subst}(f, e_2), n)$$

by Definition 6.4.2. (6.4.4) of eval for a sum expression. By induction hypothesis (6.4.16), this in turn equals

$$\mathbf{eval}(e_1, \mathbf{eval}(f, n)) + \mathbf{eval}(e_2, \mathbf{eval}(f, n))$$

Finally, this last expression equals the right hand side of (6.4.17) by Definition 6.4.2.(6.4.4) of eval for a sum expression. This proves (6.4.17) in this case.

- Case [$[e_1 * e_2]$] Similar.
- Case [$-[e_1]$] Even easier.

This covers all the constructor cases, and so completes the proof by structural induction. ■

[1]This is an example of why it's useful to notify the reader what the induction variable is—in this case it isn't n.

This page titled 6.4: Arithmetic Expressions is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

6.5: Induction in Computer Science

Induction is a powerful and widely applicable proof technique, which is why we've devoted two entire chapters to it. Strong induction and its special case of ordinary induction are applicable to any kind of thing with nonnegative integer sizes—which is an awful lot of things, including all step-by-step computational processes.

Structural induction then goes beyond number counting, and offers a simple, natural approach to proving things about recursive data types and recursive computation.

In many cases, a nonnegative integer size can be defined for a recursively defined datum, such as the length of a string, or the number of operations in an Aexp. It is then possible to prove properties of data by ordinary induction on their size. But this approach often produces more cumbersome proofs than structural induction.

In fact, structural induction is theoretically more powerful than ordinary induction. However, it's only more powerful when it comes to reasoning about infinite data types—like infinite trees, for example—so this greater power doesn't matter in practice. What does matter is that for recursively defined data types, structural induction is a simple and natural approach. This makes it a technique every computer scientist should embrace.

This page titled 6.5: Induction in Computer Science is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

7: Infinite Sets

This chapter is about infinite sets and some challenges in proving things about them.

Wait a minute! Why bring up infinity in a Mathematics for *Computer Science* text? After all, any data set in a computer is limited by the size of the computer's memory, and there is a bound on the possible size of computer memory, for the simple reason that the universe is (or at least appears to be) bounded. So why not stick with *finite* sets of some large, but bounded, size? This is a good question, but let's see if we can persuade you that dealing with infinite sets is inevitable.

You may not have noticed, but up to now you've already accepted the routine use of the integers, the rationals and irrationals, and sequences of them—infinite sets, all. Further, do you really want Physics or the other sciences to give up the real numbers on the grounds that only a bounded number of bounded measurements can be made in a bounded universe? It's pretty convincing—and a lot simpler—to ignore such big and uncertain bounds (the universe seems to be getting bigger all the time) and accept theories using real numbers.

Likewise in computer science, it's implausible to think that writing a program to add nonnegative integers with up to as many digits as, say, the stars in the sky— billions of galaxies each with billions of stars—would be different from writing a program that would add *any* two integers, no matter how many digits they had. The same is true in designing a compiler: it's neither useful nor sensible to make use of the fact that in a bounded universe, only a bounded number of programs will ever be compiled.

Infinite sets also provide a nice setting to practice proof methods, because it's harder to sneak in unjustified steps under the guise of intuition. And there has been a truly astonishing outcome of studying infinite sets. Their study led to the discovery of fundamental, logical limits on what computers can possibly do. For example, in Section 7.2, we'll use reasoning developed for infinite sets to prove that it's impossible to have a perfect type-checker for a programming language.

So in this chapter, we ask you to bite the bullet and start learning to cope with infinity.

7.1: Infinite Cardinality

7.2: The Halting Problem

7.3: The Logic of Sets

7.4: Does All This Really Work?

This page titled 7: Infinite Sets is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

7.1: Infinite Cardinality

uIn the late nineteenth century, the mathematician Georg Cantor was studying the convergence of Fourier series and found some series that he wanted to say converged "most of the time," even though there were an infinite number of points where they didn't converge. As a result, Cantor needed a way to compare the size of infinite sets. To get a grip on this, he got the idea of extending the Mapping Rule Theorem 4.5.4 to infinite sets: he regarded two infinite sets as having the "same size" when there was a bijection between them. Likewise, an infinite set A should be considered "as big as" a set B when A surj B. So we could consider A to be "strictly smaller" than B, which we abbreviate as A strict B, when A is *not* "as big as" B:

Definition 7.1.1

$$A \text{ strict } B \quad \textit{iff} \quad \text{NOT}(A \text{ surj } B).$$

On finite sets, this strict relation really does mean "strictly smaller." This follows immediately from the Mapping Rule Theorem 4.5.4.

Corollary 7.1.2. *For finite sets A, B,*

$$A \text{ strict } B \quad \textit{iff} \quad |A| < |B|.$$

Proof.

$$\begin{array}{rll} A \text{ strict } B \text{ iff} & \text{NOT}(A \text{ surj } B) & (\text{Def 7.1.1.}) \\ \text{iff} & \text{NOT}(|A| \geq |B|) & (\text{Theorem 4.5.4. (4.5.1)}) \\ \text{iff} & |A| < |B|. & \blacksquare \end{array} \tag{7.1.1}$$

Cantor got diverted from his study of Fourier series by his effort to develop a theory of infinite sizes based on these ideas. His theory ultimately had profound consequences for the foundations of mathematics and computer science. But Cantor made a lot of enemies in his own time because of his work: the general mathematical community doubted the relevance of what they called "Cantor's paradise" of unheard-of infinite sizes.

A nice technical feature of Cantor's idea is that it avoids the need for a definition of what the "size" of an infinite set might be—all it does is compare "sizes."

Warning: We haven't, and won't, define what the "size" of an infinite set is. The definition of infinite "sizes" requires the definition of some infinite sets called *ordinals* with special well-ordering properties. The theory of ordinals requires getting deeper into technical set theory than we want to go, and we can get by just fine without defining infinite sizes. All we need are the "as big as" and "same size" relations, surj and bij, between sets.

But there's something else to watch out for: we've referred to surj as an "as big as" relation and bij as a "same size" relation on sets. Of course, most of the "as big as" and "same size" properties of surj and bij on finite sets do carry over to infinite sets, but *some important ones don't*—as we're about to show. So you have to be careful: don't assume that surj has any particular "as big as" property on *infinite* sets until it's been proved.

Let's begin with some familiar properties of the "as big as" and "same size" relations on finite sets that do carry over exactly to infinite sets:

Lemma 7.1.3. *For any sets, A, B, C,*

1. A surj B *iff* B inj A.
2. *If* A surj B *and* B surj C, *then* A surj C.
3. *If* A bij B *and* B bij C, *then* A bij C.
4. A bij B *iff* B bij A.

Part 1. follows from the fact that R has the $[\leq 1 \text{ out}, \geq 1 \text{ in}]$ surjective function property iff R^{-1} has the $[\geq 1 \text{ out}, \leq 1 \text{ in}]$ total, injective property. Part 2. follows from the fact that compositions of surjections are surjections. Parts 3. and 4. follow from the first two parts because R is a bijection iff R and R^{-1} are surjective functions. We'll leave verification of these facts to Problem 4.22.

Another familiar property of finite sets carries over to infinite sets, but this time some real ingenuity is needed to prove it:

Theorem 7.1.4

[Schröder-Bernstein] For any sets A, B, if A surj B and B surj A, then A bij B.

That is, the Schröder-Bernstein Theorem says that if A is at least as big as B and conversely, B is at least as big as A, then A is the same size as B. Phrased this way, you might be tempted to take this theorem for granted, but that would be a mistake. For infinite sets A and B, the Schröder-Bernstein Theorem is actually pretty technical.

Just because there is a surjective function $f : A \to B$—which need not be a bijection—and a surjective function $g : B \to A$—which also need not be a bijection—it's not at all clear that there must be a bijection $e : A \to B$. The idea is to construct e from parts of both f and g. We'll leave the actual construction to Problem 7.11.

Another familiar set property is that for any two sets, either the first is at least as big as the second, or vice-versa. For finite sets this follows trivially from the Mapping Rule. It's actually still true for infinite sets, but assuming it was obvious would be mistaken again.

Theorem 7.1.5

For all *sets A, B,*

$$A \text{ surj } B \text{ OR } B \text{ surj } A$$

Theorem 7.1.5 lets us prove that another basic property of finite sets carries over to infinite ones:

Lemma 7.1.6.

$$A \text{ strict } B \text{ AND } B \text{ strict } C \tag{7.1.2}$$

implies

$$A \text{ strict } C$$

for all sets A, B, C.

Proof. (of Lemma 7.1.6)

Suppose 7.1.2 holds, and assume for the sake of contradiction that NOT(A strict C), which means that A surj C. Now since B strict C, Theorem 7.1.5 lets us conclude that C surj B. So we have

$$A \text{ surj } C \text{ AND } C \text{ surj } B.$$

and Lemma 7.1.3.2 lets us conclude that A surj B, contradicting the fact that A strict B. ■

We're omitting a proof of Theorem 7.1.5 because proving it involves technical set theory—typically the theory of ordinals again—that we're not going to get into. But since proving Lemma 7.1.6 is the only use we'll make of Theorem 7.1.5, we hope you won't feel cheated not to see a proof.

Infinity is different

A basic property of finite sets that does *not* carry over to infinite sets is that adding something new makes a set bigger. That is, if A is a finite set and $b \notin A$, then $|A \cup \{b\}| = |A| + 1$, and so A and $A \cup \{b\}$ are not the same size. But if A is infinite, then these two sets *are* the same size!

Lemma 7.1.7. *Let A be a set and $b \notin A$. Then A is infinite iff A bij $A \cup \{b\}$.*

Proof. Since A is *not* the same size as $A \cup \{b\}$ when A is finite, we only have to show that $A \cup \{b\}$ is the same size as A when A is infinite.

That is, we have to find a bijection between $A \cup \{b\}$ and A when A is infinite. Here's how: since A is infinite, it certainly has at least one element; call it a_0. But since A is infinite, it has at least two elements, and one of them must not equal to a_0; call this new element a_1. But since A is infinite, it has at least three elements, one of which must not equal both a_0 and a_1; call this new element

a_2. Continuing in this way, we conclude that there is an infinite sequence $a_0, a_1, a_2, \ldots, a_n, \ldots$ of different elements of A. Now it's easy to define a bijection $e : A \cup \{b\} \to A$:

Illegal pream-token (e)

Countable Sets

A set, C, is countable iff its elements can be listed in order, that is, the elements in C are precisely the elements in the sequence

$$c_0, c_1, \ldots, c_n, \ldots$$

Assuming no repeats in the list, saying that C can be listed in this way is formally the same as saying that the function, $f : \mathbb{N} \to C$ defined by the rule that $f(i) :: c_i$, is a bijection.

Definition 7.1.8

A set, C, is *countably infinite* iff $\mathbb{N}$ **bij** C. A set is *countable* iff it is finite or countably infinite.

We can also make an infinite list using just a finite set of elements if we allow repeats. For example, we can list the elements in the three-element set $2, 4, 6$ as

$$2, 4, 6, 6, 6, \ldots$$

This simple observation leads to an alternative characterization of countable sets that does not make separate cases of finite and infinite sets. Namely, a set C is countable iff there is a list

$$c_0, c_1, \ldots, c_n, \ldots$$

of the elements of C, possibly with repeats.

Lemma 7.1.9. *A set, C, is countable iff* $\mathbb{N}$ surj C. *In fact, a nonempty set C is countable iff there is a* total *surjective function $g : \mathbb{N} \to C$.*

The proof is left to Problem 7.12.

The most fundamental countably infinite set is the set, $\mathbb{N}$, itself. But the set, $\mathbb{Z}$, of *all* integers is also countably infinite, because the integers can be listed in the order:

$$0, -1, 1, -2, 2, -3, 3, \ldots \tag{7.1.3}$$

In this case, there is a simple formula for the nth element of the list (7.1.3). That is, the bijection $f : \mathbb{N} \to C$ such that $f(n)$ is the nth element of the list can be defined as:

$$f(n) ::= \begin{cases} n/2 & \text{if } n \text{ is even,} \\ -(n+1)/2 & \text{if } n \text{ is odd.} \end{cases}$$

There is also a simple way to list all *pairs* of nonnegative integers, which shows that $(\mathbb{N} \times \mathbb{N})$ is also countably infinite (Problem 7.16). From this, it's a small step to reach the conclusion that the set, $\mathbb{Q}^{\geq 0}$, of nonnegative rational numbers is countable. This may be a surprise—after all, the rationals densely fill up the space between integers, and for any two, there's another in between. So it might seem as though you couldn't write out all the rationals in a list, but Problem 7.10 illustrates how to do it. More generally, it is easy to show that countable sets are closed under unions and products (Problems 7.1 and 7.16) which implies the countability of a bunch of familiar sets:

Corollary 7.1.10. *The following sets are countably infinite:*

$$\mathbb{Z}^+, \mathbb{Z}, \mathbb{N} \times \mathbb{N}, \mathbb{Q}^+, \mathbb{Z} \times \mathbb{Z}, \mathbb{Q}.$$

A small modification of the proof of Lemma 7.1.7 shows that countably infinite sets are the "smallest" infinite sets, or more precisely that if A is an infinite set, and B is countable, then A **surj** B (see Problem 7.9).

Also, since adding one new element to an infinite set doesn't change its size, you can add any *finite* number of elements without changing the size by simply adding one element after another. Something even stronger is true: you can add a *countably* infinite

number of new elements to an infinite set and still wind up with just a set of the same size (Problem 7.13).

By the way, it's a common mistake to think that, because you can add any finite number of elements to an infinite set and have a bijection with the original set, that you can also throw in infinitely many new elements. In general it isn't true that just because it's OK to do something any finite number of times, it also OK to do it an infinite number of times. For example, starting from 3, you can increment by 1 any finite number of times, and the result will be some integer greater than or equal to 3. But if you increment an infinite number of times, you don't get an integer at all.

Power sets are strictly bigger

Cantor's astonishing discovery was that *not all infinite sets are the same size*. In particular, he proved that for any set, A, the power set, pow(A), is "strictly bigger" than A. That is,

Theorem 7.1.11

[Cantor] For any set, A

$$A \text{ strict } \text{pow}(A).$$

Proof. To show that A is strictly smaller than pow(A), we have to show that if g is a function from A to pow(A), then g is *not* a surjection. To do this, we'll simply find a subset, $A_g \subseteq A$ that is not in the range of g. The idea is, for any element $a \in A$, to look at the set $g(a) \subseteq A$ and ask whether or not a happens to be in $g(a)$. First, define

$$A_g ::= \{a \in A \mid a \notin g(a)\}.$$

A_g is now a well-defined subset of A, which means it is a member of pow(A). But A_g can't be in the range of g, because if it were, we would have

$$A_g = g(a_0)$$

for some $a_0 \in A$, so by definition of A_g,

$$a \in g(a_0) \text{ iff } a \in A_g \text{ iff } a \notin g(a)$$

for all $a \in A$. Now letting $a = a_0$ yields the contradiction

$$a_0 \in g(a_0) \text{ iff } a_0 \notin g(a_0).$$

So g is not a surjection, because there is an element in the power set of A, specifically the set A_g, that is not in the range of g. ■

Cantor's Theorem immediately implies:

Corollary 7.1.12. pow($\mathbb{N}$) is *uncountable*.

The bijection between subsets of an n-element set and the length n bit-strings, $\{0,1\}^n$, used to prove Theorem 4.5.5, carries over to a bijection between subsets of a countably infinite set and the infinite bit-strings, $\{0,1\}^\omega$. That is,

$$\text{pow}(\mathbb{N}) \text{ bij } \{0,1\}^\omega.$$

This immediately implies

Corollary 7.1.13. $\{0,1\}^\omega$ *is uncountable*.

More Countable and Uncountable Sets

Once we have a few sets we know are countable or uncountable, we can get lots more examples using Lemma 7.1.3. In particular, we can appeal to the following immediate corollary of the Lemma:

Corollary 7.1.14.

(a) *If U is an uncountable set and A surj U, then A is uncountable.*

(b) *If C is a countable set and C surj A, then A is countable.*

For example, now that we know that the set $\{0,1\}^\omega$ of infinite bit strings is uncountable, it's a small step to conclude that

Corollary 7.1.15. *The set $\mathbb{R}$ of real numbers is uncountable.*

To prove this, think about the infinite decimal expansion of a real number:

$$\begin{aligned} \sqrt{2} &= 1.4142\ldots, \\ 5 &= 5.000\ldots, \\ 1/10 &= 0.1000\ldots, \\ 1/3 &= 0.333\ldots, \\ 1/9 &= 0.111\ldots, \\ 4\frac{1}{99} &= 4.010101\ldots \end{aligned} \tag{7.1.4}$$

Let's map any real number r to the infinite bit string $b(r)$ equal to the sequence of bits in the decimal expansion of r, starting at the decimal point. If the decimal expansion of r happens to contain a digit other than 0 or 1, leave $b(r)$ undefined. For example,

$$\begin{aligned} b(5) &= 000\ldots, \\ b(1/10) &= 1000\ldots, \\ b(1/9) &= 111\ldots, \\ b(4\frac{1}{99}) &= 010101\ldots \\ b(\sqrt{2}), b(1/3) &\text{ are undefined.} \end{aligned} \tag{7.1.5}$$

Now b is a function from real numbers to infinite bit strings.[1] It is not a total function, but it clearly is a surjection. This shows that

$$\mathbb{R} \text{ surj } \{0,1\}^{\omega}.$$

and the uncountability of the reals now follows by Corollary 7.1.14.(a).

For another example, let's prove

Corollary 7.1.16. *The set $(\mathbb{Z}^+)^*$ of all finite sequences of positive integers is countable.*

To prove this, think about the prime factorization of a nonnegative integer:

$$\begin{aligned} 20 &= 2^2 \cdot 3^0 \cdot 5^1 \cdot 7^0 \cdot 11^0 \cdot 13^0 \ldots, \\ 6615 &= 2^0 \cdot 3^3 \cdot 5^1 \cdot 7^2 \cdot 11^0 \cdot 13^0 \ldots. \end{aligned} \tag{7.1.6}$$

Let's map any nonnegative integer n to the finite sequence $e(n)$ of nonzero exponents in its prime factorization. For example,

$$\begin{aligned} e(20) &= (2,1), \\ e(6615) &= (3,1,2), \\ e(5^{13} \cdot 11^9 \cdot 47^{817} \cdot 103^{44}) &= (13,9,817,44), \\ e(1) &= \lambda, \qquad \text{(the empty string)} \\ e(0) &\text{ is undefined.} \end{aligned} \tag{7.1.7}$$

Now e is a function from $\mathbb{N}$ to $(mathbbZ^+)^*$. It is defined on all positive integers, and it clearly is a surjection. This shows that

$$\mathbb{N} \text{ surj } (\mathbb{Z}^+)^*.$$

and the countability of the finite strings of positive integers now follows by Corollary 7.1.14.(b).

Larger Infinities

There are lots of different sizes of infinite sets. For example, starting with the infinite set, $\mathbb{N}$, of nonnegative integers, we can build the infinite sequence of sets

$$\mathbb{N} \text{ strict pow}(\mathbb{N}) \text{ strict pow}(\text{pow}(\mathbb{N})) \text{ strict pow}(\text{pow}(\text{pow}(\mathbb{N}))) \text{ strict } \ldots$$

By Cantor's Theorem 7.1.11, each of these sets is strictly bigger than all the preceding ones. But that's not all: the union of all the sets in the sequence is strictly bigger than each set in the sequence (see Problem 7.23). In this way you can keep going indefinitely, building "bigger" infinities all the way.

Diagonal Argument

Theorem 7.1.11 and similar proofs are collectively known as "diagonal arguments" because of a more intuitive version of the proof described in terms of on an infinite square array. Namely, suppose there was a bijection between $\mathbb{N}$ and $\{0,1\}^\omega$. If such a relation existed, we would be able to display it as a list of the infinite bit strings in some countable order or another. Once we'd found a viable way to organize this list, any given string in $\{0,1\}^\omega$ would appear in a finite number of steps, just as any integer you can name will show up a finite number of steps from 0. This hypothetical list would look something like the one below, extending to infinity both vertically and horizontally:

$$\begin{array}{lcccccccc} A_0 & = & 1 & 0 & 0 & 0 & 1 & 1 & \cdots \\ A_1 & = & 0 & 1 & 1 & 1 & 0 & 1 & \cdots \\ A_2 & = & 1 & 1 & 1 & 1 & 1 & 1 & \cdots \\ A_3 & = & 0 & 1 & 0 & 0 & 1 & 0 & \cdots \\ A_4 & = & 0 & 0 & 1 & 0 & 0 & 0 & \cdots \\ A_5 & = & 1 & 0 & 0 & 1 & 1 & 1 & \cdots \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

But now we can exhibit a sequence that's missing from our allegedly complete list of all the sequences. Look at the diagonal in our sample list:

$$\begin{array}{lcccccccc} A_0 & = & \mathbf{1} & 0 & 0 & 0 & 1 & 1 & \cdots \\ A_1 & = & 0 & \mathbf{1} & 1 & 1 & 0 & 1 & \cdots \\ A_2 & = & 1 & 1 & \mathbf{1} & 1 & 1 & 1 & \cdots \\ A_3 & = & 0 & 1 & 0 & \mathbf{0} & 1 & 0 & \cdots \\ A_4 & = & 0 & 0 & 1 & 0 & \mathbf{0} & 0 & \cdots \\ A_5 & = & 1 & 0 & 0 & 1 & 1 & \mathbf{1} & \cdots \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

Here is why the diagonal argument has its name: we can form a sequence D consisting of the bits on the diagonal.

$$D = 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \ldots,$$

Then, we can form another sequence by switching the 1's and 0's along the diagonal. Call this sequence C:

$$C = 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \ldots.$$

Now if nth term of A_n is 1 then the nth term of C is 0, and *vice versa*, which guarantees that C differs from A_n. In other words, C has at least one bit different from *every* sequence on our list. So C is an element of $\{0,1\}^\omega$ that does not appear in our list—our list can't be complete!

This diagonal sequence C corresponds to the set $\{a \in A \mid a \notin g(a)\}$ in the proof of Theorem 7.1.11. Both are defined in terms of a countable subset of the uncountable infinity in a way that excludes them from that subset, thereby proving that no countable subset can be as big as the uncountable set.

[1]Some rational numbers can be expanded in two ways—as an infinite sequence ending in all 0's or as an infinite sequence ending in all 9's. For example,

$$\begin{aligned} 5 &= 5.000\ldots = 4.999\ldots, \\ \frac{1}{10} &= 0.1000\ldots = 0.0999\ldots. \end{aligned} \tag{7.1.8}$$

In such cases, define $b(r)$ to be the sequence that ends with all 0's.

This page titled 7.1: Infinite Cardinality is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

7.2: The Halting Problem

Although towers of larger and larger infinite sets are at best a romantic concern for a computer scientist, the *reasoning* that leads to these conclusions plays a critical role in the theory of computation. Diagonal arguments are used to show that lots of problems can't be solved by computation, and there is no getting around it.

This story begins with a reminder that having procedures operate on programs is a basic part of computer science technology. For example, *compilation* refers to taking any given program text written in some "high level" programming language like Java, C++, Python, . . . , and then generating a program of low-level instructions that does the same thing but is targeted to run well on available hardware. Similarly, *interpreters* or *virtual machines* are procedures that take a program text designed to be run on one kind of computer and simulate it on another kind of computer. Routine features of compilers involve "type-checking" programs to ensure that certain kinds of run-time errors won't happen, and "optimizing" the generated programs so they run faster or use less memory.

The fundamental thing that just can't be done by computation is a *perfect* job of type-checking, optimizing, or any kind of analysis of the overall run time behavior of programs. In this section, we'll illustrate this with a basic example known as the *Halting Problem*. The general Halting Problem for some programming language is, given an arbitrary program, to determine whether the program will run forever if it is not interrupted. If the program does not run forever, it is said to halt. Real programs may halt in many ways, for example, by returning some final value, aborting with some kind of error, or by awaiting user input. But it's easy to detect when any given program will halt: just run it on a virtual machine and wait till it stops. The problem comes when the given program does *not* halt—you may wind up waiting indefinitely without realizing that the wait is fruitless. So how could you detect that the program does *not* halt? We will use a diagonal argument to prove that if an analysis program tries to recognize the non-halting programs, it is bound to give wrong answers, or no answers, for an infinite number of the programs it is supposed to be able to analyze!

To be precise about this, let's call a programming procedure—written in your favorite programming language—a *string procedure* when it is applicable to strings over a standard alphabet—say, the 256 character ASCII alphabet. As a simple example, you might think about how to write a string procedure that halts precisely when it is applied to a *double letter* ASCII string, namely, a string in which every character occurs twice in a row. For example, **aaCC33**, and **zz++ccBB** are double letter strings, but **aa;bb**, **b33**, and **AAAAA** are not.

We'll call a set of strings *recognizable* if there is a string procedure that halts when it is applied to any string in that set and does not halt when applied to any string not in the set. For example, we've just agreed that the set of double letter strings is recognizable.

Let ASCII^* be the set of (finite) strings of ASCII characters. There is no harm in assuming that every program can be written using only the ASCII characters; they usually are. When a string $s \in \text{ASCII}^*$ is actually the ASCII description of some string procedure, we'll refer to that string procedure as P_s. You can think of P_s as the result of compiling s. [2] It's technically helpful to treat *every* ASCII string as a program for a string procedure. So when a string $s \in \text{ASCII}^*$ doesn't parse as a proper string procedure, we'll define P_s to be some default string procedure—say one that never halts on any input.

Focusing just on string procedures, the general Halting Problem is to decide, given strings s and t, whether or not the procedure P_s halts when applied to t. We'll show that the general problem can't be solved by showing that a special case can't be solved, namely, whether or not P_s applied to s halts. So, let's define

Definition 7.2.1

$$\text{No-halt} ::= \{s \in \text{ASCII}^* \mid P_s \text{ applied to } s \text{ does not halt}\}. \tag{7.2.1}$$

We're going to prove

Theorem 7.2.2

No-halt is not recognizable.

We'll use an argument just like Cantor's in the proof of Theorem 7.1.11.

Proof

For any string $s \in \text{ASCII}^*$, let $f(s)$ be the set of strings recognized by P_s:

$$f(s) ::= \{t \in \text{ASCII}^* \mid P_s \text{ halts when applied to } t\}.$$

By convention, we associated a string procedure, P_s, with every string, $s \in \text{ASCII}^*$, which makes f a total function, and by definition,

$$s \in \text{No-halt IFF } s \notin f(s), \tag{7.2.2}$$

for all strings, $s \in \text{ASCII}^*$.

Now suppose to the contrary that No-halt was recognizable. This means there is some procedure P_{s0} that recognizes No-halt, which is the same as saying that

$$\text{No-halt} = f(s_0).$$

Combined with (7.2.2), we get

$$s \in f(s_0) \text{ iff } s \notin f(s) \tag{7.2.3}$$

for all $s \in \text{ASCII}^*$. Now letting $s = s_0$ in (7.2.3) yields the immediate contradiction

$$s_0 \in f(s_0) \text{ iff } s_0 \notin f(s_0)$$

This contradiction implies that No-halt cannot be recognized by any string procedure. ■

So that does it: it's logically impossible for programs in any particular language to solve just this special case of the general Halting Problem for programs in that language. And having proved that it's impossible to have a procedure that figures out whether an arbitrary program halts, it's easy to show that it's impossible to have a procedure that is a perfect recognizer for *any* overall run time property.[3]

For example, most compilers do "static" type-checking at compile time to ensure that programs won't make run-time type errors. A program that type-checks is guaranteed not to cause a run-time type-error. But since it's impossible to recognize perfectly when programs won't cause type-errors, it follows that the type-checker must be rejecting programs that really wouldn't cause a type-error. The conclusion is that no type-checker is perfect—you can always do better!

It's a different story if we think about the *practical* possibility of writing programming analyzers. The fact that it's logically impossible to analyze perfectly arbitrary programs does not mean that you can't do a very good job analyzing interesting programs that come up in practice. In fact, these "interesting" programs are commonly *intended* to be analyzable in order to confirm that they do what they're supposed to do.

In the end, it's not clear how much of a hurdle this theoretical limitation implies in practice. But the theory does provide some perspective on claims about general analysis methods for programs. The theory tells us that people who make such claims either

- are exaggerating the power (if any) of their methods, perhaps to make a sale or get a grant, or
- are trying to keep things simple by not going into technical limitations they're aware of, or
- perhaps most commonly, are so excited about some useful practical successes of their methods that they haven't bothered to think about the limitations which must be there.

So from now on, if you hear people making claims about having general program analysis/verification/optimization methods, you'll know they can't be telling the whole story.

One more important point: there's no hope of getting around this by switching programming languages. Our proof covered programs written in some given programming language like Java, for example, and concluded that no Java program can perfectly analyze all Java programs. Could there be a C++ analysis procedure that successfully takes on all Java programs? After all, C++ does allow more intimate manipulation of computer memory than Java does. But there is no loophole here: it's possible to write a virtual machine for C++ in Java, so if there were a C++ procedure that analyzed Java programs, the Java virtual machine would be able to do it too, and that's impossible. These logical limitations on the power of computation apply no matter what kinds of programs or computers you use.

[2]The string, $s \in \text{ASCII}^*$, and the procedure, P_s, have to be distinguished to avoid a type error: you can't apply a string to string. For example, let s be the string that you wrote as your program to recognize the double letter strings. Applying s to a string argument, say aabbccdd, should throw a type exception; what you need to do is compile s to the procedure P_s and then apply P_s to aabbccdd.

[3]The weasel word "overall" creeps in here to rule out some run time properties that are easy to recognize because they depend only on part of the run time behavior. For example, the set of programs that halt after executing at most 100 instructions is recognizable.

This page titled 7.2: The Halting Problem is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

7.3: The Logic of Sets

Russell's Paradox

Reasoning naively about sets turns out to be risky. In fact, one of the earliest attempts to come up with precise axioms for sets in the late nineteenth century by the logician Gotlob Frege, was shot down by a three line argument known as *Russell's Paradox*[4] which reasons in nearly the same way as the proof of Cantor's Theorem 7.1.11. This was an astonishing blow to efforts to provide an axiomatic foundation for mathematics:

Russell's Paradox

Let S be a variable ranging over all sets, and define

$$W ::= \{S \mid S \notin S\}.$$

So by definition,

$$S \in W \text{ iff } S \notin S,$$

for every set S. In particular, we can let S be W, and obtain the contradictory result that

$$W \in W \text{ iff } W \notin W.$$

The simplest reasoning about sets crashes mathematics! Russell and his colleague Whitehead spent years trying to develop a set theory that was not contradictory, but would still do the job of serving as a solid logical foundation for all of mathematics.

Actually, a way out of the paradox was clear to Russell and others at the time: *it's unjustified to assume that W is a set.* The step in the proof where we let S be W has no justification, because S ranges over sets, and W might not be a set. In fact, the paradox implies that W had better not be a set!

But denying that W is a set means we must *reject* the very natural axiom that every mathematically well-defined collection of sets is actually a set. The problem faced by Frege, Russell and their fellow logicians was how to specify *which* well-defined collections are sets. Russell and his Cambridge University colleague Whitehead immediately went to work on this problem. They spent a dozen years developing a huge new axiom system in an even huger monograph called *Principia Mathematica*, but for all intents and purposes, their approach failed. It was so cumbersome no one ever used it, and it was subsumed by a much simpler, and now widely accepted, axiomatization of set theory by the logicians Zermelo and Fraenkel.

The ZFC Axioms for Sets

A *formula of set theory*[5] is a predicate formula that only uses the predicates "$x = y$" and "$x \in y$." The domain of discourse is the collection of sets, and "$x \in y$" is interpreted to mean that x and y are variables that range over sets, and x is one of the elements in y.

It's generally agreed that, using some simple logical deduction rules, essentially all of mathematics can be derived from some formulas of set theory called the Axioms of *Zermelo-Fraenkel Set Theory* with Choice (ZFC).

For example, since x is a subset of y iff every element of x is also an element of y, here's how we can express x being a subset of y with a formula of set theory:

$$(x \subseteq y) ::= \forall z.\, (z \in x \text{ IMPLIES } z \in y). \tag{7.3.1}$$

Now we can express formulas of set theory using "$x \subseteq y$" as an abbreviation for formula (7.3.1).

We're *not* going to be studying the axioms of ZFC in this text, but we thought you might like to see them—and while you're at it, get some practice reading quantified formulas:

Extensionality. Two sets are equal if they have the same members.

$$(\forall z.\, z \in x \text{ IFF } z \in y) \text{ IMPLIES } x = y.$$

Pairing. For any two sets x and y, there is a set, $\{x, y\}$, with x and y as its only elements:

$$\forall x, y.\, \exists u.\, \forall z.\, [z \in u \text{ iff } (z = x \text{ OR } z = y)]$$

Union. The union, u, of a collection, z, of sets is also a set:

$$\forall z.\, \exists u.\, \forall x.\, (\exists y.\, x \in y \text{ AND } y \in z) \text{ IFF } x \in u.$$

Infinity. There is an infinite set. Specifically, there is a nonempty set, x, such that for any set $y \in x$, the set $\{y\}$ is also a member of x.

Subset. Given any set, x, and any definable property of sets, there is a set containing precisely those elements $y \in x$ that have the property.

$$\forall x.\, \exists z.\, \forall y.\, y \in z \text{ IFF } [y \in x \text{ AND } \phi(y)]$$

where $\phi(y)$ is any assertion about y definable in the notation of set theory.

Power Set. All the subsets of a set form another set:

$$\forall x.\, \exists p.\, \forall u.\, u \subseteq x \text{ IFF } u \in p.$$

Replacement. Suppose a formula, ϕ, of set theory defines the graph of a function, that is,

$$\forall x, y, z.\, [\phi(x, y) \text{ IFF } \phi(x, z)] \text{ IMPLIES } y = z.$$

Then the image of any set, s, under that function is also a set, t. Namely,

$$\forall s\ \exists t\ \forall y.\, [\exists x.\, \phi(x, y) \text{ IFF } y \in t].$$

Foundation. There cannot be an infinite sequence

$$\cdots \in x_n \in \cdots x_1 \in x_0$$

of sets each of which is a member of the previous one. This is equivalent to saying every nonempty set has a "member-minimal" element. Namely, define

$$\text{member-minimal}(m, x) ::= [m \in x \text{ AND } \forall y \in x.\, y \notin m].$$

Then the foundation axiom is

$$\forall x.\, x \neq \emptyset \text{ IMPLIES } \exists m.\, \text{member-minimal}(m, x).$$

Choice. Given a set, s, whose members are nonempty sets no two of which have any element in common, then there is a set, c, consisting of exactly one element from each set in s. The formula is given in Problem 7.28.

Avoiding Russell's Paradox

These modern ZFC axioms for set theory are much simpler than the system Russell and Whitehead first came up with to avoid paradox. In fact, the ZFC axioms are as simple and intuitive as Frege's original axioms, with one technical addition: the Foundation axiom. Foundation captures the intuitive idea that sets must be built up from "simpler" sets in certain standard ways. And in particular, Foundation implies that no set is ever a member of itself. So the modern resolution of Russell's paradox goes as follows: since $S \notin S$ for all sets S, it follows that W, defined above, contains every set. This means W can't be a set—or it would be a member of itself.

[4]Bertrand Russell was a mathematician/logician at Cambridge University at the turn of the Twentieth Century. He reported that when he felt too old to do mathematics, he began to study and write about philosophy, and when he was no longer smart enough to do philosophy, he began writing about politics. He was jailed as a conscientious objector during World War I. For his extensive philosophical and political writing, he won a Nobel Prize for Literature.

[5]Technically this is called a first-order predicate formula of set theory

This page titled 7.3: The Logic of Sets is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

7.4: Does All This Really Work?

So this is where mainstream mathematics stands today: there is a handful of ZFC axioms from which virtually everything else in mathematics can be logically derived. This sounds like a rosy situation, but there are several dark clouds, suggesting that the essence of truth in mathematics is not completely resolved.

- The ZFC axioms weren't etched in stone by God. Instead, they were mostly made up by Zermelo, who may have been a brilliant logician, but was also a fallible human being—probably some days he forgot his house keys. So maybe Zermelo, just like Frege, didn't get his axioms right and will be shot down by some successor to Russell who will use his axioms to prove a proposition P and its negation $\overline{P}$. Then math as we understand it would be broken—this may sound crazy, but it has happened before.

  In fact, while there is broad agreement that the ZFC axioms are capable of proving all of standard mathematics, the axioms have some further consequences that sound paradoxical. For example, the Banach-Tarski Theorem says that, as a consequence of the *axiom of choice*, a solid ball can be divided into six pieces and then the pieces can be rigidly rearranged to give *two* solid balls of the same size as the original!

- Some basic questions about the nature of sets remain unresolved. For example, Cantor raised the question whether there is a set whose size is strictly between the smallest infinite set, $\mathbb{N}$ (see Problem 7.9), and the strictly larger set, pow($\mathbb{N}$)? Cantor guessed not:

  Cantor's *Contiuum Hypothesis*: There is no set, A, such that

  $$\mathbb{N} \text{ strict } A \text{ strict } \text{pow}(\mathbb{N}).$$

  The Continuum Hypothesis remains an open problem a century later. Its difficulty arises from one of the deepest results in modern Set Theory— discovered in part by Gödel in the 1930's and Paul Cohen in the 1960's— namely, the ZFC axioms are not sufficient to settle the Continuum Hypothesis: there are two collections of sets, each obeying the laws of ZFC, and in one collection the Continuum Hypothesis is true, and in the other it is false. Until a mathematician with a deep understanding of sets can extend ZFC with persuasive new axioms, the Continuum Hypothesis will remain undecided.

- But even if we use more or different axioms about sets, there are some unavoidable problems. In the 1930's, Godel ¨ proved that, assuming that an axiom system like ZFC is consistent—meaning you can't prove both P and $\overline{P}$ for any proposition, P—then the very proposition that the system is consistent (which is not too hard to express as a logical formula) cannot be proved in the system. In other words, no consistent system is strong enough to verify itself.

Large Infinities in Computer Science

If the romance of different-size infinities and continuum hypotheses doesn't appeal to you, not knowing about them is not going to limit you as a computer scientist. These abstract issues about infinite sets rarely come up in mainstream mathematics, and they don't come up at all in computer science, where the focus is generally on "countable," and often just finite, sets. In practice, only logicians and set theorists have to worry about collections that are "too big" to be sets. That's part of the reason that the 19th century mathematical community made jokes about "Cantor's paradise" of obscure infinities. But the challenge of reasoning correctly about this far-out stuff led directly to the profound discoveries about the logical limits of computation described in Section 7.2, and that really is something every computer scientist should understand.

This page titled 7.4: Does All This Really Work? is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

SECTION OVERVIEW

2: Structures

8: Number Theory

8.1: Divisibility
8.2: The Greatest Common Divisor
8.3: Prime Mysteries
8.4: The Fundamental Theorem of Arithmetic
8.5: Alan Turing
8.6: Modular Arithmetic
8.7: Remainder Arithmetic
8.8: Turing’s Code (Version 2.0)
8.9: Multiplicative Inverses and Cancelling
8.10: Euler's Theorem
8.11: RSA Public Key Encryption
8.12: What has SAT got to do with it?

9: Directed graphs and Partial Orders

9.1: Vertex Degrees
9.2: Walks and Paths
9.3: Adjacency Matrices
9.4: Walk Relations
9.5: Directed Acyclic Graphs and Scheduling
9.6: Partial Orders
9.7: Representing Partial Orders by Set Containment
9.8: Linear Orders
9.9: Product Orders
9.10: Equivalence Relations
9.11: Summary of Relational Properties

10: Communication Networks

10.1: Complete Binary Tree
10.2: Routing Problems
10.3: Network Diameter
10.4: Switch Count
10.5: Network Latency
10.6: Congestion
10.7: 2-D Array
10.8: Butterfly
10.9: Beneš Network

11: Simple Graphs

11.1: Vertex Adjacency and Degrees
11.2: Sexual Demographics in America
11.3: Some Common Graphs
11.4: Isomorphism
11.5: Bipartite Graphs and Matchings
11.6: The Stable Marriage Problem
11.7: Coloring
11.8: Simple Walks
11.9: Connectivity
11.10: Forests and Trees

12: Planar Graphs

12.1: Drawing Graphs in the Plane
12.2: Definitions of Planar Graphs
12.3: Euler's Formula
12.4: Bounding the Number of Edges in a Planar Graph
12.5: Returning to K5 and K3,3
12.6: Coloring Planar Graphs
12.7: Classifying Polyhedra
12.8: Another Characterization for Planar Graphs

This page titled 2: Structures is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

8: Number Theory

Number theory is the study of the integers. *Why* anyone would want to study the integers may not be obvious. First of all, what's to know? There's 0, there's 1, 2, 3, and so on, and, oh yeah, -1, -2, ... Which one don't you understand? What practical value is there in it?

The mathematician G. H. Hardy delighted at its impracticality. He wrote:

> [Number theorists] may be justified in rejoicing that there is one science, at any rate, and that their own, whose very remoteness from ordinary human activities should keep it gentle and clean.

Hardy was especially concerned that number theory not be used in warfare; he was a pacifist. You may applaud his sentiments, but he got it wrong: number theory underlies modern cryptography, which is what makes secure online communication possible. Secure communication is of course crucial in war—leaving poor Hardy spinning in his grave. It's also central to online commerce. Every time you buy a book from Amazon, use a certificate to access a web page, or use a PayPal account, you are relying on number theoretic algorithms.

Number theory also provides an excellent environment for us to practice and apply the proof techniques that we developed in previous chapters. We'll work out properties of greatest common divisors (gcd's) and use them to prove that integers factor uniquely into primes. Then we'll introduce modular arithmetic and work out enough of its properties to explain the RSA public key crypto-system.

Since we'll be focusing on properties of the integers, we'll adopt the default convention in this chapter that *variables range over the set, $\mathbb{Z}$, of integers*.

8.1: Divisibility
8.2: The Greatest Common Divisor
8.3: Prime Mysteries
8.4: The Fundamental Theorem of Arithmetic
8.5: Alan Turing
8.6: Modular Arithmetic
8.7: Remainder Arithmetic
8.8: Turing's Code (Version 2.0)
8.9: Multiplicative Inverses and Cancelling
8.10: Euler's Theorem
8.11: RSA Public Key Encryption
8.12: What has SAT got to do with it?

This page titled 8: Number Theory is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.1: Divisibility

The nature of number theory emerges as soon as we consider the *divides* relation.

Definition 8.1.1

a *divides* b (notation $a \mid b$) iff there is an integer k such that

$$ak = b.$$

The divides relation comes up so frequently that multiple synonyms for it are used all the time. The following phrases all say the same thing:

- $a \mid b$,
- a divides b,
- a is a *divisor* of b,
- a is a *factor* of b,
- b is *divisible* by a,
- b is a *multiple* of a.

Some immediate consequences of Definition 8.1.1 are that for all n

$$n \mid 0, \quad n \mid n, \text{ and } \quad \pm 1 \mid n.$$

Also,

$$0 \mid n \text{ IMPLIES } n = 0.$$

Dividing seems simple enough, but let's play with this definition. The Pythagoreans, an ancient sect of mathematical mystics, said that a number is *perfect* if it equals the sum of its positive integral divisors, excluding itself. For example, $6 = 1 + 2 + 3$ and $28 = 1 + 2 + 4 + 7 + 14$ are perfect numbers. On the other hand, 10 is not perfect because $1 + 2 + 5 = 8$, and 12 is not perfect because $1 + 2 + 3 + 4 + 6 = 16$. Euclid characterized all the *even* perfect numbers around 300 BC (Problem 8.2). But is there an *odd* perfect number? More than two thousand years later, we still don't know! All numbers up to about 10^{300} have been ruled out, but no one has proved that there isn't an odd perfect number waiting just over the horizon.

So a half-page into number theory, we've strayed past the outer limits of human knowledge. This is pretty typical; number theory is full of questions that are easy to pose, but incredibly difficult to answer. We'll mention a few more such questions in later sections.[1]

Facts about Divisibility

The following lemma collects some basic facts about divisibility

Lemma 8.1.2.

1. *If* $a \mid b$ *and* $b \mid c$, *then* $a \mid c$.
2. *If* $a \mid b$ *and* $a \mid c$, *then* $a \mid sb + tc$ *for all* s *and* t.
3. *For all* $c \neq 0$, $a \mid b$ *if and only if* $ca \mid cb$.

Proof. These facts all follow directly from Definition 8.1.1. To illustrate this, we'll prove just part 2:

Given that $a \mid b$, there is some $k_1 \in \mathbb{Z}$ such that $ak_1 = b$. Likewise, $ak_2 = c$, so

$$sb \pm tc = s(k_1 a) + t(k_2 a) = (sk_1 + tk_2)a.$$

Therefore $sb + tc = k_3 a$ where $k_3 ::= (sk_1 + tk_2)$, which means that

$$a \mid sb + tc. \quad \blacksquare$$

A number of the form $sb + tc$ is called an *integer linear combination* of b and c, or, since in this chapter we're only talking about integers, just a *linear combination*. So Lemma 8.1.2.2 can be rephrased as

If a divides b and c, then a divides every linear combination of b and c.

We'll be making good use of linear combinations, so let's get the general definition on record:

Definition 8.1.3

An integer n is a *linear combination* of numbers $b_0, \ldots, b_k$ iff

$$n = s_0 b_0 + s_1 b_1 + \cdots + s_k b_k$$

for some integers $s_0, \ldots, s_k$.

When Divisibility Goes Bad

As you learned in elementary school, if one number does *not* evenly divide another, you get a "quotient" and a "remainder" left over. More precisely:

Theorem 8.1.4

[Division Theorem][2] Let n and d be integers such that $d > 0$. Then there exists a unique pair of integers q and r, such that

$$n = q \cdot r \text{ AND } 0 \leq r < d \tag{8.1.1}$$

The number q is called the *quotient* and the number r is called the *remainder* of n divided by d. We use the notation $\mathbf{qcnt}(n, d)$ for the quotient and $\mathbf{rem}(n, d)$ for the remainder. For example, $\mathbf{qcnt}(2716, 10) = 271$ and $\mathbf{rem}(2716, 10) = 6$, since $2716 = 271 \cdot 10 + 6$. Similarly, $\mathbf{rem}(-11, 7) = 3$, since $-11 = (-2) \cdot 7 + 3$.

There is a remainder operator built into many programming languages. For example, "32 % 5" will be familiar as remainder notation to programmers in Java, C, and C++; it evaluates to $\mathbf{rem}(32, 5) = 2$ in all three languages. On the other hand, these and other languages treat remainders involving negative numbers inconsistently, so don't be distracted by your programming language's behavior, and remember to stick to the definition according to the Division Theorem 8.1.4.

The remainder on division by n is a number in the (integer) *interval* from 0 to $n - 1$. Such intervals come up so often that it is useful to have a simple notation for them.

$$\begin{aligned}
(k\,..\,n) &::= \{i \mid k < i < n\},\\
(k\,..\,n] &::= (k, n) \cup \{n\},\\
[k\,..\,n) &::= \{k\} \cup (k, n),\\
[k\,..\,n] &::= \{k\} \cup (k, n) \cup \{n\} = \{i \mid k \leq i \leq n\}.
\end{aligned}$$

Die Hard

Die Hard 3 is just a B-grade action movie, but we think it has an inner message: everyone should learn at least a little number theory. In Section 5.4.4, we formalized a state machine for the Die Hard jug-filling problem using 3 and 5 gallon jugs, and also with 3 and 9 gallon jugs, and came to different conclusions about bomb explosions. What's going on in general? For example, how about getting 4 gallons from 12- and 18-gallon jugs, getting 32 gallons with 899- and 1147-gallon jugs, or getting 3 gallons into a jug using just 21- and 26-gallon jugs?

It would be nice if we could solve all these silly water jug questions at once. This is where number theory comes in handy.

A Water Jug Invariant

Suppose that we have water jugs with capacities a and b with $b \geq a$. Let's carry out some sample operations of the state machine and see what happens, assuming the b-jug is big enough:

$$
\begin{array}{lll}
(0,0) & \to (a,0) & \text{fill first jug} \\
& \to (0,a) & \text{pour first into second} \\
& \to (a,a) & \text{fill first jug} \\
& \to (2a-b,b) & \text{pour first into second (assuming } 2a \geq b) \\
& \to (2a-b,0) & \text{empty second jug} \\
& \to (0,2a-b) & \text{pour first into second} \\
& \to (a,2a-b) & \text{fill first} \\
& \to (3a-2b,b) & \text{pour first into second (assuming } 3a \geq 2b)
\end{array}
$$

What leaps out is that at every step, the amount of water in each jug is a linear combination of a and \b\). This is easy to prove by induction on the number of transitions:

Lemma 8.1.5 (Water Jugs). *In the* Die Hard *state machine of Section 5.4.4 with jugs of sizes a and b, the amount of water in each jug is always a linear combination of a and b.*

Proof. The induction hypothesis, $P(n)$, is the proposition that after n transitions, the amount of water in each jug is a linear combination of a and b.

Base case ($n = 0$): $P(0)$ is true, because both jugs are initially empty, and $0 \cdot a + 0 \cdot b = 0$.

Inductive step: Suppose the machine is in state (x, y) after n steps, that is, the little jug contains x gallons and the big one contains y gallons. There are two cases:

If we fill a jug from the fountain or empty a jug into the fountain, then that jug is empty or full. The amount in the other jug remains a linear combination of a and b. So $P(n+1$ holds.

Otherwise, we pour water from one jug to another until one is empty or the other is full. By our assumption, the amount x and y in each jug is a linear combination of a and b before we begin pouring. After pouring, one jug is either empty (contains 0 gallons) or full (contains a or b gallons). Thus, the other jug contains either $x + y$ gallons, $x + y - a$, or $x + y - b$ gallons, all of which are linear combinations of a and b since x and y are. So $P(n+1$ holds in this case as well.

Since $P(n+1$ holds in any case, this proves the inductive step, completing the proof by induction. ■

So we have established that the jug problem has a preserved invariant, namely, the amount of water in every jug is a linear combination of the capacities of the jugs. Lemma 8.1.5 has an important corollary:

Corollary. *In trying to get 4 gallons from 12- and 18-gallon jugs, and likewise to get 32 gallons from 899- and 1147-gallon jugs,*

Bruce will die!

Proof. By the Water Jugs Lemma 8.1.5, with 12- and 18-gallon jugs, the amount in any jug is a linear combination of 12 and 18. This is always a multiple of 6 by Lemma 8.1.2.2, so Bruce can't get 4 gallons. Likewise, the amount in any jug using 899- and 1147-gallon jugs is a multiple of 31, so he can't get 32 either. ■

But the Water Jugs Lemma doesn't tell the complete story. For example, it leaves open the question of getting 3 gallons into a jug using just 21- and 26-gallon jugs: the only positive factor of both 21 and 26 is 1, and of course 1 divides 3, so the Lemma neither rules out nor confirms the possibility of getting 3 gallons.

A bigger issue is that we've just managed to recast a pretty understandable question about water jugs into a technical question about linear combinations. This might not seem like a lot of progress. Fortunately, linear combinations are closely related to something more familiar, greatest common divisors, and will help us solve the general water jug problem.

[1]*Don't Panic*—we're going to stick to some relatively benign parts of number theory. These super-hard unsolved problems rarely get put on problem sets.

[2]This theorem is often called the "Division Algorithm," but we prefer to call it a theorem since it does not actually describe a division procedure for computing the quotient and remainder.

This page titled 8.1: Divisibility is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

https://eng.libretexts.org/@go/page/48331

8.2: The Greatest Common Divisor

A *common divisor* of a and b is a number that divides them both. The *greatest common divisor* of a and b is written $\gcd(a, b)$. For example, $\gcd(18, 24) = 6$.

As long as a and b are not both 0, they will have a gcd. The gcd turns out to be very valuable for reasoning about the relationship between a and b and for reasoning about integers in general. We'll be making lots of use of gcd's in what follows.

Some immediate consequences of the definition of gcd are that for $n > 0$,

$$\gcd(n, n) = n, \quad \gcd(n, 1) = 1, \quad \gcd(n, 0) = 0$$

where the last equality follows from the fact that everything is a divisor of 0.

Euclid's Algorithm

The first thing to figure out is how to find gcd's. A good way called *Euclid's algorithm* has been known for several thousand years. It is based on the following elementary observation.

Lemma 8.2.1. For $b \neq 0$,

$$\gcd(a, b) = \gcd(b, \text{rem}(a, b)).$$

Proof. By the Division Theorem 8.1.4,

$$a = qb + r$$

where $r = \text{rem}(a, b)$. So a is a linear combination of b and r, which implies that any divisor of b and r is a divisor of a by Lemma 8.1.2.2. Likewise, r is a linear combination, $a - qb$, of a and b, so any divisor of a and b is a divisor of r. This means that a and b have the same common divisors as b and r, and so they have the same *greatest* common divisor. ■

Lemma 8.2.1 is useful for quickly computing the greatest common divisor of two numbers. For example, we could compute the greatest common divisor of 1147 and 899 by repeatedly applying it:

$$\begin{aligned}
\gcd(1147, 899) &= \gcd(899, \underbrace{\text{rem}(1147, 899)}_{=248}) \\
&= \gcd(248, \text{rem}(899, 248) = 155) \\
&= \gcd(155, \text{rem}(248, 155) = 93) \\
&= \gcd(93, \text{rem}(155, 93) = 62) \\
&= \gcd(62, \text{rem}(93, 62) = 31) \\
&= \gcd(31, \text{rem}(62, 31) = 0) \\
&= 31
\end{aligned} \tag{8.2.1}$$

This calculation that $\gcd(1147, 899) = 31$ was how we figured out that with water jugs of sizes 1147 and 899, Bruce dies trying to get 32 gallons.

On the other hand, applying Euclid's algorithm to 26 and 21 gives

$$\gcd(26, 21) = \gcd(21, 5) = \gcd(5, 1) = 1.$$

so we can't use the reasoning above to rule out Bruce getting 3 gallons into the big jug. As a matter of fact, because the gcd here is 1, Bruce *will* be able to get any number of gallons into the big jug up to its capacity. To explain this, we will need a little more number theory.

Euclid's Algorithm as a State Machine

Euclid's algorithm can easily be formalized as a state machine. The set of states is $\mathbb{N}^2$ and there is one transition rule:

$$(x, y) \longrightarrow (y, \text{rem}(x, y)), \tag{8.2.2}$$

for $y > 0$. By Lemma 8.2.1, the gcd stays the same from one state to the next. That means the predicate

$$\gcd(x, y) = \gcd(a, b).$$

is a preserved invariant on the states (x, y). This preserved invariant is, of course, true in the start state (a, b). So by the Invariant Principle, if y ever becomes 0, the invariant will be true and so

$$x = \gcd(x, 0) = \gcd(a, b).$$

Namely, the value of x will be the desired gcd.

What's more, x, and therefore also y, gets to be 0 pretty fast. To see why, note that starting from (x, y), two transitions leads to a state whose the first coordinate is $\text{rem}(x, y)$, which is at most half the size of x.[3] Since x starts off equal to a and gets halved or smaller every two steps, it will reach its minimum value—which is $\gcd(a, b)$—after at most 2 log a transitions. After that, the algorithm takes at most one more transition to terminate. In other words, Euclid's algorithm terminates after at most $1 + 2$ log a transitions.[4]

The Pulverizer

We will get a lot of mileage out of the following key fact:

Theorem 8.2.2

The greatest common divisor of a and b is a linear combination of a and b. That is,

$$\gcd(a, b) = sa + tb,$$

for some integers s and t.

We already know from Lemma 8.1.2.2 that every linear combination of a and b is divisible by any common factor of a and b, so it is certainly divisible by the greatest of these common divisors. Since any constant multiple of a linear combination is also a linear combination, Theorem 8.2.2 implies that any multiple of the gcd is a linear combination, giving:

Corollary 8.2.3. *An integer is a linear combination of a and b iff it is a multiple of* $\gcd(a, b)$.

We'll prove Theorem 8.2.2 directly by explaining how to find s and t. This job is tackled by a mathematical tool that dates back to sixth-century India, where it was called *kuttak*, which means "The Pulverizer." Today, the Pulverizer is more commonly known as "the extended Euclidean gcd algorithm," because it is so close to Euclid's algorithm.

For example, following Euclid's algorithm, we can compute the gcd of 259 and 70 as follows:

$$\begin{aligned}
\gcd(259, 70) &= \gcd(70, 49) && \text{since } \text{rem}(259, 70) = 49 \\
&= \gcd(49, 21) && \text{since } \text{rem}(70, 49) = 21 \\
&= \gcd(21, 7) && \text{since } \text{rem}(49, 21) = 7 \\
&= \gcd(7, 0) && \text{since } \text{rem}(21, 7) = 0 \\
&= 7.
\end{aligned}$$

The Pulverizer goes through the same steps, but requires some extra bookkeeping along the way: as we compute $\gcd(a, b)$, we keep track of how to write each of the remainders (49, 21, and 7, in the example) as a linear combination of a and b. This is worthwhile, because our objective is to write the last nonzero remainder, which is the GCD, as such a linear combination. For our example, here is this extra bookkeeping:

x	y	$(\text{rem}(x,\ y))$	$=$	$x - q \cdot y$
259	70	49	$=$	$a - 3 \cdot b$
70	49	21	$=$	$b - 1 \cdot 49$
			$=$	$b - 1 \cdot (a - 3 \cdot b)$
			$=$	$-1 \cdot a + 4 \cdot b$
49	21	7	$=$	$49 - 2 \cdot 21$
			$=$	$(a - 3 \cdot b) - 2 \cdot (-1 \cdot a + 4 \cdot b)$
			$=$	$\boxed{3 \cdot a - 11 \cdot b}$
21	7	0		

We began by initializing two variables, $x = a$ and $y = b$. In the first two columns above, we carried out Euclid's algorithm. At each step, we computed $\text{rem}(x, y)$ which equals $x - \text{qcnt}(x, y) \cdot y$. Then, in this linear combination of x and y, we replaced x and y by

equivalent linear combinations of a and b, which we already had computed. After simplifying, we were left with a linear combination of a and b equal to $\text{rem}(x, y)$, as desired. The final solution is boxed.

This should make it pretty clear how and why the Pulverizer works. If you have doubts, it may help to work through Problem 8.13, where the Pulverizer is formalized as a state machine and then verified using an invariant that is an extension of the one used for Euclid's algorithm.

Since the Pulverizer requires only a little more computation than Euclid's algorithm, you can "pulverize" very large numbers very quickly by using this algorithm. As we will soon see, its speed makes the Pulverizer a very useful tool in the field of cryptography.

Now we can restate the Water Jugs Lemma 8.1.5 in terms of the greatest common divisor:

Corollary 8.2.4. *Suppose that we have water jugs with capacities a and b. Then the amount of water in each jug is always a multiple of* $\gcd(a, b)$.

For example, there is no way to form 4 gallons using 3- and 6-gallon jugs, because 4 is not a multiple of $\gcd(3, 6) = 3$.

One Solution for All Water Jug Problems

Corollary 8.2.3 says that 3 can be written as a linear combination of 21 and 26, since 3 is a multiple of $\gcd(21, 26) = 1$. So the Pulverizer will give us integers s and t such that

$$3 = s \cdot 21 + t \cdot 26 \tag{8.2.3}$$

The coefficient s could be either positive or negative. However, we can readily transform this linear combination into an equivalent linear combination

$$3 = s' \cdot 21 + t' \cdot 26 \tag{8.2.4}$$

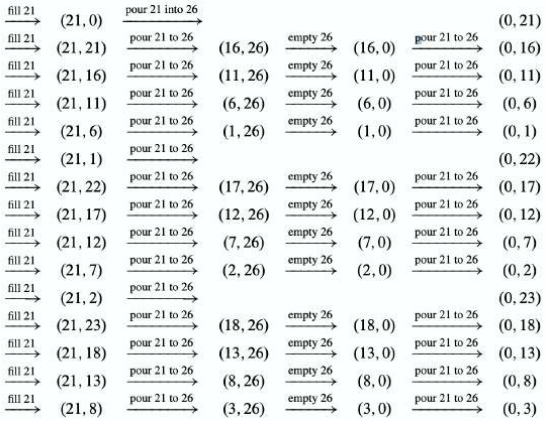
where the coefficient s' is positive. The trick is to notice that if in equation (8.2.3) we increase s by 26 and decrease t by 21, then the value of the expression $s \cdot 21 + t \cdot 26$ is unchanged overall. Thus, by repeatedly increasing the value of s (by 26 at a time) and decreasing the value of t (by 21 at a time), we get a linear combination $s' \cdot 21 + t' \cdot 26 = 3$ where the coefficient s' is positive. (Of course t' must then be negative; otherwise, this expression would be much greater than 3.)

Now we can form 3 gallons using jugs with capacities 21 and 26: We simply repeat the following steps s' times:

1. Fill the 21-gallon jug.
2. Pour all the water in the 21-gallon jug into the 26-gallon jug. If at any time the 26-gallon jug becomes full, empty it out, and continue pouring the 21- gallon jug into the 26-gallon jug.

At the end of this process, we must have emptied the 26-gallon jug exactly t' times. Here's why: we've taken $s' \cdot 21$ gallons of water from the fountain, and we've poured out some multiple of 26 gallons. If we emptied fewer than t' times, then by (8.2.4), the big jug would be left with at least $3 + 26$ gallons, which is more than it can hold; if we emptied it more times, the big jug would be left containing at most $3 - 26$ gallons, which is nonsense. But once we have emptied the 26-gallon jug exactly $-t'$ times, equation (8.2.4) implies that there are exactly 3 gallons left.

Remarkably, we don't even need to know the coefficients s' and t' in order to use this strategy! Instead of repeating the outer loop s' times, we could just repeat *until we obtain 3 gallons*, since that must happen eventually. Of course, we have to keep track of the amounts in the two jugs so we know when we're done. Here's the solution using this approach starting with empty jugs, that is, at (0, 0):



The same approach works regardless of the jug capacities and even regardless of the amount we're trying to produce! Simply repeat these two steps until the desired amount of water is obtained:

1. Fill the smaller jug.
2. Pour all the water in the smaller jug into the larger jug. If at any time the larger jug becomes full, empty it out, and continue pouring the smaller jug into the larger jug.

By the same reasoning as before, this method eventually generates every multiple— up to the size of the larger jug—of the greatest common divisor of the jug capacities, all the quantities we can possibly produce. No ingenuity is needed at all!

So now we have the complete water jug story:

Theorem 8.2.5

Suppose that we have water jugs with capacities a and b. For any $c \in [0..a]$, it is possible to get c gallons in the size a jug iff c is a multiple of $\gcd(a, b)$.

[3]In other words,

$$\text{rem}(x, y) \leq x/2 \quad \text{for } 0 < y \leq x. \tag{8.2.5}$$

This is immediate if $y \leq x/2$, since the remainder of x divided by y is less than y by definition. On the other hand, if $y > x/2$, then $\text{rem}(x, y) = x - y < x/2$.

[4]A tighter analysis shows that at most $\log_\phi(a)$ transitions are possible where ϕ is the golden ratio $(1 + \sqrt{5})/2$, see Problem 8.14.

This page titled 8.2: The Greatest Common Divisor is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.3: Prime Mysteries

Some of the greatest mysteries and insights in number theory concern properties of prime numbers:

Definition 8.3.1

A *prime* is a number greater than 1 that is divisible only by itself and 1. A number other than 0, 1, and -1 that is not a prime is called *composite*.[5]

Here are three famous mysteries:

Twin Prime Conjecture There are infinitely many primes p such that $p+2$ is also a prime.

In 1966, Chen showed that there are infinitely many primes p such that $p+2$ is the product of at most two primes. So the conjecture is known to be *almost* true!

Conjectured Inefficiency of Factoring Given the product of two large primes $n = pq$, there is no efficient procedure to recover the primes p and q. That is, no *polynomial time* procedure (see Section 3.5) is guaranteed to find p and q in a number of steps bounded by a polynomial in the length of the binary representation of n (not n itself). The length of the binary representation at most $1 + \log_2 n$.

The best algorithm known is the "number field sieve," which runs in time proportional to:

$$e^{1.9(\ln n)^{1/3}(\ln \ln n)^{2/3}}.$$

This number grows more rapidly than any polynomial in $\log n$ and is infeasible when n has 300 digits or more.

Efficient factoring is a mystery of particular importance in computer science, as we'll explain later in this chapter.

Goldbach's Conjecture We've already mentioned Goldbach's Conjecture 1.1.8 several times: every even integer greater than two is equal to the sum of two primes. For example, $4 = 2 + 2, 6 = 3 + 3, 8 = 3 + 5$, etc.

In 1939, Schnirelman proved that every even number can be written as the sum of not more than 300,000 primes, which was a start. Today, we know that every even number is the sum of at most 6 primes.

Primes show up erratically in the sequence of integers. In fact, their distribution seems almost random:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, \ldots$$

One of the great insights about primes is that their density among the integers has a precise limit. Namely, let $\pi(n)$ denote the number of primes up to n:

Definition 8.3.2

$$\pi(n) ::= |\{p \in [2..n] \mid p \text{ is prime}\}|.$$

For example, $\pi(1) = 0, \pi(2) = 1$, and $\pi(10) = 4$ because 2, 3, 5, and 7 are the primes less than or equal to 10. Step by step, π grows erratically according to the erratic spacing between successive primes, but its overall growth rate is known to smooth out to be the same as the growth of the function $n/\ln n$:

Theorem 8.3.3

(Prime Number Theorem).

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1$$

Thus, primes gradually taper off. As a rule of thumb, about 1 integer out of every $\ln n$ in the vicinity of n is a prime.

The Prime Number Theorem was conjectured by Legendre in 1798 and proved a century later by de la Vallée Poussin and Hadamard in 1896. However, after his death, a notebook of Gauss was found to contain the same conjecture, which he apparently

made in 1791 at age 15. (You have to feel sorry for all the otherwise "great" mathematicians who had the misfortune of being contemporaries of Gauss.)

A proof of the Prime Number Theorem is beyond the scope of this text, but there is a manageable proof (see Problem 8.22) of a related result that is sufficient for our applications:

Theorem 8.3.4

(Chebyshev's Theorem on Prime Density). *For* $n > 1$,

$$\pi(n) > \frac{n}{3 \ln n}.$$

A Prime for Google

In late 2004 a billboard appeared in various locations around the country:

$$\left\{ \begin{array}{l} \text{first 10 -digit prime found} \\ \text{in consecutive digits of } e \end{array} \right\}.\text{com}$$

Substituting the correct number for the expression in curly-braces produced the URL for a Google employment page. The idea was that Google was interested in hiring the sort of people that could and would solve such a problem.

How hard is this problem? Would you have to look through thousands or millions or billions of digits of e to find a 10-digit prime? The rule of thumb derived from the Prime Number Theorem says that among 10-digit numbers, about 1 in

$$\ln 10^{10} \approx 23$$

is prime. This suggests that the problem isn't really so hard! Sure enough, the first 10-digit prime in consecutive digits of e appears quite early:

$$e = 2.71828182845904523536028747135266249775724709369995957496696762772407663035354759457138217852516642742746639193200305992181741359662904357290033429526059563073813232862794349 \ldots$$

[5]So 0, 1, and -1 are the only integers that are neither prime nor composite.

This page titled 8.3: Prime Mysteries is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.4: The Fundamental Theorem of Arithmetic

There is an important fact about primes that you probably already know: every positive integer number has a *unique* prime factorization. So every positive integer can be built up from primes in *exactly one way*. These quirky prime numbers are the building blocks for the integers.

Since the value of a product of numbers is the same if the numbers appear in a different order, there usually isn't a unique way to express a number as a product of primes. For example, there are three ways to write 12 as a product of primes:

$$12 = 2 \cdot 2 \cdot 3 = 2 \cdot 3 \cdot 2 = 3 \cdot 2 \cdot 2.$$

What's unique about the prime factorization of 12 is that any product of primes equal to 12 will have exactly one 3 and two 2's. This means that if we *sort* the primes by size, then the product really will be unique.

Let's state this more carefully. A sequence of numbers is *weakly decreasing* when each number in the sequence is at least as big as the numbers after it. Note that a sequence of just one number as well as a sequence of no numbers—the empty sequence —is weakly decreasing by this definition.

Theorem 8.4.1

[Fundamental Theorem of Arithmetic] Every positive integer is a product of a unique *weakly decreasing sequence of primes.*

For example, 75237393 is the product of the weakly decreasing sequence of primes

$$23, 17, 17, 11, 7, 7, 7, 3,$$

and no other weakly decreasing sequence of primes will give 75237393.[6]

Notice that the theorem would be false if 1 were considered a prime; for example, 15 could be written as $5 \cdot 3$, or $5 \cdot 3 \cdot 1$, or $5 \cdot 3 \cdot 1 \cdot 1, \ldots$

There is a certain wonder in unique factorization, especially in view of the prime number mysteries we've already mentioned. It's a mistake to take it for granted, even if you've known it since you were in a crib. In fact, unique factorization actually fails for many integer-like sets of numbers, such as the complex numbers of the form $n + m\sqrt{-5}$ for $m, n \in \mathbb{Z}$ (see Problem 8.25).

The Fundamental Theorem is also called the *Unique Factorization Theorem*, which is a more descriptive and less pretentious, name —but we really want to get your attention to the importance and non-obviousness of unique factorization.

Proving Unique Factorization

The Fundamental Theorem is not hard to prove, but we'll need a couple of preliminary facts.

Lemma 8.4.2. *If p is a prime and $p \mid ab$, then $p \mid a$ or $p \mid b$.*

Lemma 8.4.2 follows immediately from Unique Factorization: the primes in the product ab are exactly the primes from a and from b. But proving the lemma this way would be cheating: we're going to need this lemma to prove Unique Factorization, so it would be circular to assume it. Instead, we'll use the properties of gcd's and linear combinations to give an easy, noncircular way to prove Lemma 8.4.2.

Proof. One case is if $\gcd(a, p) = p$. Then the claim holds, because a is a multiple of p.

Otherwise, $\gcd(a, p) \neq p$. In this case $\gcd(a, p)$ must be 1, since 1 and p are the only positive divisors of p. Now $\gcd(a, p)$ is a linear combination of a and p, so we have $1 = sa + tp$ for some s, t. Then $b = s(ab) + (tb)p$, that is, b is a linear combination of ab and p. Since p divides both ab and p, it also divides their linear combination b. ■

A routine induction argument extends this statement to:

Lemma 8.4.3. *Let p be a prime. If $p \mid a_1 a_2 \cdots a_n$, then p divides some a_i.*

Now we're ready to prove the Fundamental Theorem of Arithmetic.

Proof. Theorem 2.3.1 showed, using the Well Ordering Principle, that every positive integer can be expressed as a product of primes. So we just have to prove this expression is unique. We will use Well Ordering to prove this too.

The proof is by contradiction: assume, contrary to the claim, that there exist positive integers that can be written as products of primes in more than one way. By the Well Ordering Principle, there is a smallest integer with this property. Call this integer n, and let

$$\begin{aligned} n &= p_1 \cdot p_2 \cdots p_j, \\ &= q_1 \cdot q_2 \cdots q_k, \end{aligned} \tag{8.4.1}$$

where both products are in weakly decreasing order and $p_1 \leq q_1$.

If $q_1 = p_1$, then n/q_1 would also be the product of different weakly decreasing sequences of primes, namely,

$$\begin{aligned} &p_2 \cdots p_j, \\ &q_2 \cdots q_k, \end{aligned} \tag{8.4.2}$$

Since $n/q_1 < n$, this can't be true, so we conclude that $p_1 < q_1$.

Since the p_i's are weakly decreasing, all the p_i's are less than q_1. But

$$q_1 \mid n = p_1 \cdot p_2 \cdot p_j,$$

so Lemma 8.4.3 implies that q_1 divides one of the p_i's, which contradicts the fact that q_1 is bigger than all them. ■

[6]The "product" of just one number is defined to be that number, and the product of no numbers is by convention defined to be 1. So each prime, p, is uniquely the product of the primes in the lengthone sequence consisting solely of p, and 1, which you will remember is not a prime, is uniquely the product of the empty sequence.

This page titled 8.4: The Fundamental Theorem of Arithmetic is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.5: Alan Turing

The man pictured in Figure 8.1 is Alan Turing, the most important figure in the history of computer science. For decades, his fascinating life story was shrouded by government secrecy, societal taboo, and even his own deceptions.



Figure 8.1 Alan Turing

At age 24, Turing wrote a paper entitled *On Computable Numbers, with an Application to the Entscheidungsproblem*. The crux of the paper was an elegant way to model a computer in mathematical terms. This was a breakthrough, because it allowed the tools of mathematics to be brought to bear on questions of computation. For example, with his model in hand, Turing immediately proved that there exist problems that no computer can solve—no matter how ingenious the programmer. Turing's paper is all the more remarkable because he wrote it in 1936, a full decade before any electronic computer actually existed.

The word "Entscheidungsproblem" in the title refers to one of the 28 mathematical problems posed by David Hilbert in 1900 as challenges to mathematicians of the 20th century. Turing knocked that one off in the same paper. And perhaps you've heard of the "Church-Turing thesis"? Same paper. So Turing was a brilliant guy who generated lots of amazing ideas. But this lecture is about one of Turing's less-amazing ideas. It involved codes. It involved number theory. And it was sort of stupid.

Let's look back to the fall of 1937. Nazi Germany was rearming under Adolf Hitler, world-shattering war looked imminent, and—like us —Alan Turing was pondering the usefulness of number theory. He foresaw that preserving military secrets would be vital in the coming conflict and proposed a way *to encrypt communications using number theory*. This is an idea that has ricocheted up to our own time. Today, number theory is the basis for numerous public-key cryptosystems, digital signature schemes, cryptographic hash functions, and electronic payment systems. Furthermore, military funding agencies are among the biggest investors in cryptographic research. Sorry, Hardy!

Soon after devising his code, Turing disappeared from public view, and half a century would pass before the world learned the full story of where he'd gone and what he did there. We'll come back to Turing's life in a little while; for now, let's investigate the code Turing left behind. The details are uncertain, since he never formally published the idea, so we'll consider a couple of possibilities.

Turing's Code (Version 1.0)

The first challenge is to translate a text message into an integer so we can perform mathematical operations on it. This step is not intended to make a message harder to read, so the details are not too important. Here is one approach: replace each letter of the message with two digits ($A = 01, B = 02, C = 03$, etc.) and string all the digits together to form one huge number. For example, the message "victory" could be translated this way:

	v	i	c	t	o	r	y
→	22	09	03	20	15	18	25

Turing's code requires the message to be a prime number, so we may need to pad the result with some more digits to make a prime. The Prime Number Theorem indicates that padding with relatively few digits will work. In this case, appending the digits 13 gives the number 2209032015182513, which is prime.

Here is how the encryption process works. In the description below, m is the unencoded message (which we want to keep secret), $\hat{m}$ is the encrypted message (which the Nazis may intercept), and k is the key.

Beforehand The sender and receiver agree on a *secret key*, which is a large prime k.

Encryption The sender encrypts the message m by computing:

$$\hat{m} = m \cdot k$$

Decryption The receiver decrypts mb by computing:

$$\frac{\hat{m}}{k} = m.$$

For example, suppose that the secret key is the prime number $k = 22801763489$ and the message m is "victory." Then the encrypted message is:

$$\begin{aligned} \hat{m} &= m \cdot k \\ &= 2209032015182513 \cdot 22801763489 \\ &= 50369825549820718594667857 \end{aligned} \tag{8.5.1}$$

There are a couple of basic questions to ask about Turing's code.

1. How can the sender and receiver ensure that m and k are prime numbers, as required?

   The general problem of determining whether a large number is prime or composite has been studied for centuries, and tests for primes that worked well in practice were known even in Turing's time. In the past few decades, very fast primality tests have been found as described in the text box below.

2. Is Turing's code secure?

   The Nazis see only the encrypted message $\hat{m} = m \cdot k$, so recovering the original message m requires factoring $\hat{m}$. Despite ef b immense efforts, no really ficient factoring algorithm has ever been found. It appears to be a fundamentally difficult problem. So, although a breakthrough someday can't be ruled out, the conjecture that there is no efficient way to factor is widely accepted. In effect, Turing's code puts to practical use his discovery that there are limits to the power of computation. Thus, provided m and k are sufficiently large, the Nazis seem to be out of luck!

This all sounds promising, but there is a major flaw in Turing's code.

Primality Testing

It's easy to see that an integer n is prime iff it is not divisible by any number from 2 to $\lfloor\sqrt{n}\rfloor$ (see Problem 1.9). Of course this naive way to test if n is prime takes more than $\lfloor\sqrt{n}\rfloor$ steps, which is exponential in the *size* of n measured by the number of digits in the decimal or binary representation of n. Through the early 1970's, no prime testing procedure was known that would never blow up like this.

In 1974, Volker Strassen invented a simple, fast *probabilistic* primality test. Strassens's test gives the right answer when applied to any prime number, but has some probability of giving a wrong answer on a nonprime number. However, the probability of a wrong answer on any given number is so tiny that relying on the answer is the best bet you'll ever make.

Still, the theoretical possibility of a wrong answer was intellectually bothersome—even if the probability of being wrong was a lot less than the probability of an undetectable computer hardware error leading to a wrong answer. Finally in 2002, in a breakthrough paper beginning with a quote from Gauss emphasizing the importance and antiquity of primality testing, Manindra Agrawal, Neeraj Kayal, and Nitin Saxena presented an amazing, thirteen line description of a polynomial time primality test.

This definitively places primality testing way below the exponential effort apparently needed for SAT and similar problems. The polynomial bound on the Agrawal *et al*. test had degree 12, and subsequent research has reduced the degree to 5, but this is still too large to be practical, and probabilistic primality tests remain the method used in practice today. It's plausible that the degree bound can be reduced a bit more, but matching the speed of the known probabilistic tests remains a daunting challenge.

Breaking Turing's Code (Version 1.0)

Let's consider what happens when the sender transmits a *second* message using Turing's code and the same key. This gives the Nazis two encrypted messages to look at:

$$\hat{m_1} = m_1 \cdot k \backslash) and \backslash (\hat{m_2} = m_2 \cdot k$$

The greatest common divisor of the two encrypted messages, $\hat{m_1}$ and $\hat{m_2}$, is the secret key k. And, as we've seen, the GCD of two numbers can be computed very efficiently. So after the second message is sent, the Nazis can recover the secret key and read *every* message!

A mathematician as brilliant as Turing is not likely to have overlooked such a glaring problem, and we can guess that he had a slightly different system in mind, one based on *modular* arithmetic.

This page titled 8.5: Alan Turing is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.6: Modular Arithmetic

On the first page of his masterpiece on number theory, *Disquisitiones Arithmeticae*, Gauss introduced the notion of "*congruence*." Now, Gauss is another guy who managed to cough up a half-decent idea every now and then, so let's take a look at this one. Gauss said that a *is congruent to* b *modulo* n iff $n \mid (a - b)$. This is written

$$a \equiv b \pmod{n}.$$

For example:

$$29 \equiv 15 \pmod{7} \text{ because } (7 \mid (29 - 15)$$

It's not useful to allow a modulus $n \leq 1$, *and so we will assume from now on that moduli are greater than 1*.

There is a close connection between congruences and remainders:

Lemma 8.6.1 (Remainder).

\[\nonumber a \equiv b \pmod n\) *iff* \(\text{rem}(a, n) = \text{rem}(b, n)\]

Proof. By the Division Theorem 8.1.4, there exist unique pairs of integers q_1, r_1 and q_2, r_2 such that:

$$\begin{aligned} a &= q_1 n + r1 \\ b &= q_2 n + r2, \end{aligned} \tag{8.6.1}$$

where $r_1, r_2 \in [0..n)$. Subtracting the second equation from the first gives:

$$a - b = (q_1 - q_2)n + (r_1 - r_2)$$

where $r_1 - r_2$ is in the interval $(-n, n)$. Now $a \equiv b \pmod{n}$ if and only if n divides the left side of this equation. This is true if and only if n divides the right side, which holds if and only if $r_1 - r_2$ is a multiple of n. But the only multiple of n in $(-n, n)$ is 0, so $r_1 - r_2$ must in fact equal 0, that is, when $r_1 ::= \text{rem}(a, n) = r_2 ::= \text{rem}(b, n)$. ■

So we can also see that

$$29 \equiv 15 \pmod{7} \text{ because } (\text{rem}(29, 7) = 1 = \text{rem}(15, 7)$$

Notice that even though "(mod 7)" appears on the end, the $\equiv$ symbol isn't any more strongly associated with the 15 than with the 29. It would probably be clearer to write $29 \equiv_{\text{mod } 7} 15$, for example, but the notation with the modulus at the end is firmly entrenched, and we'll just live with it.

The Remainder Lemma 8.6.1 explains why the congruence relation has properties like an equality relation. In particular, the following properties[7] follow immediately:

Lemma 8.6.2.

$$\begin{aligned} a \equiv a \pmod{n} \quad & \text{(reflexivity)} \\ a \equiv b \text{ IFF } b \equiv a \pmod{n} \quad & \text{(symmetry)} \\ (a \equiv b \text{ AND } b \equiv c) \text{ IMPLIES } a \equiv c \pmod{n} \quad & \text{(transitivity)} \end{aligned}$$

We'll make frequent use of another immediate corollary of the Remainder Lemma 8.6.1:

Corollary 8.6.3.

[\nonumber (a \equiv \text{rem}(a, n) \pmod n\]

Still another way to think about congruence modulo n is that it *defines a partition of the integers into* n *sets so that congruent numbers are all in the same set*. For example, suppose that we're working modulo 3. Then we can partition the integers into 3 sets as follows:

$$\begin{aligned} &\{\ldots, -6, -3, 0, 3, 6, 9, \ldots\} \\ &\{\ldots, -5, -2, 1, 4, 7, 10, \ldots\} \\ &\{\ldots, -4, -1, 2, 5, 8, 11, \ldots\} \end{aligned}$$

according to whether their remainders on division by 3 are 0, 1, or 2. The upshot is that when arithmetic is done modulo n, there are really only n different kinds of numbers to worry about, because there are only n possible remainders. In this sense, modular arithmetic is a simplification of ordinary arithmetic.

The next most useful fact about congruences is that they are *preserved* by addition and multiplication:

Lemma 8.6.4 (Congruence). *If* $a \equiv b \pmod{n}$ *and* $c \equiv d \pmod{n}$, *then*

$$a + c \equiv b + d \pmod{n}, \tag{8.6.2}$$

$$ac \equiv bd \pmod{n}. \tag{8.6.3}$$

Proof. Let's start with 8.6.2. Since $a \equiv b \pmod{n}$, we have by definition that $n \mid (b-a) = (b+c) - (a+c)$, so

$$a + c \equiv b + c \pmod{n}.$$

Since $c \equiv d \pmod{n}$, the same reasoning leads to

$$b + c \equiv b + d \pmod{n}.$$

Now transitivity (Lemma 8.6.2) gives

$$a + c \equiv b + d \pmod{n}.$$

The proof for 8.6.3 is virtually identical, using the fact that if n divides $b - a$, then it certainly also divides $(bc - ac)$ ■.

[7]Binary relations with these properties are called *equivalence relations*, see Section 9.10.

This page titled 8.6: Modular Arithmetic is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.7: Remainder Arithmetic

The Congruence Lemma 8.6.1 says that two numbers are congruent iff their remainders are equal, so we can understand congruences by working out arithmetic with remainders. And if all we want is the remainder modulo n of a series of additions, multiplications, subtractions applied to some numbers, we can take remainders at every step so that the entire computation only involves number in the range $[0..n)$.

General Principle of Remainder Arithmetic

To find the remainder on division by n of the result of a series of additions and multiplications, applied to some integers

- replace each integer operand by its remainder on division by n,
- keep each result of an addition or multiplication in the range $[0..n)$ by immediately replacing any result outside that range by its remainder on division by n.

For example, suppose we want to find

$$\text{rem}((44427^{3456789} + 15555858^{5555})403^{6666666}, 36). \tag{8.7.1}$$

This looks really daunting if you think about computing these large powers and then taking remainders. For example, the decimal representation of $44427^{3456789}$ has about 20 million digits, so we certainly don't want to go that route. But remembering that integer exponents specify a series of multiplications, we follow the General Principle and replace the numbers being multiplied by their remainders. Since $\text{rem}(44427, 36) = 3$, $\text{rem}15555858, 36) = 6$, and $\text{rem}403, 36) = 7$, we find that (8.7.1) equals the remainder on division by 36 of

$$(3^{3456789} + 6^{5555})7^{6666666}. \tag{8.7.2}$$

That's a little better, but $3^{3456789}$ has about a million digits in its decimal representation, so we still don't want to compute that. But let's look at the remainders of the first few powers of 3:

$$\begin{aligned} \text{rem}(3, 36) &= 3 \\ \text{rem}(3^2, 36) &= 9 \\ \text{rem}(3^3, 36) &= 27 \\ \text{rem}(3^4, 36) &= 9. \end{aligned} \tag{8.7.3}$$

We got a repeat of the second step, $\text{rem}(3^2, 36)$ after just two more steps. This means means that starting at 3^2, the sequence of remainders of successive powers of 3 will keep repeating every 2 steps. So a product of an odd number of at least three 3's will have the same remainder on division by 36 as a product of just three 3's. Therefore,

$$\text{rem}(3^{3456789}, 36) = \text{rem}(3^3, 36) = 27.$$

What a win!

Powers of 6 are even easier because $\text{rem}(6^2, 36) = 0$, so 0's keep repeating after the second step. Powers of 7 repeat after six steps, but on the fifth step you get a 1, that is $\text{rem}(7^6, 36) = 1$, so (8.7.2) successively simplifies to be the remainders of the following terms:

$$\begin{aligned} &(3^{3456789} + 6^{5555})7^{6666666} \\ &(3^3 + 6^2 \cdot 6^{5553})(7^6)^{1111111} \\ &(3^3 + 0 \cdot 6^{5553})1^{1111111} \\ &= 27 \end{aligned} \tag{8.7.4}$$

Notice that *it would be a disastrous blunder to replace an exponent by its remainder*. The general principle applies to numbers that are *operands* of plus and times, whereas the exponent is a number that controls how many multiplications to perform. Watch out for this.

The ring $\mathbb{Z}_n$

It's time to be more precise about the general principle and why it works. To begin, let's introduce the notation $+n$ for doing an addition and then immediately taking a remainder on division by n, as specified by the general principle; likewise for multiplying:

$$\begin{aligned} i +_n j &::= \text{rem}(i+j, n), \\ i \cdot_n j &::= \text{rem}(ij, n). \end{aligned} \tag{8.7.5}$$

Now the General Principle is simply the repeated application of the following lemma.

Lemma 8.7.1.

$$\text{rem}(i+j, n) = \text{rem}(i, n) +_n \text{rem}(j, n), \tag{8.7.6}$$
$$\text{rem}(ij, n) = \text{rem}(i, n) +_n \text{rem}(j, n), \tag{8.7.7}$$

Proof. By Corollary 8.6.3, $i \equiv \text{rem}(i, n)$ and $j \equiv \text{rem}(j, n)$, so by the Congruence Lemma 8.6.4

$$i + j \equiv \text{rem}(i, n) + \text{rem}(j, n) \pmod{n}.$$

By Corollary 8.6.3 again, the remainders on each side of this congruence are equal, which immediately gives (8.7.6). An identical proof applies to (8.7.7). ■

The set of integers in the range $[0..n)$ together with the operations $+_n$ and $\cdot n$ is referred to as $\mathbb{Z}_n$, the *ring of integers modulo* n. As a consequence of Lemma 8.7.1, the familiar rules of arithmetic hold in $\mathbb{Z}_n$, for example:

$$(i \cdot_n j) \cdot_n k = i \cdot_n (j \cdot_n k).$$

These subscript-n's on arithmetic operations really clog things up, so instead we'll just write "$(\mathbb{Z}_n)$" on the side to get a simpler looking equation:

$$(i \cdot j) \cdot k = i \cdot (j \cdot k) \, (\mathbb{Z}_n)$$

In particular, all of the following equalities[8] are true in $\mathbb{Z}_n$:

$$\begin{aligned} (i \cdot j) \cdot k &= i \cdot (j \cdot k) && \text{(associativity of } \cdot), \\ (i + j) + k &= i + (j + k) && \text{(associativity of } +), \\ 1 \cdot k &= k && \text{identity for } \cdot), \\ 0 + k &= k && \text{(identity for } +), \\ k + (-k) &= 0 && \text{(inverse for } +), \\ i + j &= j + i && \text{(commutativity of } +) \\ i \cdot (j + k) &= (i \cdot j) + (i \cdot k) && \text{(distributivity)}, \\ i \cdot j &= j \cdot i && \text{(commutativity of } \cdot) \end{aligned} \tag{8.7.8}$$

Associativity implies the familiar fact that it's safe to omit the parentheses in products:

$$k_1 \cdot k_2 \cdots k_m$$

comes out the same in $\mathbb{Z}_n$ no matter how it is parenthesized.

The overall theme is that remainder arithmetic is a lot like ordinary arithmetic. But there are a couple of exceptions we're about to examine.

[8] A set with addition and multiplication operations that satisfy these equalities is known as a commutative ring. In addition to $\mathbb{Z}_n$, the integers, rationals, reals, and polynomials with integer coefficients are all examples of commutative rings. On the other hand, the set $\{\mathbf{T}, \mathbf{F}\}$ of truth values with **OR** for addition and **AND** for multiplication is *not* a commutative ring because it fails to satisfy one of these equalities. The $n \times n$ matrices of integers are not a commutative ring because they fail to satisfy another one of these equalities.

This page titled 8.7: Remainder Arithmetic is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.8: Turing's Code (Version 2.0)

In 1940, France had fallen before Hitler's army, and Britain stood alone against the Nazis in western Europe. British resistance depended on a steady flow of supplies brought across the north Atlantic from the United States by convoys of ships. These convoys were engaged in a cat-and-mouse game with German "U-boats" —submarines—which prowled the Atlantic, trying to sink supply ships and starve Britain into submission. The outcome of this struggle pivoted on a balance of information: could the Germans locate convoys better than the Allies could locate U-boats, or vice versa?

Germany lost.

A critical reason behind Germany's loss was not made public until 1974: Germany's naval code, *Enigma*, had been broken by the Polish Cipher Bureau,[9] and the secret had been turned over to the British a few weeks before the Nazi invasion of Poland in 1939. Throughout much of the war, the Allies were able to route convoys around German submarines by listening in to German communications. The British government didn't explain *how* Enigma was broken until 1996. When the story was finally released (by the US), it revealed that Alan Turing had joined the secret British codebreaking effort at Bletchley Park in 1939, where he became the lead developer of methods for rapid, bulk decryption of German Enigma messages. Turing's Enigma deciphering was an invaluable contribution to the Allied victory over Hitler.

Governments are always tight-lipped about cryptography, but the half-century of official silence about Turing's role in breaking Enigma and saving Britain may be related to some disturbing events after the war—more on that later. Let's get back to number theory and consider an alternative interpretation of Turing's code. Perhaps we had the basic idea right (multiply the message by the key), but erred in using *conventional* arithmetic instead of *modular* arithmetic. Maybe this is what Turing meant:

Beforehand The sender and receiver agree on a large number n, which may be made public. (This will be the modulus for all our arithmetic.) As in Version 1.0, they also agree that some prime number $k < n$ will be the secret key.

Encryption As in Version 1.0, the message m should be another prime in $[0..n)$. The sender encrypts the message m to produce $\hat{m}$ by computing mk, but this time modulo n:

$$\hat{m} ::= m \cdot k (\mathbb{Z}_n) \tag{8.8.1}$$

Decryption (Uh-oh.)

The decryption step is a problem. We might hope to decrypt in the same way as before by dividing the encrypted message $\hat{m}$ by the key k. The difficulty is that $\hat{m}$ is the *remainder* when mk is divided by n. So dividing $\hat{m}$ by k might not even give us an integer!

This decoding difficulty can be overcome with a better b understanding of when it is ok to divide by k in modular arithmetic.

[9]See http://en.Wikipedia.org/wiki/Polish_Cipher_Bureau.

This page titled 8.8: Turing's Code (Version 2.0) is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.9: Multiplicative Inverses and Cancelling

The *multiplicative inverse* of a number x is another number x^{-1} such that

$$x^{-1} \cdot x = 1$$

From now on, when we say "inverse," we mean *multiplicative* (not relational) inverse.

For example, over the rational numbers, $1/3$ is, of course, an inverse of 3, since,

$$\frac{1}{3} \cdot 3 = 1.$$

In fact, with the sole exception of 0, every rational number n/m has an inverse, namely, m/n. On the other hand, over the integers, only 1 and -1 have inverses. Over the ring $\mathbb{Z}_n$, things get a little more complicated. For example, in $\mathbb{Z}^{15}$, 2 is a multiplicative inverse of 8, since

$$2 \cdot 8 = 1(\mathbb{Z}_{15}).$$

On the other hand, 3 does not have a multiplicative inverse in $\mathbb{Z}_{15}$. We can prove this by contradiction: suppose there was an inverse j for 3, that is

$$1 = 3 \cdot j(\mathbb{Z}_{15}).$$

Then multiplying both sides of this equality by 5 leads directly to the contradiction $5 = 0$:

$$\begin{aligned} 5 &= 5 \cdot (3 \cdot j) \\ &= (5 \cdot 3) \cdot j \\ &= 0 \cdot j = 0(\mathbb{Z}_{15}) \end{aligned} \tag{8.9.1}$$

So there can't be any such inverse j.

So some numbers have inverses modulo 15 and others don't. This may seem a little unsettling at first, but there's a simple explanation of what's going on.

Relative Primality

Integers that have no prime factor in common are called *relatively prime*.[10] This is the same as having no common divisor (prime or not) greater than 1. It's also equivalent to saying $\gcd(a, b) = 1$.

For example, 8 and 15 are relatively prime, since $\gcd(8, 15) = 1$. On the other hand, 3 and 15 are not relatively prime, since $\gcd(3, 15) = 3 \neq 1$. This turns out to explain why 8 has an inverse over $\mathbb{Z}_{15}$ and 3 does not.

Lemma 8.9.1. *If $k \in [0..n)$ is relatively prime to n, then k has an inverse in $\mathbb{Z}_n$.*

Proof. If k is relatively prime to n, then $\gcd(n, k) = 1$ by definition of gcd. This means we can use the Pulverizer from section 8.2.2 to find a linear combination of n and k equal to 1:

$$sn + tk = 1.$$

So applying the General Principle of Remainder Arithmetic (Lemma 8.7.1), we get

$$\mathrm{rem}(s, n) \cdot \mathrm{rem}(n, n)) + (\mathrm{rem}(t, n) \cdot \mathrm{rem}(k, n)) = 1(\mathbb{Z}_n).$$

But $\mathrm{rem}(n, n) = 0$, and $\mathrm{rem}(k, n) = k$ since $k \in [0..n)$, so we get

$$\mathrm{rem}(t, n) \cdot k = 1(\mathbb{Z}_n)$$

Thus, $\mathrm{rem}(t, n)$ is a multiplicative inverse of k. ■

By the way, it's nice to know that when they exist, inverses are unique. That is,

Lemma 8.9.2. *If i and j are both inverses of k in $\mathbb{Z}_n$, then $i = j$.*

Proof.

$$i = i \cdot 1 = i \cdot (k \cdot j) = (i \cdot k) \cdot j = 1 \cdot j = j (\mathbb{Z}_n). \quad \blacksquare$$

So the proof of Lemma 8.9.1 shows that for any k relatively prime to n, the inverse of k in $\mathbb{Z}_n$ is simply the remainder of a coefficient we can easily find using the Pulverizer.

Working with a prime modulus is attractive here because, like the rational and real numbers, when p is prime, every nonzero number has an inverse in $\mathbb{Z}_p$. But arithmetic modulo a composite is really only a little more painful than working modulo a prime—though you may think this is like the doctor saying, "This is only going to hurt a little," before he jams a big needle in your arm.

Cancellation

Another sense in which real numbers are nice is that it's ok to cancel common factors. In other words, if we know that $tr = ts$ for real numbers r, s, t, then as long as $t \neq 0$, we can cancel the t's and conclude that $r = s$. In general, cancellation is not valid in $\mathbb{Z}_n$. For example,

$$3 \cdot 10 = 3 \cdot 5 (\mathbb{Z}_{15}), \tag{8.9.2}$$

but cancelling the 3's leads to the absurd conclusion that 10 equals 5.

The fact that multiplicative terms cannot be cancelled is the most significant way in which $\mathbb{Z}_n$ arithmetic differs from ordinary integer arithmetic.

Definition 8.9.3

A number k is *cancellable* in $\mathbb{Z}^n$ iff

$$k \cdot a = k \cdot b \text{ implies } a = b (\mathbb{Z}_n)$$

for all $a, b \in [0..n)$.

If a number is relatively prime to 15, it can be cancelled by multiplying by its inverse. So cancelling works for numbers that have inverses:

Lemma 8.9.4. *If k has an inverse in $\mathbb{Z}_n$, then it is cancellable.*

But 3 is not relatively prime to 15, and that's why it is not cancellable. More generally, if k is not relatively prime to n, then we can show it isn't cancellable in $\mathbb{Z}_n$ in the same way we showed that 3 is not cancellable in (8.9.2).

To summarize, we have

Theorem 8.9.5

The following are equivalent for $k \in [0..n)$:

$$\gcd(k, n) = 1,$$

k has an inverse in $\mathbb{Z}_n$,

k is cancellable in $\mathbb{Z}_n$.

Decrypting (Version 2.0)

Multiplicative inverses are the key to decryption in Turing's code. Specifically, we can recover the original message by multiplying the encoded message by the $\mathbb{Z}_n$-inverse, j, of the key:

$$\hat{m} \cdot j = (m \cdot k) \cdot j = m \cdot (k \cdot j) = m \cdot 1 = m (\mathbb{Z}_n).$$

So all we need to decrypt the message is to find an inverse of the secret key k, which will be easy using the Pulverizer—providing k has an inverse. But k is positive and less than the modulus n, so one simple way to ensure that k is relatively prime to the modulus is to have n be a prime number.

Breaking Turing's Code (Version 2.0)

The Germans didn't bother to encrypt their weather reports with the highly-secure Enigma system. After all, so what if the Allies learned that there was rain off the south coast of Iceland? But amazingly, this practice provided the British with a critical edge in the Atlantic naval battle during 1941.

The problem was that some of those weather reports had originally been transmitted using Enigma from U-boats out in the Atlantic. Thus, the British obtained both unencrypted reports and the same reports encrypted with Enigma. By comparing the two, the British were able to determine which key the Germans were using that day and could read all other Enigma-encoded traffic. Today, this would be called a *known-plaintext attack*.

Let's see how a known-plaintext attack would work against Turing's code. Suppose that the Nazis know both the plain text, m, and its encrypted form, $\hat{m}$. Now in ersion 2.0,

$$\hat{m} = m \cdot k(\mathbb{Z}_n).$$

and since m is positive and less than b the prime n, the Nazis can use the Pulverizer to find the $\mathbb{Z}_n$)-inverse, j, of m. Now

$$j \cdot \hat{m} = j \cdot (m \cdot k) = j \cdot (m \cdot k) = 1 \cdot k = k(\mathbb{Z}_n).$$

So by computing $j \cdot \hat{m} = k(\mathbb{Z}_n)$, the Nazis get the secret key and can then decrypt any message!

This is a huge vulnerability b , so Turing's hypothetical Version 2.0 code has no practical value. Fortunately, Turing got better at cryptography after devising this code; his subsequent deciphering of Enigma messages surely saved thousands of lives, if not the whole of Britain.

Turing Postscript

A few years after the war, Turing's home was robbed. Detectives soon determined that a former homosexual lover of Turing's had conspired in the robbery. So they arrested him—that is, they arrested Alan Turing—because at that time in Britain, homosexuality was a crime punishable by up to two years in prison. Turing was sentenced to a hormonal "treatment" for his homosexuality: he was given estrogen injections. He began to develop breasts.

Three years later, Alan Turing, the founder of computer science, was dead. His mother explained what happened in a biography of her own son. Despite her repeated warnings, Turing carried out chemistry experiments in his own home. Apparently, her worst fear was realized: by working with potassium cyanide while eating an apple, he poisoned himself.

However, Turing remained a puzzle to the very end. His mother was a devout woman who considered suicide a sin. And, other biographers have pointed out, Turing had previously discussed committing suicide by eating a poisoned apple. Evidently, Alan Turing, who founded computer science and saved his country, took his own life in the end, and in just such a way that his mother could believe it was an accident.

Turing's last project before he disappeared from public view in 1939 involved the construction of an elaborate mechanical device to test a mathematical conjecture called the Riemann Hypothesis. This conjecture first appeared in a sketchy paper by Bernhard Riemann in 1859 and is now one of the most famous unsolved problems in mathematics.

[10]Other texts call them coprime.

This page titled 8.9: Multiplicative Inverses and Cancelling is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.10: Euler's Theorem

The RSA cryptosystem examined in the next section, and other current schemes for encoding secret messages, involve computing remainders of numbers raised to large powers. A basic fact about remainders of powers follows from a theorem due to Euler about congruences.

Definition 8.10.1

For $n > 0$, define [11]

$$\phi(n) ::= \textbf{the number of integers in } [0..n), \textbf{ that are relatively prime to } n.$$

This function ϕ is known as Euler's ϕ function.[12]

For example, $\phi(7) = 6$ because all 6 positive numbers in $[0..7)$ are relatively prime to the prime number 7. Only 0 is not relatively prime to 7. Also, $\phi(12) = 4$ since 1, 5, 7, and 11 are the only numbers in $[0..12)$ that are relatively prime to 12.

More generally, if p is prime, then $\phi(p) = p - 1$ since every positive number in $[0..p)$ is relatively prime to p. When n is composite, however, the ϕ function gets a little complicated. We'll get back to it in the next section.

Euler's Theorem is traditionally stated in terms of congruence:

Theorem

(Euler's Theorem). If n and k are relatively prime, then

$$k^{\phi(n)} \equiv 1 \pmod{n} \tag{8.10.1}$$

The Riemann Hypothesis

The formula for the sum of an infinite geometric series says:

$$1 + x + x^2 + x^3 + \cdots = \frac{1}{1-x}$$

Substituting $x = \frac{1}{2^s}, x = \frac{1}{3^s}, x = \frac{1}{5^s}$, and so on for each prime number gives a sequence of equations:

$$\begin{aligned} 1 + \frac{1}{2^s} + \frac{1}{2^{2s}} + \frac{1}{2^{3s}} + \cdots &= \frac{1}{1-1/2^s} \\ 1 + \frac{1}{3^s} + \frac{1}{3^{2s}} + \frac{1}{3^{3s}} + \cdots &= \frac{1}{1-1/3^s} \\ 1 + \frac{1}{5^s} + \frac{1}{5^{2s}} + \frac{1}{5^{3s}} + \cdots &= \frac{1}{1-1/5^s} \\ &\textit{etc.} \end{aligned} \tag{8.10.2}$$

Multiplying together all the left sides and all the right sides gives:

$$\sum_{n=1}^{\infty} \frac{1}{n^s} = \prod_{p \in \text{ primes}} \left(\frac{1}{1-1/p^s}\right)$$

The sum on the left is obtained by multiplying out all the infinite series and applying the Fundamental Theorem of Arithmetic. For example, the term $1/300^s$ in the sum is obtained by multiplying $1/2^{2s}$ from the first equation by $1/e^s$ in the second and $1/5^{2s}$ in the third. Riemann noted that every prime appears in the expression on the right. So he proposed to learn about the primes by studying the equivalent, but simpler expression on the left. In particular, he regarded s as a complex number and the left side as a function, $\xi(s)$. Riemann found that the distribution of primes is related to values of s for which $\xi(s) = 0$, which led to his famous conjecture:

Definition 8.9.6.

The Riemann Hypothesis: Every nontrivial zero of the zeta function $\xi(s)$ lies on the line $s = 1/2 + ci$ in the complex plane.

A proof would immediately imply, among other things, a strong form of the Prime Number Theorem.

Researchers continue to work intensely to settle this conjecture, as they have for over a century. It is another of the Millennium Problems whose solver will earn $1,000,000 from the Clay Institute.

Things get simpler when we rephrase Euler's Theorem in terms of $\mathbb{Z}_n$.

Definition 8.10.2

Let $\mathbb{Z}_n^*$ be the integers in $(0..n)$, that are relatively prime to n: [13]

$$\mathbb{Z}_n^* ::= \{k \in (0..n) \mid \gcd(k,n) = 1\}. \tag{8.10.3}$$

Consequently,

$$\phi(n) = |\mathbb{Z}_n^*|$$

Theorem 8.10.3

(Euler's Theorem for $\mathbb{Z}_n$). For all $k \in \mathbb{Z}_n^*$,

$$\backslash k^{\phi(n)} = 1(\mathbb{Z}_n) \tag{8.10.4}$$

Theorem 8.10.3 will follow from two very easy lemmas.

Let's start by observing that $\mathbb{Z}_n^*$ is closed under multiplication in $\mathbb{Z}_n$:

Lemma 8.10.4. *If $j, k \in \mathbb{Z}_n^*$, then $j \cdot nk \in \mathbb{Z}_n^*$.*

There are lots of easy ways to prove this (see Problem 8.67).

Definition 8.10.5

For any element k and subset S of $\mathbb{Z}_n$, let

$$kS ::= \{k \cdot_n s \mid s \in S\}$$

Lemma 8.10.6. If $k \in \mathbb{Z}_n^*$ and $S \subseteq \mathbb{Z}_n$, then

$$|kS| = |S|.$$

Proof. Since $k \in \mathbb{Z}_n^*$, by Theorem 8.9.5. it is cancellable. Therefore,

$$[ks = kt\,(\mathbb{Z}_n)] \quad \text{implies} \quad s = t.$$

So mulitplying by k in $\mathbb{Z}_n$ maps all the elements of S to distinct elements of kS, which implies S and kS are the same size. ■

Corollary 8.10.7. If $k \in \mathbb{Z}_n^*$

$$k\mathbb{Z}_n^* = \mathbb{Z}_n^*.$$

Proof. A product of elements in $\mathbb{Z}_n^*$ remains in $\mathbb{Z}_n^*$ by Lemma 8.10.4. So if $k \in \mathbb{Z}_n^*$, then $k\mathbb{Z}_n^* \subseteq \mathbb{Z}_n^*$. But by Lemma 8.10.6, $k\mathbb{Z}_n^*$ and $\mathbb{Z}_n^*$ are the same size, so they must be equal. ■

Proof. (of Euler's Theorem 8.10.3 for $\mathbb{Z}_n$)

Let

$$P ::= k_1 \cdot k_2 \cdots k_{\phi(n)}(\mathbb{Z}_n)$$

be the product in $\mathbb{Z}_n$ of all the numbers in $\mathbb{Z}_n^*$. Let

$$Q ::= (k \cdot k_1) \cdot (k \cdot k_2) \cdots (k \cdot k_{\phi(n)})(\mathbb{Z}_n)$$

for some $k \in \mathbb{Z}_n^*$. Factoring out k 's immediately gives

$$Q = k^{\phi(n)} P(\mathbb{Z}_n).$$

But Q is the same as the product of the numbers in $k\mathbb{Z}_n^*$, and $k\mathbb{Z}_n^* = \mathbb{Z}_n^*$, so we realize that Q is the product of the same numbers as P, just in a different order. Altogether, we have

$$P = Q = k^{\phi(n)} P(\mathbb{Z}_n).$$

Furthermore, $P \in \mathbb{Z}_n^*$ by Lemma 8.10.4, and so it can be cancelled from both sides of this equality, giving

$$1 = k^{\phi(n)} (\mathbb{Z}_n).$$

Euler's theorem offers another way to find inverses modulo n: if k is relatively prime to n, then $k^{\phi(n)-1}$ is a $\mathbb{Z}_n$-inverse of k, and we can compute this power of k efficiently using fast exponentiation. However, this approach requires computing $\phi(n)$. In the next section, we'll show that computing $\phi(n)$ is easy *if* we know the prime factorization of n. But we know that finding the factors of n is generally hard to do when n is large, and so the Pulverizer remains the best approach to computing inverses modulo n.

Fermat's Little Theorem

For the record, we mention a famous special case of Euler's Theorem that was known to Fermat a century earlier.

Corollary 8.10.8 (*Fermat's Little Theorem*). *Suppose p is a prime and k is not a multiple of p. Then:*

$$k^{p-1} \equiv 1 \pmod{p}$$

Computing Euler's ϕ Function

RSA works using arithmetic modulo the product of two large primes, so we begin with an elementary explanation of how to compute $\phi(pq)$ for primes p and q:

Lemma 8.10.9.

$$\phi(pq) = (p-1)(q-1)$$

for primes $p \neq q$.

Proof. Since p and q are prime, any number that is not relatively prime to pq must be a multiple of p or a multiple of q. Among the pq numbers in $[0..pq)$, there are precisely q multiples of p and p multiples of q. Since p and q are relatively prime, the only number in $[0..pq)$ that is a multiple of both p and q is 0. Hence, there are $p+q-1$ numbers in $[0..pq)$ that are not relatively prime to n. This means that

$$\begin{aligned} \phi(pq) &= pq - (p+q-1) \\ &= (p-1)(q-1). \end{aligned} \tag{8.10.5}$$

as claimed.[14] ■

The following theorem provides a way to calculate $\phi(n)$ for arbitrary n.

Theorem 8.10.10

(a) If p is a prime, then $\phi(p^k) = p^k - p^{k-1}$ for $k \geq 1$.
(b) If a and b are relatively prime, then $\phi(ab) = \phi(a)\phi(b)$.

Here's an example of using Theorem 8.10.10. to compute $\phi(300)$:

$$\begin{aligned} \phi(300) &= \phi\left(2^2 \cdot 3 \cdot 5^2\right) \\ &= \phi\left(2^2\right) \cdot \phi(3) \cdot \phi\left(5^2\right) && \text{(by Theorem 8.10.10.(b))} \\ &= \left(2^2 - 2^1\right)\left(3^1 - 3^0\right)\left(5^2 - 5^1\right) && \text{(by Theorem 8.10.10.(a))} \\ &= 80. \end{aligned} \tag{8.10.6}$$

Note that Lemma 8.10.9 also follows as a special case of Theorem 8.10.10.(b), since we know that $\phi(p) = p-1$ for any prime, p.

To prove Theorem 8.10.10.(a), notice that every pth number among the p^k numbers in $[0..p^k)$ is divisible by p, and only these are divisible by p. So $1/p$ of these numbers are divisible by p and the remaining ones are not. That is,

$$\phi(p^k) = p^k - (1/p)p^k = p^k - p^{k-1}.$$

We'll leave a proof of Theorem 8.10.10.(b) to Problem 8.62.

As a consequence of Theorem 8.10.10, we have

Corollary 8.10.11. *For any number n, if $p_1, p_2, \ldots, p_j$ are the (distinct) prime factors of n, then*

$$\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2}) \cdots (1 - \frac{1}{p_j}).$$

We'll give another proof of Corollary 8.10.11 based on rules for counting in Section 14.9.5.

[11]Since 0 is not relatively prime to anything, $\phi(n)$ could equivalently be defined using the interval $(0..n)$ instead of $[0..n)$.

[12]Some texts call it Euler's *totient function*.

[13]Some other texts use the notation n^* for $\mathbb{Z}_n^*$.

[14]This proof previews a kind of counting argument that we will explore more fully in Part III.

This page titled 8.10: Euler's Theorem is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.11: RSA Public Key Encryption

Turing's code did not work as he hoped. However, his essential idea—using number theory as the basis for cryptography—succeeded spectacularly in the decades after his death.

In 1977, Ronald Rivest, Adi Shamir, and Leonard Adleman at MIT proposed a highly secure cryptosystem, called **RSA**, based on number theory. The purpose of the RSA scheme is to transmit secret messages over public communication channels. As with Turing's codes, the messages transmitted are nonnegative integers of some fixed size.

Moreover, RSA has a major advantage over traditional codes: the sender and receiver of an encrypted message need not meet beforehand to agree on a secret key. Rather, the receiver has both a *private key*, which they guard closely, and a *public key*, which they distribute as widely as possible. A sender wishing to transmit a secret message to the receiver encrypts their message using the receiver's widelydistributed public key. The receiver can then decrypt the received message using their closely held private key. The use of such a *public key cryptography* system allows you and Amazon, for example, to engage in a secure transaction without meeting up beforehand in a dark alley to exchange a key.

Interestingly, RSA does not operate modulo a prime, as Turing's hypothetical Version 2.0 may have, but rather modulo the product of *two* large primes—typically primes that are hundreds of digits long. Also, instead of encrypting by multiplication with a secret key, RSA exponentiates to a secret power—which is why Euler's Theorem is central to understanding RSA.

The scheme for RSA public key encryption appears in the box.

If the message m is relatively prime to n, then a simple application of Euler's Theorem implies that this way of decoding the encrypted message indeed reproduces the original unencrypted message. In fact, the decoding always works—even in (the highly unlikely) case that m is not relatively prime to n. The details are worked out in Problem 8.81.

RSA Cryptosystem

A **Receiver** who wants to be able to receive secret numerical messages creates a *private key*, which they keep secret, and a *public key*, which they make publicly available. Anyone with the public key can then be a **Sender** who can publicly send secret messages to the **Receiver**—even if they have never communicated or shared any information besides the public key.

Here is how they do it:

Beforehand The **Receiver** creates a public key and a private key as follows.

1. Generate two distinct primes, p and q. These are used to generate the private key, and they must be kept hidden. (In current practice, p and q are chosen to be hundreds of digits long.)
2. Let $n ::= pq$.
3. Select an integer $e \in [0..n)$ such that $\gcd(e,(p-1)(q-1)) = 1$. The *public key* is the pair (e,n). This should be distributed widely.
4. Let the *private key* $d \in [0..n)$ be the inverse of e in the ring $\mathbb{Z}_{(p-1)(q-1)}$. This private key can be found using the Pulverizer. The private key d should be kept hidden!

Encoding To transmit a message $m \in [0..n)$ to **Receiver**, a **Sender** uses the public key to encrypt m into a numerical message

$$\hat{m} ::= m^e(\mathbb{Z}_n).$$

The **Sender** can then publicly transmit $\hat{m}$ to the **Receiver**.

Decoding The **Receiver** decrypts message $\hat{m}$ back to message m using the private key:

$$m = \hat{m}^d(\mathbb{Z}_n).$$

Why is RSA thought to be secure? It would be easy to figure out the private key d if you knew p and q—you could do it the same way the **Receiver** does using the Pulverizer. But assuming the conjecture that it is hopelessly hard to factor a number that is the product of two primes with hundreds of digits, an effort to factor n is not going to break RSA.

Could there be another approach to reverse engineer the private key d from the public key that did not involve factoring n? Not really. It turns out that given just the private and the public keys, it is easy to factor n [15] (a proof of this is sketched in Problem

8.83). So if we are confident that factoring is hopelessly hard, then we can be equally confident that finding the private key just from the public key will be hopeless.

But even if we are confident that an RSA private key won't be found, this doesn't rule out the possibility of decoding RSA messages in a way that sidesteps the private key. It is an important unproven conjecture in cryptography that *any* way of cracking RSA—not just by finding the secret key—would imply the ability to factor. This would be a much stronger theoretical assurance of RSA security than is presently known.

But the real reason for confidence is that RSA has withstood all attacks by the world's most sophisticated cryptographers for nearly 40 years. Despite decades of these attacks, no significant weakness has been found. That's why the mathematical, financial, and intelligence communities are betting the family jewels on the security of RSA encryption.

You can hope that with more studying of number theory, you will be the first to figure out how to do factoring quickly and, among other things, break RSA. But be further warned that even Gauss worked on factoring for years without a lot to show for his efforts —and if you do figure it out, you might wind up meeting some humorless fellows working for a Federal agency in charge of security. . . .

[15]In practice, for this reason, the public and private keys should be randomly chosen so that neither is "too small".

This page titled 8.11: RSA Public Key Encryption is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

8.12: What has SAT got to do with it?

So why does society, or at least everybody's secret codes, fall apart if there is an efficient **test for satisfiability (SAT)**, as we claimed in Section 3.5? To explain this, remember that RSA can be managed computationally because multiplication of two primes is fast, but factoring a product of two primes seems to be overwhelmingly demanding.

Let's begin with the observation from Section 3.2 that a digital circuit can be described by a bunch of propositional formulas of about the same total size as the circuit. So testing circuits for satisfiability is equivalent to the SAT problem for propositional formulas (see Problem 3.18).

Now designing digital multiplication circuits is completely routine. We can easily build a digital "product checker" circuit out of AND, OR, and NOT gates with 1 output wire and $4n$ digital input wires. The first n inputs are for the binary representation of an integer i, the next n inputs for the binary representation of an integer j, and the remaining $2n$ inputs for the binary representation of an integer k. The output of the circuit is 1 iff $ij = k$ and $i, j > 1$. A straightforward design for such a product checker uses proportional to n^2 gates.

Now here's how to factor any number m with a length $2n$ binary representation using a SAT solver. First, fix the last $2n$ digital inputs—the ones for the binary representation of k—so that k equals m.

Next, set the first of the n digital inputs for the representation of i to be 1. Do a SAT test to see if there is a satisfying assignment of values for the remaining $2n - 1$ inputs used for the i and j representations. That is, see if the remaining inputs for i and j can be filled in to cause the circuit to give output 1. If there is such an assignment, fix the first i-input to be 1, otherwise fix it to be 0. So now we have set the first i-input equal to the first digit of the binary representations of an i such that $ij = m$.

Now do the same thing to fix the second of the n digital inputs for the representation of i, and then third, proceeding in this way through all the n inputs for the number i. At this point, we have the complete n-bit binary representation of an $i > 1$ such $ij = m$ for some $j > 1$. In other words, we have found an integer i that is a factor of m. We can now find j by dividing m by i.

So after n SAT tests, we have factored m. This means that if SAT for digital circuits with $4n$ inputs and about n^2 gates could be determined by a procedure taking a number of steps bounded above by a degree d polynomial in n, then $2n$ digit numbers can be factored in n times this many steps, that is, with a number of steps bounded by a polynomial of degree $d + 1$ in n. So if SAT could be solved in polynomial time, then so could factoring, and consequently RSA would be "easy" to break.

This page titled 8.12: What has SAT got to do with it? is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

9: Directed graphs and Partial Orders

Directed graphs, called *digraphs* for short, provide a handy way to represent how things are connected together and how to get from one thing to another by following those connections. They are usually pictured as a bunch of dots or circles with arrows between some of the dots, as in Figure 9.1. The dots are called *nodes* or *vertices* and the lines are called *directed edges* or *arrows*; the digraph in Figure 9.1 has 4 nodes and 6 directed edges.

Digraphs appear everywhere in computer science. For example, the digraph in Figure 9.2 represents a communication net, a topic we'll explore in depth in Chapter 10. Figure 9.2 has three "in" nodes (pictured as little squares) representing locations where packets may arrive at the net, the three "out" nodes representing destination locations for packets, and the remaining six nodes (pictured with little circles) represent switches. The 16 edges indicate paths that packets can take through the router.

Another place digraphs emerge in computer science is in the hyperlink structure of the World Wide Web. Letting the vertices $x_1, \cdots, x_n$ correspond to web pages, and using arrows to indicate when one page has a hyperlink to another, results in a digraph like the one in Figure 9.3—although the graph of the real World Wide Web would have n be a number in the billions and probably even the trillions. At first glance, this graph wouldn't seem to be very interesting. But in 1995, two students at Stanford, Larry Page and Sergey Brin, ultimately became multibillionaires from the realization of how useful the structure of this graph could be in building a search engine. So pay attention to graph theory, and who knows what might happen!

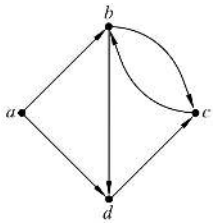


Figure 9.1 A 4-node directed graph with 6 edges.

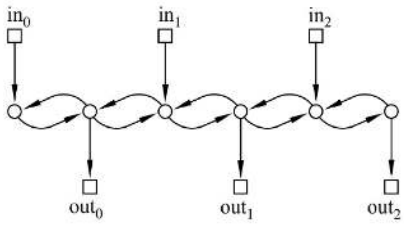


Figure 9.2 A 6-switch packet routing digraph.

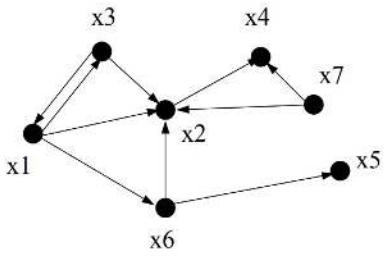


Figure 9.3 Links among Web Pages.

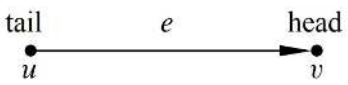


Figure 9.4 A directed edge $e = \langle u \to v \rangle$. The edge e starts at the tail vertex, u, and ends at the head vertex, v.

Definition 9.0.1.

A *directed graph*, G, consists of a nonempty set, $V(G)$, called the *vertices* of G, and a set, $E(G)$, called the *edges* of G. An element of $V(G)$ is called a *vertex*. A vertex is also called a *node*; the words "vertex" and "node" are used interchangeably. An element of $E(G)$ is called a *directed edge*. A directed edge is also called an "arrow" or simply an "edge." A directed edge *starts* at some vertex, u, called the *tail* of the edge, and *ends* at some vertex, v, called the *head* of the edge, as in Figure 9.4. Such an edge can be represented by the ordered pair (u, v). The notation $\langle u \to v \rangle$ denotes this edge.

There is nothing new in Definition 9.0.1 except for a lot of vocabulary. Formally, a digraph G is the same as a binary relation on the set, $V = V(G)$—that is, a digraph is just a binary relation whose domain and codomain are the same set, V. In fact, we've already referred to the arrows in a relation G as the "graph" of G. For example, the divisibility relation on the integers in the interval $[1..12]$ could be pictured by the digraph in Figure 9.5.

9.1: Vertex Degrees
9.2: Walks and Paths
9.3: Adjacency Matrices
9.4: Walk Relations
9.5: Directed Acyclic Graphs and Scheduling
9.6: Partial Orders
9.7: Representing Partial Orders by Set Containment
9.8: Linear Orders
9.9: Product Orders
9.10: Equivalence Relations
9.11: Summary of Relational Properties

This page titled 9: Directed graphs and Partial Orders is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.1: Vertex Degrees

The *in-degree* of a vertex in a digraph is the number of arrows coming into it, and similarly its *out-degree* is the number of arrows out of it. More precisely,

Definition 9.1.1

If G is a digraph and $v \in V(G)$, then

$$\text{indeg}(v) ::= |\{e \in E(G) \mid \text{head}(e) = v\}|$$

$$\text{outdeg}(v) ::= |\{e \in E(G) \mid \text{tail}(e) = v\}|$$

An immediate consequence of this definition is

Lemma 9.1.2.

$$\sum_{v \in V(G)} \text{indeg}(v) = \sum_{v \in V(G)} \text{outdeg}(v)$$

Proof. Both sums are equal to $|E(G)|$. ■

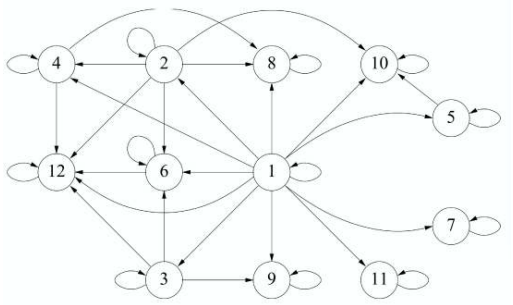


Figure 9.5 The Digraph for Divisibility on $\{1, 2, \ldots, 12\}$.

This page titled 9.1: Vertex Degrees is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.2: Walks and Paths

Picturing digraphs with points and arrows makes it natural to talk about following successive edges through the graph. For example, in the digraph of Figure 9.5, you might start at vertex 1, successively follow the edges from vertex 1 to vertex 2, from 2 to 4, from 4 to 12, and then from 12 to 12 twice (or as many times as you like). The sequence of edges followed in this way is called a *walk* through the graph. A *path* is a walk which never visits a vertex more than once. So following edges from 1 to 2 to 4 to 12 is a path, but it stops being a path if you go to 12 again.

The natural way to represent a walk is with the sequence of successive vertices it went through, in this case:

$$1\ 2\ 4\ 12\ 12\ 12.$$

However, it is conventional to represent a walk by an alternating sequence of successive vertices and edges, so this walk would formally be

$$1\langle 1 \to 2\rangle 2\langle 2 \to 4\rangle 4\langle 4 \to 12\rangle 12\langle 12 \to 12\rangle 12\langle 12 \to 12\rangle 12. \tag{9.2.1}$$

The redundancy of this definition is enough to make any computer scientist cringe, but it does make it easy to talk about how many times vertices and edges occur on the walk. Here is a formal definition:

Definition 9.2.1.

A *walk in a digraph*, G, is an alternating sequence of vertices and edges that begins with a vertex, ends with a vertex, and such that for every edge $\langle u \to v\rangle$ in the walk, vertex u is the element just before the edge, and vertex v is the next element after the edge.

So a walk, $\mathbf{v}$, is a sequence of the form

$$\mathbf{v} ::= v_0\langle v_0 \to v_1\rangle v_1\langle v_1 \to v_2\rangle v_2 \ldots \langle v_{k-1} \to v_k\rangle v_k$$

where $v_i \to v_{i+1} \in E(G)$ for $i \in [0..k)$. The walk is said to *start* at v_0, to *end* at v_k, and the *length*, $|\mathbf{v}|$, of the walk is defined to be k.

The walk is a *path* iff all the v_i's are different, that is, if $i \neq j$, then $v_i \neq v_j$.

A *closed walk* is a walk that begins and ends at the same vertex. A *cycle* is a positive length closed walk whose vertices are distinct except for the beginning and end vertices.

Note that a single vertex counts as a length zero path that begins and ends at itself. It also is a closed walk, but does not count as a cycle, since cycles by definition must have positive length. Length one cycles are possible when a node has an arrow leading back to itself. The graph in Figure 9.1 has none, but every vertex in the divisibility relation digraph of Figure 9.5 is in a length one cycle. Length one cycles are sometimes called *self-loops*.

Although a walk is officially an alternating sequence of vertices and edges, it is completely determined just by the sequence of successive vertices on it, or by the sequence of edges on it. We will describe walks in these ways whenever it's convenient. For example, for the graph in Figure 9.1,

- (a, b, d), or simply abd, is a (vertex-sequence description of a) length two path,
- $(\langle a \to b\rangle, \langle b \to d\rangle)$, or simply $\langle a \to b\rangle\langle b \to d\rangle$, is (an edge-sequence description of) the same length two path,
- $abcbd$ is a length four walk,
- $dcbcbd$ is a length five closed walk,
- $bdcb$ is a length three cycle,
- $\langle b \to c\rangle\langle c \to b\rangle$ is a length two cycle, and
- $\langle c \to b\rangle\langle b \to a\rangle\langle a \to d\rangle$ is *not* a walk. A walk is not allowed to follow edges in the wrong direction.

If you walk for a while, stop for a rest at some vertex, and then continue walking, you have broken a walk into two parts. For example, stopping to rest after following two edges in the walk (9.1) through the divisibility graph breaks the walk into the first part of the walk

$$1\langle 1 \to 2\rangle 2\langle 2 \to 4\rangle 4 \tag{9.2.2}$$

from 1 to 4, and the rest of the walk

$$4\langle 4 \to 12\rangle 12\langle 12 \to 12\rangle 12\langle 12 \to 12\rangle 12. \tag{9.2.3}$$

from 4 to 12, and we'll say the whole walk (9.1) is the merge of the walks (9.2) and (9.3). In general, if a walk $\mathbf{f}$ ends with a vertex, v, and a walk $\mathbf{r}$ starts with the same vertex, v, we'll say that their *merge*, $\mathbf{f}\hat{\mathbf{r}}$, is the walk that starts with $\mathbf{f}$ and continues with $\mathbf{r}$. [1] Two walks can only be merged if the first ends with the same vertex, v, that the second one starts with. Sometimes it's useful to name the node v where the walks merge; we'll use the notation $\mathbf{f}\hat{v}\mathbf{r}$ to describe the merge of a walk $\mathbf{f}$ that ends at v with a walk $\mathbf{r}$ that begins at v.

A consequence of this definition is that

Lemma 9.2.2.

$$|\mathbf{f}\hat{\mathbf{r}}| = |\mathbf{f}| + |\mathbf{r}|$$

In the next section we'll get mileage out of walking this way.

Finding a Path

If you were trying to walk somewhere quickly, you'd know you were in trouble if you came to the same place twice. This is actually a basic theorem of graph theory.

Theorem 9.2.3

The shortest walk from one vertex to another is a path.

Proof. If there is a walk from vertex u to another vertex $v \neq u$, then by the Well Ordering Principle, there must be a minimum length walk $\mathbf{w}$ from u to v. We claim $\mathbf{w}$ is a path.

To prove the claim, suppose to the contrary that $\mathbf{w}$ is not a path, meaning that some vertex x occurs twice on this walk. That is,

$$\mathbf{w} = \mathbf{e}\hat{x}\mathbf{f}\hat{x}\mathbf{g}$$

for some walks $\mathbf{e}, \mathbf{f}, \mathbf{g}$ where the length of $\mathbf{f}$ is positive. But then "deleting" $\mathbf{f}$ yields a strictly shorter walk

$$\mathbf{e}\hat{x}\mathbf{g}$$

from u to v, contradicting the minimality b of $\mathbf{w}$. ■

Definition 9.2.4

The *distance*, $\mathbf{dist}(u, v)$, in a graph from vertex u to vertex v is the length of a shortest path from u to v.

As would be expected, this definition of distance satisfies:

Lemma 9.2.5. *[The Triangle Inequality]*

$$\mathbf{dist}(u, v) \leq \mathbf{dist}(u, x) + \mathbf{dist}(x, v)$$

for all vertices u, v, x with equality holding iff x is on a shortest path from u to v.

Of course, you might expect this property to be true, but distance has a technical definition and its properties can't be taken for granted. For example, unlike ordinary distance in space, the distance from u to v is typically different from the distance from v to u. So, let's prove the Triangle Inequality

Proof. To prove the inequality, suppose $\mathbf{f}$ is a shortest path from u to x and $\mathbf{r}$ is a shortest path from x to v. Then by Lemma 9.2.2, $\mathbf{f}\hat{x}\mathbf{r}$ is a walk of length $\mathbf{dist}(u, x) + \mathbf{dist}(x, v)$ from u to v, so this sum is an upper bound on the length of the shortest path from u to v by Theorem 9.2.3.

Proof of the "iff" is in Problem 9.3. ■

Finally, the relationship between walks and paths extends to closed walks and cycles:

Lemma 9.2.6. *The shortest positive length closed walk through a vertex is a cycle through that vertex.*

The proof of Lemma 9.2.6 is essentially the same as for Theorem 9.2.3; see Problem 9.7.

[1]It's tempting to say the *merge* is the concatenation of the two walks, but that wouldn't quite be right because if the walks were concatenated, the vertex v would appear twice in a row where the walks meet.

This page titled 9.2: Walks and Paths is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.3: Adjacency Matrices

If a graph, G, has n vertices, $v_0, v_1, \cdots, v_{n-1}$, a useful way to represent it is with an $n \times n$ matrix of zeroes and ones called its *adjacency matrix*, A_G. The ijth entry of the adjacency matrix, $(A_G)_{ij}$, is 1 if there is an edge from vertex v_i to vertex v_j and 0 otherwise. That is,

$$(A_G)_{ij} ::= \begin{cases} 1 & \text{if } \langle v_i \to v_j \rangle \in E(G), \\ 0 & \text{otherwise.} \end{cases}$$

For example, let H be the 4-node graph shown in Figure 9.1. Its adjacency matrix, A_H, is the 4×4 matrix:

$$A_H = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 1 & 0 & 1 \\ b & 0 & 0 & 1 & 1 \\ c & 0 & 1 & 0 & 0 \\ d & 0 & 0 & 1 & 0 \end{array}$$

A payoff of this representation is that we can use matrix powers to count numbers of walks between vertices. For example, there are two length two walks between vertices a and c in the graph H:

$$a\langle a \to b\rangle b\langle b \to c\rangle c$$

$$a\langle a \to d\rangle d\langle d \to c\rangle c$$

and these are the only length two walks from a to c. Also, there is exactly one length two walk from b to c and exactly one length two walk from c to c and from d to b, and these are the only length two walks in H. It turns out we could have read these counts from the entries in the matrix $(A_H)^2$:

$$(A_H)^2 = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 0 & 2 & 1 \\ b & 0 & 1 & 1 & 0 \\ c & 0 & 0 & 1 & 1 \\ d & 0 & 1 & 0 & 0 \end{array}$$

More generally, the matrix $(A_G)^k$: provides a count of the number of length k walks between vertices in any digraph, G, as we'll now explain.

Definition 9.3.1

The length-k walk counting matrix for an n-vertex graph G is the $n \times n$ matrix C such that

$$C_{uv} ::= \text{the number of length-k walks from } u \text{ to } v. \tag{9.3.1}$$

Notice that the adjacency matrix A_G is the length-1 walk counting matrix for G, and that $(A_G)^0$, which by convention is the identity matrix, is the length-0 walk counting matrix.

Theorem 9.3.2

If C is the length-k walk counting matrix for a graph G, and D is the length-m walk counting matrix, then CD is the length $k + m$ walk counting matrix for G.

According to this theorem, the square $(A_G)^2$ of the adjacency matrix is the length two walk counting matrix for G. Applying the theorem again to $(A_G)^2 A_G$ shows that the length-3 walk counting matrix is $(A_G)^3$. More generally, it follows by induction that

Corollary 9.3.3. The length-k counting matrix of a digraph, G, is $(A_G)^k$, for all $k \in \mathbb{N}$.

In other words, you can determine the number of length k walks between any pair of vertices simply by computing the kth power of the adjacency matrix!

That may seem amazing, but the proof uncovers this simple relationship between matrix multiplication and numbers of walks.

Proof of Theorem 9.3.2. Any length $(k+m)$ walk between vertices u and v begins with a length k walk starting at u and ending at some vertex, w, followed by a length m walk starting at w and ending at v. So the number of length $(k+m)$ walks from u to v that go through w at the kth step equals the number C_{uw} of length k walks from u to w, times the number D_{wv} of length m walks from w to v. We can get the total number of length $k+m$ walks from u to v by summing, over all possible vertices w, the number of such walks that go through w at the kth step. In other words,

$$\#\text{length } (k+m) \text{ walks from } u \text{ to } v = \sum_{w \in V(G)} C_{uw} \cdot D_{wv} \tag{9.3.2}$$

But the right hand side of (9.3.2) is precisely the definition of $(CD)_{uv}$. Thus, CD is indeed the length-$(k+m)$ walk counting matrix.

Shortest Paths

The relation between powers of the adjacency matrix and numbers of walks is cool—to us math nerds at least—but a much more important problem is finding shortest paths between pairs of nodes. For example, when you drive home for vacation, you generally want to take the shortest-time route.

One simple way to find the lengths of all the shortest paths in an n-vertex graph, G, is to compute the successive powers of A_G one by one up to the $n-1$st, watching for the first power at which each entry becomes positive. That's because Theorem 9.3.2 implies that the length of the shortest path, if any, between u and v, that is, the distance from u to v, will be the smallest value k for which $(A_G)^k_{uv}$ is nonzero, and if there is a shortest path, its length will be $\leq n-1$. Refinements of this idea lead to methods that find shortest paths in reasonably efficient ways. The methods apply as well to weighted graphs, where edges are labelled with weights or costs and the objective is to find least weight, cheapest paths. These refinements are typically covered in introductory algorithm courses, and we won't go into them any further.

This page titled 9.3: Adjacency Matrices is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.4: Walk Relations

A basic question about a digraph is whether there is a way to get from one particular vertex to another. So for any digraph, G, we are interested in a binary relation, G^*, called the *walk relation* on $V(G)$ where

$$uG^*v ::= \text{ there is a walk in } G \text{ from } u \text{ to } v. \tag{9.4.1}$$

Similarly, there is a *positive walk relation*

$$uG^+v ::= \text{ there is a positive length walk in } G \text{ from } u \text{ to } v. \tag{9.4.2}$$

Definition 9.4.1

When there is a walk from vertex v to vertex w, we say that w is *reachable* from v, or equivalently, that v is *connected* to w.

Composition of Relations

There is a simple way to extend composition of functions to composition of relations, and this gives another way to talk about walks and paths in digraphs.

Definition 9.4.2

Let $R : B \to C$ and $S : A \to B$ be binary relations. Then the composition of R with S is the binary relation $(R \circ S) : A \to C$ defined by the rule

$$a \circ (R \circ S)c ::= \exists b \in B.\, (aSb) \text{ AND } (bRc). \tag{9.4.3}$$

This agrees with the Definition 4.3.1 of composition in the special case when R and S are functions.[2]

Remembering that a digraph is a binary relation on its vertices, it makes sense to compose a digraph G with itself. Then if we let G^n denote the composition of G with itself n times, it's easy to check (see Problem 9.9) that G^n is the *length-n walk relation*:

$$aG^nb \text{ iff there is a length } n \text{ walk in } G \text{ from } a \text{ to } b.$$

This even works for $n = 0$, with the usual convention that G^0 is the *identity relation* $Id_{V(G)}$ on the set of vertices.[3] Since there is a walk iff there is a path, and every path is of length at most $|V(G)| - 1$, we now have[4]

$$G^* = G^0 \cup G^1 \cup G^2 \cup \ldots \cup G^{|V(G)-1|} = (G \cup G^0)^{|V(G)|-1}. \tag{9.4.4}$$

The final equality points to the use of repeated squaring as a way to compute G^* with $\log n$ rather than $n - 1$ compositions of relations.

[2]The reversal of the order of R and S in (9.8) is not a typo. This is so that relational composition generalizes function composition. The value of function f composed with function g at an argument, x, is $f(g(x))$. So in the composition, $f \circ g$, the function g is applied first.

This page titled 9.4: Walk Relations is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.5: Directed Acyclic Graphs and Scheduling

Some of the prerequisites of MIT computer science subjects are shown in Figure 9.6. An edge going from subject s to subject t indicates that s is listed in the catalogue as a direct prerequisite of t. Of course, before you can take subject t, you have to take not only subject s, but also all the prerequisites of s, and any prerequisites of those prerequisites, and so on. We can state this precisely in terms of the positive walk relation: if D is the direct prerequisite relation on subjects, then subject u has to be completed before taking subject v iff uD^+v.

Of course it would take forever to graduate if this direct prerequisite graph had a positive length closed walk. We need to forbid such closed walks, which by Lemma 9.2.6 is the same as forbidding cycles. So, the direct prerequisite graph among subjects had better be *acyclic*:

Definition 9.5.1

A *directed acyclic graph (DAG)* is a directed graph with no cycles.

DAGs have particular importance in computer science. They capture key concepts used in analyzing task scheduling and concurrency control. When distributing a program across multiple processors, we're in trouble if one part of the program needs an output that another part hasn't generated yet! So let's examine DAGs and their connection to scheduling in more depth.

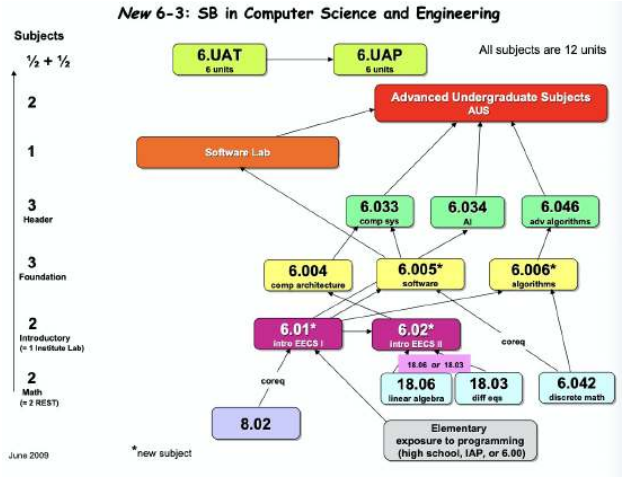


Figure 9.6 Subject prerequisites for MIT Computer Science (6-3) Majors.

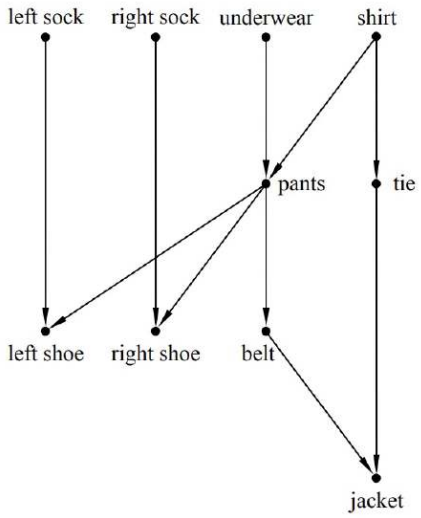


Figure 9.7 DAG describing which clothing items have to be put on before others.

Scheduling

In a scheduling problem, there is a set of tasks, along with a set of constraints specifying that starting certain tasks depends on other tasks being completed beforehand. We can map these sets to a digraph, with the tasks as the nodes and the direct prerequisite constraints as the edges.

For example, the DAG in Figure 9.7 describes how a man might get dressed for a formal occasion. As we describe above, vertices correspond to garments and the edges specify which garments have to be put on before which others.

When faced with a set of prerequisites like this one, the most basic task is finding an order in which to perform all the tasks, one at a time, while respecting the dependency constraints. Ordering tasks in this way is known as *topological sorting*.

Definition 9.5.2

A *topological sort* of a finite DAG is a list of all the vertices such that each vertex v appears earlier in the list than every other vertex reachable from v.

There are many ways to get dressed one item at a time while obeying the constraints of Figure 9.7. We have listed two such topological sorts in Figure 9.8.

(a)	(b)
underwear	left sock
shirt	shirt
pants	tie
belt	underwear
tie	right sock
jacket	pants
left sock	right shoe
right sock	belt
left shoe	jacket
right shoe	left shoe

Figure 9.8 Two possible topological sorts of the prerequisites described in Figure 9.7

In fact, we can prove that every finite DAG has a topological sort. You can think of this as a mathematical proof that you can indeed get dressed in the morning. Topological sorts for finite DAGs are easy to construct by starting from minimal elements:

Definition 9.5.3

An vertex v of a DAG, D, is *minimum* iff every other vertex is reachable from v.

A vertex v is *minimal* iff v is not reachable from any other vertex.

It can seem peculiar to use the words “minimum” and “minimal” to talk about vertices that start paths. These words come from the perspective that a vertex is “smaller” than any other vertex it connects to. We’ll explore this way of thinking about DAGs in the next section, but for now we’ll use these terms because they are conventional.

One peculiarity of this terminology is that a DAG may have no minimum element but lots of minimal elements. In particular, the clothing example has four minimal elements: leftsock, rightsock, underwear, and shirt.

To build an order for getting dressed, we pick one of these minimal elements— say, shirt. Now there is a new set of minimal elements; the three elements we didn’t chose as step 1 are still minimal, and once we have removed shirt, tie becomes minimal as well. We pick another minimal element, continuing in this way until all elements have been picked. The sequence of elements in the order they were picked will be a topological sort. This is how the topological sorts above were constructed.

So our construction shows:

Theorem 9.5.4

Every finite DAG has a topological sort.

There are many other ways of constructing topological sorts. For example, instead of starting from the minimal elements at the beginning of paths, we could build a topological sort starting from *maximal* elements at the end of paths. In fact, we could build a topological sort by picking vertices arbitrarily from a finite DAG and simply inserting them into the list wherever they will fit.[5]

Parallel Task

Scheduling For task dependencies, topological sorting provides a way to execute tasks one after another while respecting those dependencies. But what if we have the ability to execute more than one task at the same time? For example, say tasks are programs, the DAG indicates data dependence, and we have a parallel machine with lots of processors instead of a sequential machine with only one. How should we schedule the tasks? Our goal should be to minimize the total *time* to complete all the tasks. For simplicity, let's say all the tasks take the same amount of time and all the processors are identical.

So given a finite set of tasks, how long does it take to do them all in an optimal parallel schedule? We can use walk relations on acyclic graphs to analyze this problem.

In the first unit of time, we should do all minimal items, so we would put on our left sock, our right sock, our underwear, and our shirt.[6] In the second unit of time, we should put on our pants and our tie. Note that we cannot put on our left or right shoe yet, since we have not yet put on our pants. In the third unit of time, we should put on our left shoe, our right shoe, and our belt. Finally, in the last unit of time, we can put on our jacket. This schedule is illustrated in Figure 9.9.

The total time to do these tasks is 4 units. We cannot do better than 4 units of time because there is a sequence of 4 tasks that must each be done before the next. We have to put on a shirt before pants, pants before a belt, and a belt before a jacket. Such a sequence of items is known as a *chain*.

Definition 9.5.5

Two vertices in a DAG are *comparable* when one of them is reachable from the other. A *chain* in a DAG is a set of vertices such that any two of them are comparable. A vertex in a chain that is reachable from all other vertices in the chain is called a *maximum element* of the chain. A finite chain is said to *end at* its maximum element.

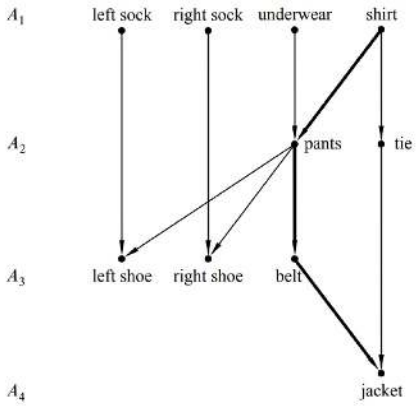


Figure 9.9 A parallel schedule for the tasks-getting-dressed digraph in Figure 9.7. The tasks in A_i can be performed in step i for $1 \leq i \leq 4$. A chain of 4 tasks (the critical path in this example) is shown with bold edges.

The time it takes to schedule tasks, even with an unlimited number of processors, is at least as large as the number of vertices in any chain. That's because if we used less time than the size of some chain, then two items from the chain would have to be done at the same step, contradicting the precedence constraints. For this reason, a *largest* chain is also known as a *critical path*. For example, Figure 9.9 shows the critical path for the getting-dressed digraph.

In this example, we were able to schedule all the tasks with t steps, where t is the size of the largest chain. A nice feature of DAGs is that this is always possible! In other words, for any DAG, there is a legal parallel schedule that runs in t total steps.

In general, a *schedule* for performing tasks specifies which tasks to do at successive steps. Every task, a, has to be scheduled at some step, and all the tasks that have to be completed before task a must be scheduled for an earlier step. Here's a rigorous definition of schedule.

Definition 9.5.6

A *partition* of a set A is a set of nonempty subsets of A called the *blocks*[7] of the partition, such that every element of A is in exactly one block.

For example, one possible partition of the set $\{a, b, c, d, e\}$ into three blocks is

$$\{a, c\} \quad \{b, e\} \quad \{d\}.$$

Definition 9.5.7

A *parallel schedule* for a DAG, D, is a partition of $V(D)$ into blocks $A_0, A_1, \ldots$, such that when $j < k$ no vertex in A_j is reachable from any vertex in A_k. The block A_k is called the set of elements *scheduled at step* k, and the *time* of the schedule is the number of blocks. The maximum number of elements scheduled at any step is called the *number of processors* required by the schedule.

A *largest* chain ending at an element a is called a *critical path* to a, and the number of elements less than a in the chain is called the *depth* of a. So in any possible parallel schedule, there must be at least depth (a) steps before task a can be started. In particular, the minimal elements are precisely the elements with depth 0.

There is a very simple schedule that completes every task in its minimum number of steps: just use a "greedy" strategy of performing tasks as soon as possible. Schedule all the elements of depth k at step k. That's how we found the above schedule for getting dressed.

Theorem 9.5.8

A minimum time schedule for a finite DAG D consists of the sets $A_0, A_1, \ldots$, where

$$A_k ::= \{a \in V(D) \mid \text{depth}(a) = k\}.$$

We'll leave to Problem 9.19 the proof that the sets A_k are a parallel schedule according to Definition 9.5.7. We can summarize the story above in this way: with an unlimited number of processors, the parallel time to complete all tasks is simply the size of a critical path:

Corollary 9.5.9. *Parallel time = size of critical path.*

Things get more complex when the number of processors is bounded; see Problem 9.20 for an example.

Dilworth's Lemma

Definition 9.5.10

An *antichain* in a DAG is a set of vertices such that *no* two elements in the set are comparable—no walk exists between any two different vertices in the set.

Our conclusions about scheduling also tell us something about antichains.

Corollary 9.5.11. *In a DAG, D, if the size of the largest chain is t, then $V(D)$ can be partitioned into t antichains.*

Proof. Let the antichains be the sets $A_k ::= \{a \in V(D) \mid \text{depth}(a) = k\}$. It is an easy exercise to verify that each A_k is an antichain (Problem 9.19). ■

Corollary 9.5.11 implies[8] a famous result about acyclic digraphs:

Lemma 9.5.12 (Dilworth). *For all $t > 0$, every DAG with n vertices must have either a chain of size greater than t or an antichain of size at least n/t.*

Proof. Assume that there is no chain of size greater than t. Let l be the size of the largest antichain. If we make a parallel schedule according to the proof of Corollary 9.5.11, we create a number of antichains equal to the size of the largest chain, which is less than or equal t. Each element belongs to exactly one antichain, none of which are larger than l. So the total number of elements at most l times t—that is, $lt \geq n$. Simple division implies that $l \geq n/t$ ■.

Corollary 9.5.13. *Every DAG with n vertices has a chain of size greater than $\sqrt{n}$ or an antichain of size at least $\sqrt{n}$.*

Proof. Set $t = \sqrt{n}$ in Lemma 9.5.12. ■

Example 9.5.14. When the man in our example is getting dressed, $n = 10$.

Try $t = 3$. There is a chain of size 4.

Try $t = 4$. There is no chain of size 5, but there is an antichain of size $4 \geq 10/4$.

[5]In fact, the DAG doesn't even need to be finite, but you'll be relieved to know that we have no need to go into this.

[6]Yes, we know that you can't actually put on both socks at once, but imagine you are being dressed by a bunch of robot processors and you are in a big hurry. Still not working for you? Ok, forget about the clothes and imagine they are programs with the precedence constraints shown in Figure 9.7.

[7]We think it would be nicer to call them the *parts* of the partition, but "blocks" is the standard terminology.

[8]Lemma 9.5.12 also follows from a more general result known as Dilworth's Theorem, which we will not discuss.

This page titled 9.5: Directed Acyclic Graphs and Scheduling is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.6: Partial Orders

After mapping the "direct prerequisite" relation onto a digraph, we were then able to use the tools for understanding computer scientists' graphs to make deductions about something as mundane as getting dressed. This may or may not have impressed you, but we can do better. In the introduction to this chapter, we mentioned a useful fact that bears repeating: any digraph is formally the same as a binary relation whose domain and codomain are its vertices. This means that *any* binary relation whose domain is the same as its codomain can be translated into a digraph! Talking about the edges of a binary relation or the image of a set under a digraph may seem odd at first, but doing so will allow us to draw important connections between different types of relations. For instance, we can apply Dilworth's lemma to the "direct prerequisite" relation for getting dressed, because the graph of that relation was a DAG.

But how can we tell if a binary relation is a DAG? And once we know that a relation is a DAG, what exactly can we conclude? In this section, we will abstract some of the properties that a binary relation might have, and use those properties to define classes of relations. In particular, we'll explain this section's title, *partial orders*.

The Properties of the Walk Relation in DAGs

To begin, let's talk about some features common to all digraphs. Since merging a walk from u to v with a walk from v to w gives a walk from u to w, both the walk and positive walk relations have a relational property called *transitivity*:

Definition 9.6.1

A binary relation, R, on a set, A, is *transitive* iff

$$(aRb \text{ AND } bRc) \text{ IMPLIES } aRc$$

for every $a, b, c \in A$.

So we have

Lemma 9.6.2. *For any digraph, G, the walk relations G^+ and G^* are transitive.*

Since there is a length zero walk from any vertex to itself, the walk relation has another relational property called *reflexivity*:

Definition 9.6.3

A binary relation, R, on a set, A, is *reflexive* iff aRa for all $a \in A$.

Now we have

Lemma 9.6.4. *For any digraph, G, the walk relation G^* is reflexive.*

We know that a digraph is a DAG iff it has no positive length closed walks. Since any vertex on a closed walk can serve as the beginning and end of the walk, saying a graph is a DAG is the same as saying that there is no positive length path from any vertex back to itself. This means that the positive walk relation of D^+ of a DAG has a relational property called *irreflexivity*.

Definition 9.6.5

A binary relation, R, on a set, A, is *irreflexive* iff

$$\text{NOT}(aRa)$$

for all $a \in A$.

So we have

Lemma 9.6.6. *R is a DAG iff R^+ is irreflexive.*

Strict Partial Orders

Here is where we begin to define interesting classes of relations:

Definition 9.6.7

A relation that is transitive and irreflexive is called a *strict partial order*.

A simple connection between strict partial orders and DAGs now follows from Lemma 9.6.6:

Theorem 9.6.8

A relation R is a strict partial order iff R is the positive walk relation of a DAG.

Strict partial orders come up in many situations which on the face of it have nothing to do with digraphs. For example, the less-than order, $<$, on numbers is a strict partial order:

- if $x < y$ and $y < z$ then $x < z$, so less-than is transitive, and
- $\text{NOT}(x < x)$, so less-than is irreflexive.

The proper containment relation $\subset$ is also a partial order:

- if $A \subset B$ and $B \subset C$ then $A \subset C$, so containment is transitive, and
- $\text{NOT}(A \subset A)$, so proper containment is irreflexive.

If there are two vertices that are reachable from each other, then there is a positive length closed walk that starts at one vertex, goes to the other, and then comes back. So DAGs are digraphs in which no two vertices are mutually reachable. This corresponds to a relational property called *asymmetry*.

Definition 9.6.9

A binary relation, R, on a set, A, is *asymmetric* iff

$$aRb \text{ IMPLIES } \text{NOT}(bRa)$$

for all $a, b \in A$.

So we can also characterize DAGs in terms of asymmetry:

Corollary 9.6.10. *A digraph D is a DAG iff D^+ is asymmetric.*

Corollary 9.6.10 and Theorem 9.6.8 combine to give

Corollary 9.6.11. *A binary relation R on a set A is a strict partial order iff it is transitive and asymmetric.*[9]

A strict partial order may be the positive walk relation of different DAGs. This raises the question of finding a DAG with the *smallest* number of edges that determines a given strict partial order. For *finite* strict partial orders, the smallest such DAG turns out to be unique and easy to find (see Problem 9.25).

Weak Partial Orders

The less-than-or-equal relation, $\leq$, is at least as familiar as the less-than strict partial order, and the ordinary containment relation, $\subseteq$, is even more common than the proper containment relation. These are examples of *weak partial orders*, which are just strict partial orders with the additional condition that every element is related to itself. To state this precisely, we have to relax the asymmetry property so it does not apply when a vertex is compared to itself; this relaxed property is called *antisymmetry*:

Definition 9.6.12

A binary relation, R, on a set A, is *antisymmetric* iff, for all $a \neq b \in A$,

$$aRb \text{ IMPLIES } \text{NOT}(bRa)$$

Now we can give an axiomatic definition of weak partial orders that parallels the definition of strict partial orders.[10]

Definition 9.6.13

A binary relation on a set is a *weak partial order* iff it is transitive, reflexive, and antisymmetric.

The following lemma gives another characterization of weak partial orders that follows directly from this definition.

Lemma 9.6.14. A relation R on a set, A, is a weak partial order iff there is a strict partial order, S, on A such that

$$aRb \text{ iff } (aSb \text{ OR } a = b),$$

for all $a, b \in A$.

Since a length zero walk goes from a vertex to itself, this lemma combined with Theorem 9.6.8 yields:

Corollary 9.6.15. *A relation is a weak partial order iff it is the walk relation of a DAG.*

For weak partial orders in general, we often write an ordering-style symbol like $\preceq$ or $\sqsubseteq$ instead of a letter symbol like R.[11] Likewise, we generally use $\prec$ or $\sqsubset$ to indicate a strict partial order.

Two more examples of partial orders are worth mentioning:

Example 9.6.16. Let A be some family of sets and define $aRb \text{ iff } a \supseteq b$. Then R is a strict partial order.

Example 9.6.17. The divisibility relation is a weak partial order on the nonnegative integers.

For practice with the definitions, you can check that two more examples are vacuously partial orders on a set D: the identity relation Id_D is a weak partial order, and the *empty relation*—the relation with no arrows—is a strict partial order.

[9]Some texts use this Corollary to define strict partial orders.

[10]Some authors define partial orders to be what we call weak partial orders, but we'll use the phrase "partial order" to mean either a weak or strict one.

[11]General relations are usually denoted by a letter like R instead of a cryptic squiggly symbol, so $\preceq$ is kind of like the musical performer/composer Prince, who redefined the spelling of his name to be his own squiggly symbol. A few years ago he gave up and went back to the spelling "Prince."

This page titled 9.6: Partial Orders is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.7: Representing Partial Orders by Set Containment

Axioms can be a great way to abstract and reason about important properties of objects, but it helps to have a clear picture of the things that satisfy the axioms. DAGs provide one way to picture partial orders, but it also can help to picture them in terms of other familiar mathematical objects. In this section, we'll show that every partial order can be pictured as a collection of sets related by containment. That is, every partial order has the "same shape" as such a collection. The technical word for "same shape" is "isomorphic."

Definition 9.7.1

A binary relation, R, on a set, A, is *isomorphic* to a relation, S, on a set B iff there is a relation-preserving bijection from A to B; that is, there is a bijection $f : A \to B$ such that for all $a, a' \in A$,

$$aRa' \text{ iff } f(a)Sf(a').$$

To picture a partial order, $\preceq$, on a set, A, as a collection of sets, we simply represent each element A by the set of elements that are $\preceq$ to that element, that is,

$$a \longleftrightarrow \{b \in A \mid b \preceq a\}.$$

For example, if $\preceq$ is the divisibility relation on the set of integers, $\{1, 3, 4, 6, 8, 12\}$, then we represent each of these integers by the set of integers in A that divides it. So

$$\begin{aligned} 1 &\longleftrightarrow \{1\} \\ 3 &\longleftrightarrow \{1, 3\} \\ 4 &\longleftrightarrow \{1, 4\} \\ 6 &\longleftrightarrow \{1, 3, 6\} \\ 8 &\longleftrightarrow \{1, 4, 8\} \\ 12 &\longleftrightarrow \{1, 3, 4, 6, 12\} \end{aligned}$$

So, the fact that $3|12$ corresponds to the fact that $\{1, 3\} \subseteq \{1, 3, 4, 6, 12\}$.

In this way we have completely captured the weak partial order $\preceq$ by the subset relation on the corresponding sets. Formally, we have

Lemma 9.7.2. *Let $\preceq$ be a weak partial order on a set, A. Then $\preceq$ is isomorphic to the subset relation, $\subseteq$, on the collection of inverse images under the $\preceq$ relation of elements $a \in A$.*

We leave the proof to Problem 9.29. Essentially the same construction shows that strict partial orders can be represented by sets under the proper subset relation, $\subset$ (Problem 9.30). To summarize:

Theorem 9.7.3

Every weak partial order, $\preceq$, is isomorphic to the subset relation, $\subseteq$, on a collection of sets.

Every strict partial order, $\prec$, is isomorphic to the proper subset relation, $\subset$, on a collection of sets.

This page titled 9.7: Representing Partial Orders by Set Containment is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.8: Linear Orders

The familiar order relations on numbers have an important additional property: given two different numbers, one will be bigger than the other. Partial orders with this property are said to be *linear orders*. You can think of a linear order as one where all the elements are lined up so that everyone knows exactly who is ahead and who is behind them in the line. [12]

Definition 9.8.1

Let R be a binary relation on a set, A, and let a, b be elements of A. Then a and b are *comparable* with respect to R iff $[aRb \text{ OR } bRa]$. A partial order for which every two different elements are comparable is called a *linear order*.

So $<$ and $\leq$ are linear orders on $\mathbb{R}$. On the other hand, the subset relation is *not* linear, since, for example, any two different finite sets of the same size will be incomparable under $\subseteq$. The prerequisite relation on Course 6 required subjects is also not linear because, for example, neither 8.01 nor 6.042 is a prerequisite of the other.

[12]Linear orders are often called “total” orders, but this terminology conflicts with the definition of “total relation,” and it regularly confuses students.

Being a linear order is a much stronger condition than being a partial order that is a total relation. For example, any weak partial order is a total relation but generally won’t be linear.

This page titled 9.8: Linear Orders is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.9: Product Orders

Taking the product of two relations is a useful way to construct new relations from old ones.

Definition 9.9.1

The product, $R_1 \times R_2$, of relations R_1 and R_2 is defined to be the relation with

$$\begin{aligned} \text{domain}(R_1 \times R_2) &::= \text{domain}(R_1) \times \text{domain}(R_2), \\ \text{codomain}(R_1 \times R_2) &::= \text{codomain}(R_1) \times \text{codomain}(R_2), \\ (a_1, a_2)(R_1 \times R_2)(b_1, b_2) &\text{ iff } [a_1 R_1 b_1 \text{ and } a_2 R_2 b_2] \end{aligned} \tag{9.9.1}$$

It follows directly from the definitions that products preserve the properties of transitivity, reflexivity, irreflexivity, and antisymmetry (see Problem 9.41). If R_1 and R_2 both have one of these properties, then so does $R_1 \times R_2$. This implies that if R_1 and R_2 are both partial orders, then so is $R_1 \times R_2$.

Example 9.9.2

Define a relation, Y, on age-height pairs of being younger *and* shorter. This is the relation on the set of $\text{pair}(y, h)$ where y is a nonnegative integer ≤ 2400 that we interpret as an age in months, and h is a nonnegative integer ≤ 120 describing height in inches. We define Y by the rule

$$(y_1, h_1) Y (y_2, h_2) \text{ iff } y_1 < y_2 \text{ AND } h_1 \leq h_2.$$

That is, Y is the product of the $\leq$-relation on ages and the $\leq$-relation on heights.

Since both ages and heights are ordered numerically, the age-height relation Y is a partial order. Now suppose we have a class of 101 students. Then we can apply Dilworth's lemma 9.5.12 to conclude that there is a chain of 11 students—that is, 11 students who get taller as they get older–or an antichain of 11 students—that is, 11 students who get taller as they get younger, which makes for an amusing in-class demo.

On the other hand, the property of being a linear order is not preserved. For example, the age-height relation Y is the product of two linear orders, but it is not linear: the age 240 months, height 68 inches pair, (240,68), and the pair (228,72) are incomparable under Y.

This page titled 9.9: Product Orders is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.10: Equivalence Relations

Definition 9.10.1

A relation is an *equivalence relation* if it is reflexive, symmetric, and transitive.

Congruence modulo n is an important example of an equivalence relation:

- It is reflexive because $x \equiv x \pmod{n}$.
- It is symmetric because $x \equiv y \pmod{n}$ implies $y \equiv x \pmod{n}$.
- It is transitive because $x \equiv y \pmod{n}$ and $y \equiv z \pmod{n}$ imply that $x \equiv z \pmod{n}$.

There is an even more well-known example of an equivalence relation: equality itself.

Any total function defines an equivalence relation on its domain:

Definition 9.10.2

If $f : A \to B$ is a total function, define a relation $\equiv_f$ by the rule:

$$a \equiv_f a' \text{ IFF } f(a) = f(a').$$

From its definition, $\equiv_f$ is reflexive, symmetric and transitive because these are properties of equality. That is, $\equiv_f$ is an equivalence relation. This observation gives another way to see that congruence modulo n is an equivalence relation: the Remainder Lemma 8.6.1 implies that congruence modulo n is the same as $\equiv_r$ where $r(a)$ is the remainder of a divided by n.

In fact, a relation is an equivalence relation iff it equals $\equiv_f$ for some total function f (see Problem 9.47). So equivalence relations could have been defined using Definition 9.10.2.

Equivalence Classes

Equivalence relations are closely related to partitions because the images of elements under an equivalence relation are the blocks of a partition.

Definition 9.10.3

Given an equivalence relation $R : A \to A$, the *equivalence class*, $[a]_R$, of an element $a \in A$ is the set of all elements of A related to a by R.

Namely,

$$[a]_R ::= \{x \in A \mid aRx\}.$$

In other words, $[a]_R$ is the image $R(a)$.

For example, suppose that $A = \mathbb{Z}$ and aRb means that $a \equiv b \pmod{5}$. Then

$$[7]_R = \{\ldots, -3, 2, 7, 12, 22, \ldots\}.$$

Notice that 7,12, 17, etc., all have the same equivalence class; that is, $[7]_R = [12]_R = [17]_R = \cdots$.

There is an exact correspondence between equivalence relations on A and partitions of A. Namely, given any partition of a set, being in the same block is obviously an equivalence relation. On the other hand we have:

Theorem 9.10.4

The equivalence classes of an equivalence relation on a set A are the blocks of a partition of A.

We'll leave the proof of Theorem 9.10.4 as a basic exercise in axiomatic reasoning (see Problem 9.46), but let's look at an example. The congruent-mod-5 relation partitions the integers into five equivalence classes:

$\{\ldots, -5, 0, 5, 10, 15, 20, \ldots\}$
$\{\ldots, -4, 1, 6, 11, 16, 21, \ldots\}$
$\{\ldots, -3, 2, 7, 12, 17, 22, \ldots\}$
$\{\ldots, -2, 3, 8, 13, 18, 23, \ldots\}$
$\{\ldots, -1, 4, 9, 14, 19, 24, \ldots\}$

In these terms, $x \equiv y \pmod 5$ is equivalent to the assertion that x and y are both in the same block of this partition. For example, $6 \equiv 16 \pmod 5$, because they're both in the second block, but $2 \not\equiv 9 \pmod 5$ because 2 is in the third block while 9 is in the last block.

In social terms, if "likes" were an equivalence relation, then everyone would be partitioned into cliques of friends who all like each other and no one else.

This page titled 9.10: Equivalence Relations is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

9.11: Summary of Relational Properties

A relation $R : A \to A$ is the same as a digraph with vertices A.

Reflexivity R is *reflexive* when

$$\forall x \in A.\, xRx.$$

Every vertex in R has a self-loop.

Irreflexivity R is *irreflexive* when

$$\text{NOT}[\exists x \in A.\, xRx].$$

There are no self-loops in R.

Symmetry R is *symmetric* when

$$\forall x, y \in A.\, xRy IMPLIES yRx.$$

If there is an edge from x to y in R, then there is an edge back from y to x as well.

Asymmetry R is *asymmetric* when

$$\forall x, y \in A.\, xRy IMPLIES NOT yRx.$$

There is at most one directed edge between any two vertices in R, and there are no self-loops.

Antisymmetry R is *antisymmetric* when

$$\forall x \neq y \in A.\, xRy IMPLIES NOT yRx.$$

Equivalently,

$$\forall x, y \in A.\, (xRy AND yRx) \text{ IMPLIES } x = y.$$

There is at most one directed edge between any two distinct vertices, but there may be self-loops.

Transitivity R is *transitive* when

$$\forall x, y, z \in A.\, (xRy AND yRz) \text{ IMPLIES } xRz.$$

If there is a positive length path from u to v, then there is an edge from u to v.

Linear R is *linear* when

$$\forall x \neq y \in A.\, (xRy OR yRx)$$

Given any two vertices in R, there is an edge in one direction or the other between them.

For any finite, nonempty set of vertices of R, there is a directed path going through exactly these vertices.

Strict Partial Order R is a *strict partial order* iff R is transitive and irreflexive iff R is transitive and asymmetric iff it is the positive length walk relation of a DAG.

Weak Partial Order R is a *weak partial order* iff R is transitive and anti-symmetric and reflexive iff R is the walk relation of a DAG.

Equivalence Relation R is an *equivalence relation* iff R is reflexive, symmetric and transitive iff R equals the in-the-same-block-relation for some partition of domain(R).

This page titled 9.11: Summary of Relational Properties is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

10: Communication Networks

Modeling communication networks is an important application of digraphs in computer science. In this such models, vertices represent computers, processors, and switches; edges will represent wires, fiber, or other transmission lines through which data flows. For some communication networks, like the internet, the corresponding graph is enormous and largely chaotic. Highly structured networks, by contrast, find application in telephone switching systems and the communication hardware inside parallel computers. In this chapter, we'll look at some of the nicest and most commonly used structured networks.

10.1: Complete Binary Tree
10.2: Routing Problems
10.3: Network Diameter
10.4: Switch Count
10.5: Network Latency
10.6: Congestion
10.7: 2-D Array
10.8: Butterfly
10.9: Beneš Network

This page titled 10: Communication Networks is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

10.1: Complete Binary Tree

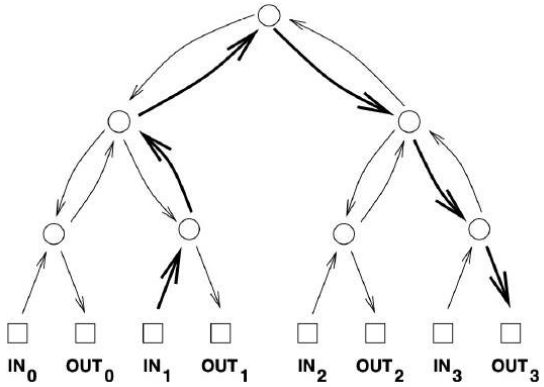
Let's start with a *complete binary tree*. Here is an example with 4 inputs and 4 outputs. The kinds of communication networks we consider aim to transmit packets of data between computers, processors, telephones, or other devices. The term *packet* refers to some roughly fixed-size quantity of data— 256 bytes or 4096 bytes or whatever. In this diagram and many that follow, the squares represent *terminals*, sources and destinations for packets of data. The circles represent *switches*, which direct packets through the network. A switch receives packets on incoming edges and relays them forward along the outgoing edges. Thus, you can imagine a data packet hopping through the network from an input terminal, through a sequence of switches joined by directed edges, to an output terminal.

Recall that there is a unique path between every pair of vertices in a tree. So, the natural way to route a packet of data from an input terminal to an output in the complete binary tree is along the corresponding directed path. For example, the route of a packet traveling from input 1 to output 3 is shown in bold.

This page titled 10.1: Complete Binary Tree is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

10.2: Routing Problems

Communication networks are supposed to get packets from inputs to outputs, with each packet entering the network at its own input switch and arriving at its own output switch. We're going to consider several different communication network designs, where each network has N inputs and N outputs; for convenience, we'll assume N is a power of two.



Which input is supposed to go where is specified by a permutation of $\{0, 1, \ldots, N-1\}$. So a permutation, π, defines a *routing problem*: get a packet that starts at input i to output $\pi(i)$. A *routing*, P, that solves a routing problem, π, is a set of paths from each input to its specified output. That is, P is a set of n paths, P_i, for $i = 0 \ldots, N-1$, where P_i goes from input i to output $\pi(i)$.

This page titled 10.2: Routing Problems is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

10.3: Network Diameter

The delay between the time that a packets arrives at an input and arrives at its designated output is a critical issue in communication networks. Generally, this delay is proportional to the length of the path a packet follows. Assuming it takes one time unit to travel across a wire, the delay of a packet will be the number of wires it crosses going from input to output.

Packets are usually routed from input to output by the shortest path possible. With a shortest-path routing, the worst-case delay is the distance between the input and output that are farthest apart. This is called the *diameter* of the network. In other words, the diameter of a network[1] is the maximum length of any shortest path between an input and an output. For example, in the complete binary tree above, the distance from input 1 to output 3 is six. No input and output are farther apart than this, so the diameter of this tree is also six.

More broadly, the diameter of a complete binary tree with N inputs and outputs is $2 \log N + 2$. This is quite good, because the logarithm function grows very slowly. We could connect up $2^{10} = 1024$ inputs and outputs using a complete binary tree and the worst input-output delay for any packet would be $2 \log(2^{10}) + 2 = 22$.

Switch Size

One way to reduce the diameter of a network is to use larger switches. For example, in the complete binary tree, most of the switches have three incoming edges and three outgoing edges, which makes them 3×3 switches. If we had 4×4 switches, then we could construct a complete *ternary* tree with an even smaller diameter. In principle, we could even connect up all the inputs and outputs via a single monster $N \times N$ switch.

This isn't very productive, however. Using an $N \times N$ switch would just conceal the original network design problem inside this abstract switch. Eventually, we'll have to design the internals of the monster switch using simpler components, and then we're right back where we started. So, the challenge in designing a communication network is figuring out how to get the functionality of an $N \times N$ switch using fixed size, elementary devices, like 3×3 switches.

[1]The usual definition of *diameter* for a general *graph* (simple or directed) is the largest distance between *any* two vertices, but in the context of a communication network we're only interested in the distance between inputs and outputs, not between arbitrary pairs of vertices.

This page titled 10.3: Network Diameter is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

10.4: Switch Count

Another goal in designing a communication network is to use as few switches as possible. The number of switches in a complete binary tree is $1+2+4+8+\cdots+N$, since there is 1 switch at the top (the "root switch"), 2 below it, 4 below those, and so forth. By the formula for geometric sums from Problem 5.4,

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1},$$

the total number of switches is $2N-1$, which is nearly the best possible with 3×3 switches.

This page titled 10.4: Switch Count is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

10.5: Network Latency

We'll sometimes be choosing routings through a network that optimize some quantity besides delay. For example, in the next section we'll be trying to minimize packet congestion. When we're not minimizing delay, shortest routings are not always the best, and in general, the delay of a packet will depend on how it is routed. For any routing, the most delayed packet will be the one that follows the longest path in the routing. The length of the longest path in a routing is called its *latency*.

The latency of a *network* depends on what's being optimized. It is measured by assuming that optimal routings are always chosen in getting inputs to their specified outputs. That is, for each routing problem, π, we choose an optimal routing that solves π. Then *network latency* is defined to be the largest routing latency among these optimal routings. Network latency will equal network diameter if routings are always chosen to optimize delay, but it may be significantly larger if routings are chosen to optimize something else.

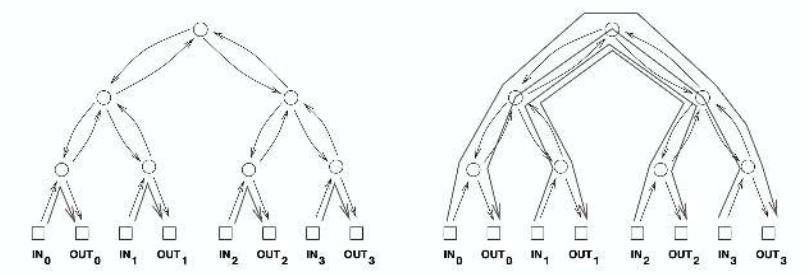
For the networks we consider below, paths from input to output are uniquely determined (in the case of the tree) or all paths are the same length, so network latency will always equal network diameter.

This page titled 10.5: Network Latency is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

10.6: Congestion

The complete binary tree has a fatal drawback: the root switch is a bottleneck. At best, this switch must handle right and vice-versa. Passing all these packets through a single switch could take a long time. At worst, if this switch fails, the network is broken into two equal-sized pieces.

It's true that if the routing problem is given by the identity permutation, $Id(i) ::= i$, then there is an easy routing, P, that solves the problem: let P_i be the path from input i up through one switch and back down to output i. On the other hand, if the problem was given by $\pi(i) ::= (N-1) - i$, then in *any* solution, Q, for π, each path Q_i beginning at input i must eventually loop all the way up through the root switch and then travel back down to output $(N-1) - i$. These two situations are illustrated below. We can distinguish between a "good" set of paths and a "bad" set based on congestion. The *congestion* of a routing, P, is equal to the largest number of paths in P that pass through a single switch. For example, the congestion of the routing on the left is 1, since at most 1 path passes through each switch. However, the congestion of the routing on the right is 4, since 4 paths pass through the root switch (and the two switches directly below the root). Generally, lower congestion is better since packets can be delayed at an overloaded switch.



By extending the notion of congestion to networks, we can also distinguish between "good" and "bad" networks with respect to bottleneck problems. For each routing problem, π, for the network, we assume a routing is chosen that optimizes congestion, that is, that has the minimum congestion among all routings that solve π. Then the largest congestion that will ever be suffered by a switch will be the maximum congestion among these optimal routings. This "maximin" congestion is called the *congestion of the network*.

So for the complete binary tree, the worst permutation would be $\pi(i) ::= (N-1) - i$. Then in every possible solution for π, *every* packet would have to follow a path passing through the root switch. Thus, the max congestion of the complete binary tree is N—which is horrible!

Let's tally the results of our analysis so far:

network	diameter	switch size	# switches	congestion
complete binary tree	$2 \log N + 2$	3×3	$2N - 1$	N

This page titled 10.6: Congestion is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

10.7: 2-D Array

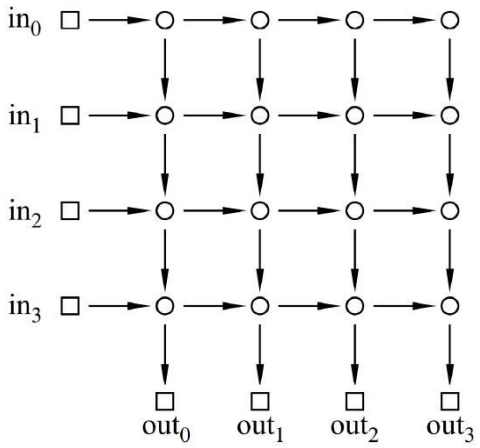
Let's look at an another communication network. This one is called a *2-dimensional array* or *grid*.

Here there are four inputs and four outputs, so $N = 4$.

The diameter in this example is 8, which is the number of edges between input 0 and output 3. More generally, the diameter of an array with N inputs and outputs is $2N$, which is much worse than the diameter of $2 \log N + 2$ in the complete binary tree. But we get something in exchange: replacing a complete binary tree with an array almost eliminates congestion.

Theorem 10.7.1

The congestion of an N-input array is 2.



Proof. First, we show that the congestion is at most 2. Let π be any permutation. Define a solution, P, for π to be the set of paths, P_i, where P_i goes to the right from input i to column $\pi(i)$ and then goes down to output $\pi(i)$. Thus, the switch in row i and column j transmits at most two packets: the packet originating at input i and the packet destined for output j.

Next, we show that the congestion is at least 2. This follows because in any routing problem, π, where $\pi(0) = 0$ and $\pi(N-1) = N-1$, two packets must pass through the lower left switch. ■

As with the tree, the network latency when minimizing congestion is the same as the diameter. That's because all the paths between a given input and output are the same length.

Now we can record the characteristics of the 2-D array.

network	diameter	switch size	# switches	congestion
complete binary tree	$2 \log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2

The crucial entry here is the number of switches, which is N^2. This is a major defect of the 2-D array; a network of size $N = 1000$ would require a *million* 2×2 switches! Still, for applications where N is small, the simplicity and low congestion of the array make it an attractive choice.

This page titled 10.7: 2-D Array is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

10.8: Butterfly

The Holy Grail of switching networks would combine the best properties of the complete binary tree (low diameter, few switches) and of the array (low congestion). The *butterfly* is a widely-used compromise between the two.

A good way to understand butterfly networks is as a recursive data type. The recursive definition works better if we define just the switches and their connections, omitting the terminals. So we recursively define F_n to be the switches and connections of the butterfly net with $N ::= 2^n$ input and output switches.

The base case is F_1 with 2 input switches and 2 output switches connected as in Figure 10.1.

In the constructor step, we construct F_{n+1} with 2^{n+1} inputs and outputs out of two F_n nets connected to a new set of 2^{n+1} input switches, as shown in as in Figure 10.2. That is, the ith and $2^n + i$th new input switches are each connected to the same two switches, the ith input switches of each of two F_n components for $i = 1, \ldots, 2^n$. The output switches of F_{n+1} are simply the output switches of each of the F_n copies.

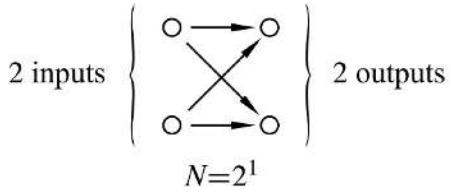


Figure 10.1 F_1, the Butterfly Net switches with $N = 2^1$.

So F_{n+1} is laid out in columns of height 2^{n+1} by adding one more column of switches to the columns in F_n. Since the construction starts with two columns when $n + 1$, the F_{n+1} switches are arrayed in $n + 1$ columns. The total number of switches is the height of the columns times the number of columns, $2^{n+1}(n + 1)$. Remembering that $n = \log N$, we conclude that the Butterfly Net with N inputs has $N(\log N + 1)$ switches.

Since every path in F_{n+1} from an input switch to an output is the same length, $n + 1$, the diameter of the Butterfly net with 2^{n+1} inputs is this length plus two because of the two edges connecting to the terminals (square boxes) —one edge from input terminal to input switch (circle) and one from output switch to output terminal.

There is an easy recursive procedure to route a packet through the Butterfly Net. In the base case, there is only one way to route a packet from one of the two inputs to one of the two outputs. Now suppose we want to route a packet from an input switch to an output switch in F_{n+1}. If the output switch is in the "top" copy of F_n, then the first step in the route must be from the input switch to the unique switch it is connected to in the top copy; the rest of the route is determined by recursively routing the rest of the way in the top copy of F_n. Likewise, if the output switch is in the "bottom" copy of F_n, then the first step in the route must be to the switch in the bottom copy, and the rest of the route is determined by recursively routing in the bottom copy of F_n. In fact, this argument shows that the routing is *unique*: there is exactly one path in the Butterfly Net from each input to each output, which implies that the network latency when minimizing congestion is the same the diameter.

The congestion of the butterfly network is about p as $\sqrt{N}$. More precisely, the congestion is $\sqrt{N}$ if N is an even power of 2 and $\sqrt{N/2}$ if N is an odd power of 2. A simple proof of this appears in Problem 10.8.

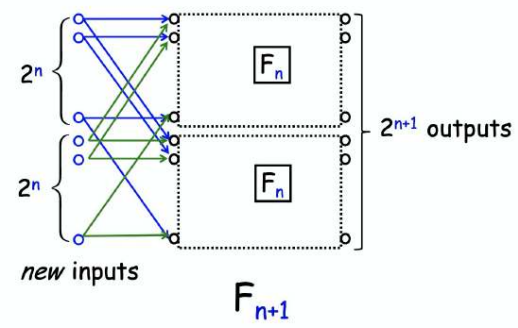


Figure 10.2 F_{n+1}, the Butterfly Net switches with 2^{n+1} inputs and outputs.

Let's add the butterfly data to our comparison table:

network	diameter	switch size	# switches	congestion
complete binary tree	$2\log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2
butterfly	$\log N + 2$	2×2	$N(\log(N) + 1)$	$\sqrt{N}$ or $\sqrt{N/2}$

The butterfly has lower congestion than the complete binary tree. It also uses fewer switches and has lower diameter than the array. However, the butterfly does not capture the best qualities of each network, but rather is a compromise somewhere between the two. Our quest for the Holy Grail of routing networks goes on.

This page titled 10.8: Butterfly is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

10.9: Beneš Network

In the 1960's, a researcher at Bell Labs named Václav E. Beneš had a remarkable idea. He obtained a marvelous communication network with congestion 1 by placing *two* butterflies back-to-back. This amounts to recursively growing *Beneš nets* by adding both inputs and outputs at each stage. Now we recursively define B_n to be the switches and connections (without the terminals) of the Beneš net with $N ::= 2^n$ input and output switches.

The base case, B_1, with 2 input switches and 2 output switches is exactly the same as F_1 in Figure 10.1.

In the constructor step, we construct B_{n+1} out of two B_n nets connected to a new set of 2^{n+1} input switches and also a new set of 2^{n+1} output switches. This is illustrated in Figure 10.3.

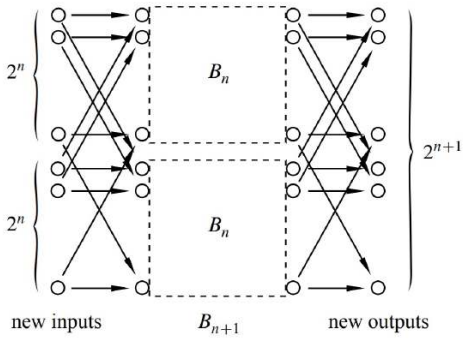


Figure 10.3 B_{n+1}, the Beneš Net switches with 2^{n+1} inputs and outputs.

The ith and 2^{n+i}th new input switches are each connected to the same two switches: the ith input switches of each of two B_n components for $i = 1, ldots, 2^n$, exactly as in the Butterfly net. In addition, the ith and 2^{n+i}th new *output* switches are connected to the same two switches, namely, to the ith output switches of each of two B_n components.

Now, B_{n+1} is laid out in columns of height 2^{n+1} by adding two more columns of switches to the columns in B_n. So, the B_{n+1} switches are arrayed in $2(n+1)$ columns. The total number of switches is the number of columns times the height of the columns, $2(n+1)2^{n+1}$.

All paths in B_{n+1} from an input switch to an output are length $2(n+1)-1$, and the diameter of the Beneš net with 2^{n+1} inputs is this length plus two because of the two edges connecting to the terminals.

So Beneš has doubled the number of switches and the diameter, but by doing so he has completely eliminated congestion problems! The proof of this fact relies on a clever induction argument that we'll come to in a moment. Let's first see how the Beneš network stacks up:

network	diameter	switch size	# switches	congestion
complete binary tree	$2\log N + 2$	3×3	$2N - 1$	N
2-D array	$2N$	2×2	N^2	2
butterfly	$\log N + 2$	2×2	$N(\log(N) + 1)$	$\sqrt{N}$ or $\sqrt{N/2}$
Beneš	$2\log N + 1$	2×2	$2N \log N$	1

The Beneš network has small size and diameter, and it completely eliminates congestion. The Holy Grail of routing networks is in hand!

Theorem 10.9.1

The congestion of the N-input Beneš network is 1.

Proof

By induction on n where $N = 2^n$. So the induction hypothesis is

$$P(n) ::= \text{the congestion of } B_n \text{ is 1.}$$

Base case $(n = 1)$: $B_1 = F_1$ is shown in Figure 10.1. The unique routings in F_1 have congestion 1.

Inductive step: We assume that the congestion of an $N = 2^n$-input Beneš network is 1 and prove that the congestion of a $2N$-input Beneš network is also 1.

Digression. Time out! Let's work through an example, develop some intuition, and then complete the proof. In the Beneš network shown in Figure 10.4 with $N = 8$ inputs and outputs, the two 4-input/output subnetworks are in dashed boxes.

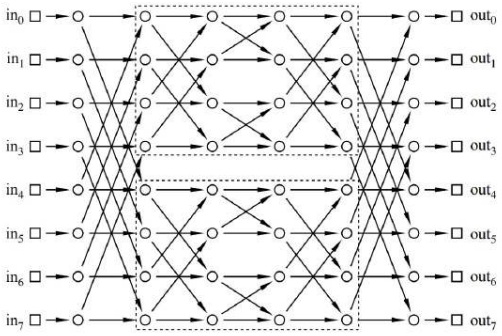
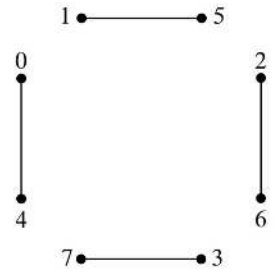


Figure 10.4 Beneš net B3.

By the inductive assumption, the subnetworks can each route an arbitrary permutation with congestion 1. So if we can guide packets safely through just the first and last levels, then we can rely on induction for the rest! Let's see how this works in an example. Consider the following permutation routing problem:

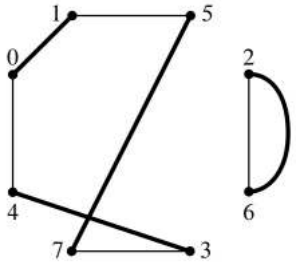
$$\begin{array}{ll} \pi(0) = 1 & \pi(4) = 3 \\ \pi(1) = 5 & \pi(5) = 6 \\ \pi(2) = 4 & \pi(6) = 0 \\ \pi(3) = 7 & \pi(7) = 2 \end{array}$$

We can route each packet to its destination through either the upper subnetwork or the lower subnetwork. However, the choice for one packet may constrain the choice for another. For example, we cannot route both packet 0 *and* packet 4 through the same network, since that would cause two packets to collide at a single switch, resulting in congestion. Rather, one packet must go through the upper network and the other through the lower network. Similarly, packets 1 and 5, 2 and 6, and 3 and 7 must be routed through different networks. Let's record these constraints in a graph. The vertices are the 8 packets. If two packets must pass through different networks, then there is an edge between them. Thus, our constraint graph looks like this:



Notice that at most one edge is incident to each vertex.

The output side of the network imposes some further constraints. For example, the packet destined for output 0 (which is packet 6) and the packet destined for output 4 (which is packet 2) cannot both pass through the same network; that would require both packets to arrive from the same switch. Similarly, the packets destined for outputs 1 and 5, 2 and 6, and 3 and 7 must also pass through different switches. We can record these additional constraints in our graph with gray edges:



Notice that at most one new edge is incident to each vertex. The two lines drawn between vertices 2 and 6 reflect the two different reasons why these packets must be routed through different networks. However, we intend this to be a simple graph; the two lines still signify a single edge.

Now here's the key insight: suppose that we could color each vertex either red or blue so that adjacent vertices are colored differently. Then all constraints are satisfied if we send the red packets through the upper network and the blue packets through the lower network. Such a *2-coloring of the graph corresponds to a solution to the routing problem*. The only remaining question is whether the constraint graph is 2-colorable, which is easy to verify:

Lemma 10.9.2. *Prove that if the edges of a graph can be grouped into two sets such that every vertex has at most 1 edge from each set incident to it, then the graph is 2-colorable.*

Proof. It is not hard to show that a graph is 2-colorable iff every cycle in it has even length (see Theorem 11.9.3). We'll take this for granted here.

So all we have to do is show that every cycle has even length. Since the two sets of edges may overlap, let's call an edge that is in both sets a *doubled edge*.

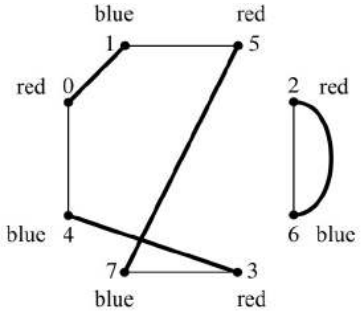
There are two cases:

Case 1: [The cycle contains a doubled edge.] No other edge can be incident to either of the endpoints of a doubled edge, since that endpoint would then be incident to two edges from the same set. So a cycle traversing a doubled edge has nowhere to go but back and forth along the edge an even number of times.

Case 2: [No edge on the cycle is doubled.] Since each vertex is incident to at most one edge from each set, any path with no doubled edges must traverse successive edges that alternate from one set to the other. In particular, a cycle must traverse a path of alternating edges that begins and ends with edges from different sets. This means the cycle has to be of even length. ■

For example, here is a 2-coloring of the constraint graph:

The solution to this graph-coloring problem provides a start on the packet routing problem:



We can complete the routing in the two smaller Beneš networks by induction! Back to the proof. **End of Digression**.

Let π be an arbitrary permutation of $\{0, 1, \ldots, N-1\}$. Let G be the graph whose vertices are packet numbers $0, 1, \ldots, N-1$ and whose edges come from the union of these two sets:

$$E_1 ::= \{\langle u - v\rangle \mid |u - v| = N/2\}, \text{ and}$$
$$E_2 ::= \{\langle u - w\rangle \mid |\pi(u) - \pi(w)| = N/2\}.$$

Now any vertex, u, is incident to at most two edges: a unique edge $\langle u - v \rangle \in E_1$ and a unique edge $\langle u - w \rangle \in E_2$. So according to Lemma 10.9.2, there is a 2- coloring for the vertices of G. Now route packets of one color through the upper subnetwork and packets of the other color through the lower subnetwork. Since for each edge in E_1, one vertex goes to the upper subnetwork and the other to the lower subnetwork, there will not be any conflicts in the first level. Since for each edge in E_2, one vertex comes from the upper subnetwork and the other from the lower subnetwork, there will not be any conflicts in the last level. We can complete the routing within each subnetwork by the induction hypothesis $P(n)$. ■

This page titled 10.9: Beneš Network is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

11: Simple Graphs

Simple graphs model relationships that are *symmetric*, meaning that the relationship is mutual. Examples of such mutual relationships are being married, speaking the same language, not speaking the same language, occurring during overlapping time intervals, or being connected by a conducting wire. They come up in all sorts of applications, including scheduling, constraint satisfaction, computer graphics, and communications, but we'll start with an application designed to get your attention: we are going to make a professional inquiry into sexual behavior. Specifically, we'll look at some data about who, on average, has more opposite-gender partners: men or women.

Sexual demographics have been the subject of many studies. In one of the largest, researchers from the University of Chicago interviewed a random sample of 2500 people over several years to try to get an answer to this question. Their study, published in 1994 and entitled *The Social Organization of Sexuality*, found that men have on average 74% more opposite-gender partners than women.

Other studies have found that the disparity is even larger. In particular, ABC News claimed that the average man has 20 partners over his lifetime, and the average woman has 6, for a percentage disparity of 233%. The ABC News study, aired on Primetime Live in 2004, purported to be one of the most scientific ever done, with only a 2.5% margin of error. It was called "American Sex Survey: A peek between the sheets"—raising some questions about the seriousness of their reporting.

Yet again in August, 2007, the New York Times reported on a study by the National Center for Health Statistics of the U.S. government showing that men had seven partners while women had four. So, whose numbers do you think are more accurate: the University of Chicago, ABC News, or the National Center?

Don't answer—this is a trick question designed to trip you up. Using a little graph theory, we'll explain why none of these findings can be anywhere near the truth.

11.1: Vertex Adjacency and Degrees
11.2: Sexual Demographics in America
11.3: Some Common Graphs
11.4: Isomorphism
11.5: Bipartite Graphs and Matchings
11.6: The Stable Marriage Problem
11.7: Coloring
11.8: Simple Walks
11.9: Connectivity
11.10: Forests and Trees

This page titled 11: Simple Graphs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.1: Vertex Adjacency and Degrees

Simple graphs are defined as digraphs in which edges are *undirected*—they connect two vertices without pointing in either direction between the vertices. So instead of a directed edge $\langle v \to w \rangle$ which starts at vertex v and ends at vertex w, a simple graph only has an undirected edge, $\langle v \to w \rangle$, that connects v and w.

Definition 11.1.1

A *simple graph*, G, consists of a nonempty set, $V(G)$, called the *vertices* of G, and a set $E(G)$ called the *edges* of G. An element of $V(G)$ is called a *vertex*. A vertex is also called a *node*; the words "vertex" and "node" are used interchangeably. An element of $E(G)$ is an *undirected edge* or simply an "edge." An undirected edge has two vertices $u \neq v$ called its *endpoints*. Such an edge can be represented by the two element set $\{u, v\}$. The notation $\langle u \to v \rangle$ denotes this edge.

Both $\langle u \to v \rangle$ and $\langle v \to u \rangle$ define the same undirected edge, whose endpoints are u and v.

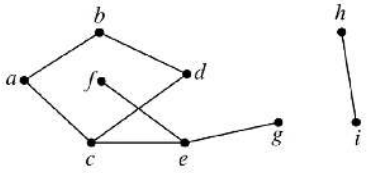


Figure 11.1 An example of a graph with 9 nodes and 8 edges.

For example, let H be the graph pictured in Figure 11.1. The vertices of H correspond to the nine dots in Figure 11.1, that is,

$$V(H) = \{a, b, c, d, e, f, g, h, i\}.$$

The edges correspond to the eight lines, that is,

$$E(H) = \{\langle a \to b \rangle, \langle a \to c \rangle, \langle b \to d \rangle, \langle c \to d \rangle, \langle c \to e \rangle, \langle e \to f \rangle, \langle e \to g \rangle, \langle h \to i \rangle\}.$$

Mathematically, that's all there is to the graph H.

Definition: 11.1.2

Two vertices in a simple graph are said to be *adjacent* iff they are the endpoints of the same edge, and an edge is said to be *incident* to each of its endpoints. The number of edges incident to a vertex v is called the *degree* of the vertex and is denoted by $\deg(v)$. Equivalently, the degree of a vertex is the number of vertices adjacent to it.

For example, for the graph H of Figure 11.1, vertex a is adjacent to vertex b, and b is adjacent to d. The edge $\langle a \to c \rangle$ is incident to its endpoints a and c. Vertex h has degree 1, d has degree 2, and $\deg(e) = 3$. It is possible for a vertex to have degree 0, in which case it is not adjacent to any other vertices. A simple graph, G, does not need to have any edges at all. $|E(G)|$ could be zero, implying that the degree of every vertex would also be zero. But a simple graph must have at least one vertex—$|V(G)|$ is required to be at least one.

An edge whose endpoints are the same is called a *self-loop*. Self-loops aren't allowed in simple graphs.[1] In a more general class of graphs called *multigraphs*, there can be more than one edge with the same two endpoints, but this doesn't happen in simple graphs, because every edge is uniquely determined by its two endpoints. Sometimes graphs with no vertices, with self-loops, or with more than one edge between the same two vertices are convenient to have, but we don't need them, and sticking with simple graphs is simpler.

For the rest of this chapter we'll use "graphs" as an abbreviation for "simple graphs."

A synonym for "vertices" is "*nodes*," and we'll use these words interchangeably. Simple graphs are sometimes called *networks*, edges are sometimes called *arcs*. We mention this as a "heads up" in case you look at other graph theory literature; we won't use these words.

[1]You might try to represent a self-loop going between a vertex v and itself as $\{v, v\}$, but this equals $\{v\}$. It wouldn't be an edge, which is defined to be a set of *two* vertices.

This page titled 11.1: Vertex Adjacency and Degrees is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.2: Sexual Demographics in America

Let's model the question of heterosexual partners in graph theoretic terms. To do this, we'll let G be the graph whose vertices, V, are all the people in America. Then we split V into two separate subsets: M, which contains all the males, and F, which contains all the females.[2] We'll put an edge between a male and a female iff they have been sexual partners. This graph is pictured in Figure 11.2 with males on the left and females on the right.

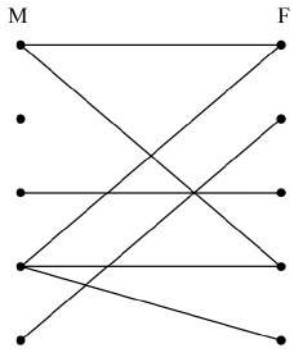


Figure 11.2 The sex partners graph.

Actually, this is a pretty hard graph to figure out, let alone draw. The graph is *enormous*: the US population is about 300 million, so $|V| \approx 300M$. Of these, approximately 50.8% are female and 49.2% are male, so $|M| \approx 147.6M$, and $|F| \approx 152.4M$. And we don't even have trustworthy estimates of how many edges there are, let alone exactly which couples are adjacent. But it turns out that we don't need to know any of this—we just need to figure out the relationship between the average number of partners per male and partners per female. To do this, we note that every edge has exactly one endpoint at an M vertex (remember, we're only considering male-female relationships); so the sum of the degrees of the M vertices equals the number of edges. For the same reason, the sum of the degrees of the F vertices equals the number of edges. So these sums are equal:

$$\sum_{x \in M} \deg(x) = \sum_{y \in F} \deg(y)$$

Now suppose we divide both sides of this equation by the product of the sizes of the two sets, $|M| \cdot |F|$:

$$\frac{\sum_{x \in M} \deg(x)}{|M|} \cdot \frac{1}{|F|} = \frac{\sum_{y \in F} \deg(y)}{|F|} \cdot \frac{1}{|M|}$$

The terms above in parentheses are the *average degree of an M vertex and the average degree of an F vertex. So we know:*

$$\text{Avg. deg in } M = \frac{|F|}{|M|} \cdot \text{Avg. deg in } F \tag{11.2.1}$$

In other words, we've proved that the average number of female partners of males in the population compared to the average number of males per female is *determined solely by the relative number of males and females in the population.*

Now the Census Bureau reports that there are slightly more females than males in America; in particular $|F|/|M|$ is about 1.035. So we know that males have on average 3.5% more opposite-gender partners than females, and that this tells us nothing about any sex's promiscuity or selectivity. Rather, it just has to do with the relative number of males and females. Collectively, males and females have the same number of opposite gender partners, since it takes one of each set for every partnership, but there are fewer males, so they have a higher ratio. This means that the University of Chicago, ABC, and the Federal government studies are way off. After a huge effort, they gave a totally wrong answer.

There's no definite explanation for why such surveys are consistently wrong. One hypothesis is that males exaggerate their number of partners—or maybe females downplay theirs—but these explanations are speculative. Interestingly, the principal author of the National Center for Health Statistics study reported that she knew the results had to be wrong, but that was the data collected, and her job was to report it.

The same underlying issue has led to serious misinterpretations of other survey data. For example, a couple of years ago, the Boston Globe ran a story on a survey of the study habits of students on Boston area campuses. Their survey showed that on

average, minority students tended to study with non-minority students more than the other way around. They went on at great length to explain why this “remarkable phenomenon” might be true. But it’s not remarkable at all. Using our graph theory formulation, we can see that all it says is that there are fewer minority students than non-minority students, which is, of course, what “minority” means.

Handshaking Lemma

The previous argument hinged on the connection between a sum of degrees and the number of edges. There is a simple connection between these in any graph:

Lemma 11.2.1. *The sum of the degrees of the vertices in a graph equals twice the number of edges.*

Proof. Every edge contributes two to the sum of the degrees, one for each of its endpoints. ■

We refer to Lemma 11.2.1 as the *Handshaking Lemma*: if we total up the number of people each person at a party shakes hands with, the total will be twice the number of handshakes that occurred.

[2]For simplicity, we’ll ignore the possibility of someone being *both* a man and a woman, or neither.

This page titled 11.2: Sexual Demographics in America is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.3: Some Common Graphs

Some graphs come up so frequently that they have names. A *complete graph* K_n has n vertices and an edge between every two vertices, for a total of $n(n-1)/2$ edges. For example, K_5 is shown in Figure 11.3.

The *empty graph* has no edges at all. For example, the empty graph with 5 nodes is shown in Figure 11.4.

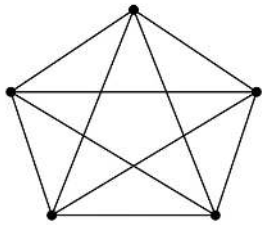


Figure 11.3 K_5: the complete graph on 5 nodes.

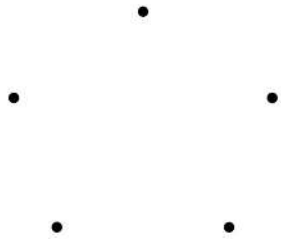


Figure 11.4 An empty graph with 5 nodes.

An n-node graph containing n_1 edges in sequence is known as a *line graph* L_n. More formally, L_n has

$$V(L_n) = \{v_1, v_2, \ldots, v_n\}$$

and

$$E(L_n) = \{\langle v_1 - v_2\rangle, \langle v_2 - v_3\rangle, \ldots, \langle v_{n-1} - v_n\rangle\}$$

For example, L_5 is pictured in Figure 11.5.

There is also a one-way infinite line graph L_∞ which can be defined by letting the nonnegative integers $\mathbb{N}$ be the vertices with edges $(k-(k+1))$ for all $k \in \mathbb{N}$.

If we add the edge $\langle v_n - v_1\rangle$ to the line graph L_n, we get a graph called a *length-n cycle* C_n. Figure 11.6 shows a picture of length-5 cycle.

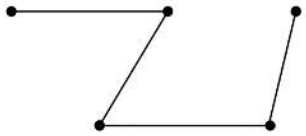


Figure 11.5 L_5: a 5-node line graph.

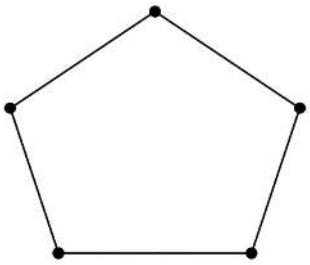


Figure 11.6 C_5: a 5-node cycle graph.

This page titled 11.3: Some Common Graphs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.4: Isomorphism

Two graphs that look different might actually be the same in a formal sense. For example, the two graphs in Figure 11.7 are both 4-vertex, 5-edge graphs and you get graph (b) by a o 90 clockwise rotation of graph (a).

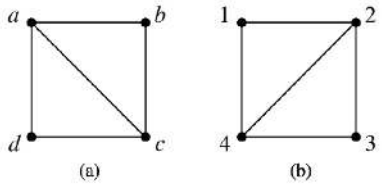


Figure 11.7 Two Isomorphic graphs.

Strictly speaking, these graphs are different mathematical objects, but this difference doesn't reflect the fact that the two graphs can be described by the same picture—except for the labels on the vertices. This idea of having the same picture "up to relabeling" can be captured neatly by adapting Definition 9.7.1 of isomorphism of digraphs to handle simple graphs. An isomorphism between two graphs is an edge-preserving bijection between their sets of vertices:

Definition 11.4.1

An isomorphism between graphs G and H is a bijection $f : V(G) \to V(H)$ such that

$$\langle u - v\rangle \in E(G) \text{ iff } \langle f(u) - f(v)\rangle \in E(H)$$

for all $u, v \in V(G)$. Two graphs are isomorphic when there is an isomorphism between them.

Here is an isomorphism, f , between the two graphs in Figure 11.7:

$$\begin{array}{llll} f(a) ::= 2 & f(b) & ::= 3 \\ f(c) ::= 4 & f(d) & ::= 1. \end{array} \tag{11.4.1}$$

You can check that there is an edge between two vertices in the graph on the left if and only if there is an edge between the two corresponding vertices in the graph on the right.

Two isomorphic graphs may be drawn very differently. For example, Figure 11.8 shows two different ways of drawing C_5.

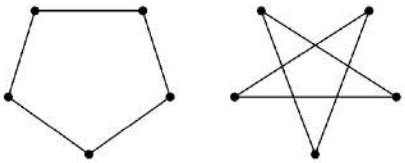


Figure 11.8 Isomorphic C_5 graphs.

Notice that if f is an isomorphism between G and H, then f^{-1} is an isomorphism between H and G. Isomorphism is also transitive because the composition of isomorphisms is an isomorphism. In fact, isomorphism is an equivalence relation.

Isomorphism preserves the connection properties of a graph, abstracting out what the vertices are called, what they are made out of, or where they appear in a drawing of the graph. More precisely, a property of a graph is said to be *preserved under isomorphism* if whenever G has that property, every graph isomorphic to G also has that property. For example, since an isomorphism is a bijection between sets of vertices, isomorphic graphs must have the same number of vertices. What's more, if f is a graph isomorphism that maps a vertex, v, of one graph to the vertex, $f(v)$, of an isomorphic graph, then by definition of isomorphism, every vertex adjacent to v in the first graph will be mapped by f to a vertex adjacent to $f(v)$ in the isomorphic graph. Thus, v and $f(v)$ will have the same degree. If one graph has a vertex of degree 4 and another does not, then they can't be isomorphic. In fact, they can't be isomorphic if the number of degree 4 vertices in each of the graphs is not the same.

Looking for preserved properties can make it easy to determine that two graphs are not isomorphic, or to guide the search for an isomorphism when there is one. It's generally easy in practice to decide whether two graphs are isomorphic. However, no one has

yet found a procedure for determining whether two graphs are isomorphic that is *guaranteed* to run in polynomial time on all pairs of graphs.[3]

Having such a procedure would be useful. For example, it would make it easy to search for a particular molecule in a database given the molecular bonds. On the other hand, knowing there is no such efficient procedure would also be valuable: secure protocols for encryption and remote authentication can be built on the hypothesis that graph isomorphism is computationally exhausting.

The definitions of bijection and isomorphism apply to infinite graphs as well as finite graphs, as do most of the results in the rest of this chapter. But graph theory focuses mostly on finite graphs, and we will too. *In the rest of this chapter we'll assume graphs are finite*.

We've actually been taking isomorphism for granted ever since we wrote "K_n has n vertices. . . " at the beginning of Section 11.3.

Graph theory is all about properties preserved by isomorphism.

[3]A procedure runs in *polynomial time* when it needs an amount of time of at most $p(n)$, where n is the total number of vertices and $p()$ is a fixed polynomial.

This page titled 11.4: Isomorphism is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.5: Bipartite Graphs and Matchings

There were two kinds of vertices in the “Sex in America” graph, males and females, and edges only went between the two kinds. Graphs like this come up so frequently that they have earned a special name: *bipartite graphs*.

Definition 11.5.1

A *bipartite graph* is a graph whose vertices can be partitioned[4] into two sets, $L(G)$ and $R(G)$, such that every edge has one endpoint in $L(G)$ and the other endpoint in $R(G)$.

So every bipartite graph looks something like the graph in Figure 11.2.

The Bipartite Matching Problem

The bipartite matching problem is related to the sex-in-America problem that we just studied; only now, the goal is to get everyone happily married. As you might imagine, this is not possible for a variety of reasons, not the least of which is the fact that there are more women in America than men. So, it is simply not possible to marry every woman to a man so that every man is married at most once.

But what about getting a mate for every man so that every woman is married at most once? Is it possible to do this so that each man is paired with a woman that he likes? The answer, of course, depends on the bipartite graph that represents who likes who, but the good news is that it is possible to find natural properties of the who-likes-who graph that completely determine the answer to this question.

In general, suppose that we have a set of men and an equal-sized or larger set of women, and there is a graph with an edge between a man and a woman if the man likes the woman. In this scenario, the “likes” relationship need not be symmetric, since for the time being, we will only worry about finding a mate for each man that he likes.[5] (Later, we will consider the “likes” relationship from the female perspective as well.) For example, we might obtain the graph in Figure 11.9.

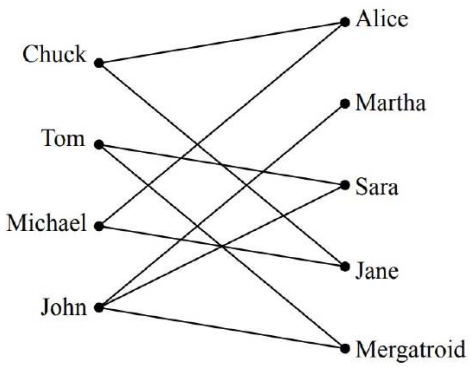


Figure 11.9 A graph where an edge between a man and woman denotes that the man likes the woman.

A *matching* is defined to be an assignment of a woman to each man so that different men are assigned to different women, and a man is always assigned a woman that he likes. For example, one possible matching for the men is shown in Figure 11.10.

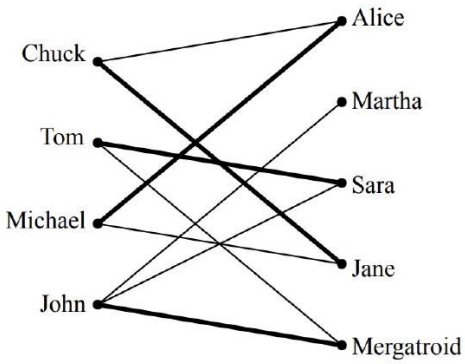


Figure 11.10 One possible matching for the men is shown with bold edges. For example, John is matched with Mergatroid.

The Matching Condition

A famous result known as Hall's Matching Theorem gives necessary and sufficient conditions for the existence of a matching in a bipartite graph. It turns out to be a remarkably useful mathematical tool.

We'll state and prove Hall's Theorem using man-likes-woman terminology. Define *the set of women liked by a given set of men* to consist of all women liked by at least one of those men. For example, the set of women liked by Tom and John in Figure 11.9 consists of Martha, Sara, and Mergatroid. For us to have any chance at all of matching up the men, the following *matching condition* must hold:

The *Matching Condition*: every subset of men likes at least as large a set of women.

For example, we cannot find a matching if some set of 4 men like only 3 women. Hall's Theorem says that this necessary condition is actually sufficient; if the matching condition holds, then a matching exists.

Theorem 11.5.2

A matching for a set M of men with a set W of women can be found if and only if the matching condition holds.

Proof

First, let's suppose that a matching exists and show that the matching condition holds. For any subset of men, each man likes at least the woman he is matched with and a woman is matched with at most one man. Therefore, every subset of men likes at least as large a set of women. Thus, the matching condition holds.

Next, let's suppose that the matching condition holds and show that a matching exists. We use strong induction on $|M|$, the number of men, on the predicate:

$$P(m) ::= \text{if the matching condition holds for a set, } M, \text{ of } m \text{ men, then there is a matching for } M.$$

Base case ($|M| = 1$): If $|M| = 1$, then the matching condition implies that the lone man likes at least one woman, and so a matching exists.

Inductive Step: Suppose that $|M| = m + 1 \geq 2$. To find a matching for M, there are two cases.

Case 1: Every nonempty subset of at most m men likes a *strictly larger* set of women. In this case, we have some latitude: we pair an arbitrary man with a woman he likes and send them both away. This leaves m men and one fewer women, and the matching condition will still hold. So the induction hypothesis $P(m)$ implies we can match the remaining m men.

Case 2: Some nonempty subset, X, of at most m men likes an *equal-size* set, Y, of women. The matching condition must hold within X, so the strong induction hypothesis implies we can match the men in X with the women in Y. This leaves the problem of matching the set $M - X$ of men to the set $W - Y$ of women.

But the problem of matching $M - X$ against $W - Y$ also satisfies the Matching condition, because any subset of men in $M - X$ who liked fewer women in $W - Y$ would imply there was a set of men who liked fewer women in the whole set W. Namely, if a subset $M_0 \subseteq M - X$ liked only a strictly smaller subset of women $W_0 \subseteq W - Y$, then the set $M_0 \cup X$ of men

would like only women in the strictly smaller set $W_0 \cup Y$. So again the strong induction hypothesis implies we can match the men in $M - X$ with the women in $W - Y$, which completes a matching for M.

So in both cases, there is a matching for the men, which completes the proof of the Inductive step. The theorem follows by induction. ■

The proof of Theorem 11.5.2 gives an algorithm for finding a matching in a bipartite graph, albeit not a very efficient one. However, efficient algorithms for finding a matching in a bipartite graph do exist. Thus, if a problem can be reduced to finding a matching, the problem is essentially solved from a computational perspective.

A Formal Statement

Let's restate Theorem 11.5.2 in abstract terms so that you'll not always be condemned to saying, "Now this group of men likes at least as many women. . . "

Definition 11.5.3

A *matching* in a graph G is a set M of edges of G such that no vertex is an endpoint of more than one edge in M. A matching is said to *cover* a set, S, of vertices iff each vertex in S is an endpoint of an edge of the matching. A matching is said to be *perfect* if it covers $V(G)$. In any graph, G, the set $N(S)$ of *neighbors* of some set S of vertices is the image of S under the edge-relation, that is,

$$N(S) ::= \{r \mid \langle s - r\rangle \in E(G) \text{ for some } s \in S\}.$$

S is called a *bottleneck* if

$$|S| > |N(S)|.$$

Theorem 11.5.4

(Hall's Theorem). *Let G be a bipartite graph. There is a matching in G that covers $L(G)$ iff no subset of $L(G)$ is a bottleneck.*

An Easy Matching Condition

The bipartite matching condition requires that *every* subset of men has a certain property. In general, verifying that every subset has some property, even if it's easy to check any particular subset for the property, quickly becomes overwhelming because the number of subsets of even relatively small sets is enormous—over a billion subsets for a set of size 30. However, there is a simple property of vertex degrees in a bipartite graph that guarantees the existence of a matching. Call a bipartite graph *degree-constrained* if vertex degrees on the left are at least as large as those on the right. More precisely,

Definition 11.5.5

A bipartite graph G is *degree-constrained* when $\deg(l) \geq \deg(r)$ for every $l \in L(G)$ and $r \in R(G)$.

For example, the graph in Figure 11.9 is degree-constrained since every node on the left is adjacent to at least two nodes on the right while every node on the right is adjacent to at most two nodes on the left.

Theorem 11.5.6

If G is a degree-constrained bipartite graph, then there is a matching that covers $L(G)$.

Proof

We will show that G satisfies Hall's condition, namely, if S is an arbitrary subset of $L(G)$, then

$$|N(S)| \geq |S|. \tag{11.5.1}$$

Since G is degree-constrained, there is a $d > 0$ such that $\deg(l) \geq d \geq \deg(r)$ for every $l \in L$ and $r \in R$. Since every edge with an endpoint in S has its other endpoint in $N(S)$ by definition, and every node in $N(S)$ is incident to at most d edges, we

know that

Misplaced '#'

Also, since every node in S is the endpoint of at least d edges,

Misplaced '#'

It follows that $d|N(S)| \geq d|S|$. Cancelling d completes the derivation of equation (11.5.1).

Regular graphs are a large class of degree-constrained graphs that often arise in practice. Hence, we can use Theorem 11.5.6 to prove that every regular bipartite graph has a perfect matching. This turns out to be a surprisingly useful result in computer science.

Definition 11.5.7

A graph is said to be *regular* if every node has the same degree.

Theorem 11.5.8

Every regular bipartite graph has a perfect matching.

Proof

Let G be a regular bipartite graph. Since regular graphs are degree-constrained, we know by Theorem 11.5.6 that there must be a matching in G that covers $L(G)$. Such a matching is only possible when $|L(G)| \leq |R(G)|$. But G is also degreeconstrained if the roles of $L(G)$ and $R(G)$ are switched, which implies that $|R(G)| \leq |L(G)|$ also. That is, $L(G)$ and $R(G)$ are the same size, and any matching covering $L(G)$ will also cover $R(G)$. So every node in G is an endpoint of an edge in the matching, and thus G has a perfect matching. ■

[4]Partitioning a set means cutting it up into *nonempty* pieces. In this case, it means that $L(G)$ and $R(G)$ are nonempty, $L(G) \cup R(G) = V(G)$, and $L(G) \cap R(G) = \emptyset$.

[5]By the way, we do not mean to imply that marriage should or should not be heterosexual. Nor do we mean to imply that men should get their choice instead of women. It's just that there are fewer men than women in America, making it impossible to match up all the women with different men. So please don't take offense.

This page titled 11.5: Bipartite Graphs and Matchings is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.6: The Stable Marriage Problem

Let's look at another man/woman matching problem with an equal number of men and women. The set up is that each person has preferences about who they would like to marry: each man has preference list of all the women, and each woman has a preference list of all of the men.

The preferences don't have to be symmetric. That is, Jennifer might like Brad best, but Brad doesn't necessarily like Jennifer best. The goal is to marry everyone: every man must marry exactly one woman and *vice-versa*—no polygamy. Moreover, we would like to find a matching between men and women that is *stable* in the sense that there is no pair of people who prefer one another to their spouses.

For example, suppose Brad likes Angelina best, and Angelina likes Brad best, but Brad and Angelina are married to other people, say Jennifer and Billy Bob. Now *Brad and Angelina prefer each other to their spouses*, which puts their marriages at risk. Pretty soon, they're likely to start spending late nights together working on problem sets!

This unfortunate situation is illustrated in Figure 11.11, where the digits "1" and "2" near a man shows which of the two women he ranks first and second, respectively, and similarly for the women.

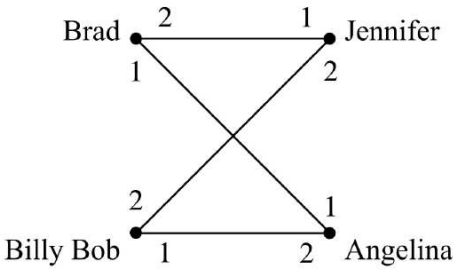


Figure 11.11 Preferences for four people. Both men like Angelina best and both women like Brad best.

More generally, in any matching, a man and woman who are not married to each other and who like each other better than their spouses is called a rogue couple. In the situation shown in Figure 11.11, Brad and Angelina would be a rogue couple.

Having a rogue couple is not a good thing, since it threatens the stability of the marriages. On the other hand, if there are no rogue couples, then for any man and woman who are not married to each other, at least one likes their spouse better than the other, and so there won't be any mutual temptation to start an affair.

Definition 11.6.1

A *stable matching* is a matching with no rogue couples.

The question is, given everybody's preferences, can you find a stable set of marriages? In the example consisting solely of the four people in Figure 11.11, we could let Brad and Angelina both have their first choices by marrying each other. Now neither Brad nor Angelina prefers anybody else to their spouse, so neither will be in a rogue couple. This leaves Jen not-so-happily married to Billy Bob, but neither Jen nor Billy Bob can entice somebody else to marry them, and so this is a stable matching.

It turns out there *always* is a stable matching among a group of men and women. We don't know of any immediate way to recognize this, and it seems surprising. In fact, in the apparently similar "buddy" matching problem where people are supposed to be paired off as buddies, regardless of gender, a stable matching *may not* be possible. An example of preferences among four people where there is no stable buddy match is given in Problem 11.22. But when men are only allowed to marry women, and *vice-versa*, then we will be able to describe a simple procedure to produce a stable matching.[6]

The Mating Ritual

The procedure for finding a stable matching can be described in a memorable way as a *Mating Ritual* that takes place over several days. The following events happen each day:

Morning: Each man stands under the balcony of top choice among the women on his list, and he serenades her. He is said to be her *suitor*. If a man has no women left on his list, he stays home and does his math homework.

Afternoon: Each woman who has one or more suitors says to her favorite among them, "We might get engaged. Please stay around." To the other suitors, she says, "No. I will never marry you! Take a hike!"

Evening: Any man who is told by a woman to take a hike crosses that woman off his preference list.

Termination condition: When a day arrives in which every woman has at most one suitor, the ritual ends with each woman marrying her suitor, if she has one.

There are a number of facts about this Mating Ritual that we would like to prove:

- The Ritual eventually reaches the termination condition.
- Everybody ends up married.
- The resulting marriages are stable

Mating Ritual at Akamai



The Internet infrastructure company Akamai, cofounded by Tom Leighton, also uses a variation of the Mating Ritual to assign web traffic to its servers.

In the early days, Akamai used other combinatorial optimization algorithms that got to be too slow as the number of servers (over 65,000 in 2010) and requests (over 800 billion per day) increased. Akamai switched to a Ritual-like approach, since a Ritual is fast and can be run in a distributed manner. In this case, web requests correspond to women and web servers correspond to men. The web requests have preferences based on latency and packet loss, and the web servers have preferences based on cost of bandwidth and co-location.

There is a Marriage Day

It's easy to see why the Mating Ritual has a terminal day when people finally get married. Every day on which the ritual hasn't terminated, at least one man crosses a woman off his list. (If the ritual hasn't terminated, there must be some woman serenaded by at least two men, and at least one of them will have to cross her off his list). If we start with n men and n women, then each of the n men's lists initially has n women on it, for a total of n^2 list entries. Since no women ever gets added to a list, the total number of entries on the lists decreases every day that the Ritual continues, and so the Ritual can continue for at most n^2 days.

They All Live Happily Ever After. . .

We will prove that the Mating Ritual leaves everyone in a stable marriage. To do this, we note one very useful fact about the Ritual: if on some morning a woman has any suitor, then her favorite suitor will still be serenading her the next morning—his list won't have changed. So she is sure to have today's favorite suitor among her suitors tomorrow. That means she will be able to choose a favorite suitor tomorrow who is at least as desirable to her as today's favorite. So day by day, her favorite suitor can stay the same or get better, never worse. This sounds like an invariant, and it is.

Definition 11.6.2

Let P be the predicate: for every woman, w, and man, m, if w is crossed off m's list, then w has a suitor whom she prefers over m.

Lemma 11.6.3. *P is a preserved invariant for The Mating Ritual.*

Proof. Woman w gets crossed off m's list only when w has a suitor she prefers to m. Thereafter, her favorite suitor doesn't change until one she likes better comes along. So if her favorite suitor was preferable to m, then any new favorite suitor will be as well.

Notice that the invariant P holds vacuously at the beginning since no women are crossed off to start. So by the Invariant Principle, P holds throughout the Ritual. Now we can prove:

Theorem 11.6.4

Everyone is married at the end of the Mating Ritual.

Proof

Assume to the contrary that on the last day of the Mating Ritual, some man—call him Bob—is not married. This means Bob can't be serenading anybody, that is, his list must be empty. So every woman must have been crossed off his list and, since P is true, every woman has a suitor whom she prefers to Bob. In particular, every woman has *some* suitor, and since it is the last day, they have only one suitor, and this is who they marry. But there are an equal number of men and women, so if all women are married, so are all men, contradicting the assumption that Bob is not married. ■

Theorem 11.6.5

The Mating Ritual produces a stable matching.

Proof

Let Brad and Jen be any man and woman, respectively, that are *not* married to each other on the last day of the Mating Ritual. We will prove that Brad and Jen are not a rogue couple, and thus that all marriages on the last day are stable. There are two cases to consider.

Case 1: Jen is not on Brad's list by the end. Then by invariant P, we know that Jen has a suitor (and hence a husband) whom she prefers to Brad. So she's not going to run off with Brad—Brad and Jen cannot be a rogue couple.

Case 2: Jen is on Brad's list. Since Brad picks women to serenade by working down his list, his wife must be higher on his preference list than Jen. So he's not going to run off with Jen—once again, Brad and Jen are not a rogue couple. ■

. . Especially the Men

Who is favored by the Mating Ritual, the men or the women? The women *seem* to have all the power: each day they choose their favorite suitor and reject the rest. What's more, we know their suitors can only change for the better as the Ritual progresses. Similarly, a man keeps serenading the woman he most prefers among those on his list until he must cross her off, at which point he serenades the next most preferred woman on his list. So from the man's perspective, the woman he is serenading can only change for the worse. Sounds like a good deal for the women.

But it's not! We will show that the men are by far the favored gender under the Mating Ritual.

While the Mating Ritual produces one stable matching, stable matchings need not be unique. For example, reversing the roles of men and women will often yield a different stable matching among them. So a man may have different wives in different sets of stable marriages. In some cases, a man can stably marry every one of the woman, but in most cases, there are some woman who cannot be a man's wife in any stable matching. For example, given the preferences shown in Figure 11.11, Jennifer cannot be Brad's wife in any stable matching because if he was married to her, then he and Angelina would be a rogue couple. It is not feasible for Jennifer to be stably married to Brad.

Definition 11.6.6

Given a set of preferences for the men and women, one person is a *feasible spouse* for another person when there is a stable matching in which these two people are married.

Definition 11.6.7

Let Q be the predicate: for every woman, w, and man, m, if w is crossed off m's list, then w is not a feasible spouse for m.

Lemma 11.6.8. *Q is a preserved invariant for The Mating Ritual.*

Proof. Suppose Q holds at some point in the Ritual and some woman, Alice, is about to be crossed off some man's, Bob's, list. We claim that Alice must not be feasible for Bob. Therefore Q will still hold after Alice is crossed off, proving that Q is invariant

To verify the claim, notice that when Alice gets crossed of Bob's list, it's because Alice has a suitor, Ted, she prefers to Bob. What's more since Q holds, all Ted's feasible wives are still on his list, and Alice is at the top. So Ted likes Alice better than all his other feasible spouses. Now if Alice could be married to Bob in some set of stable marriages, then Ted must be married to a wife he likes less than Alice, making Alice and Ted a rogue couple and contradicting stability. So Alice can't be married to Bob, that is, Alice is not a feasible wife for Bob, as claimed. ■

Definition 11.6.9

A person's *optimal spouse* is their most preferred feasible spouse. A person's *pessimal spouse* is their least preferred feasible spouse.

Everybody has an optimal and a pessimal spouse, since we know there is at least one stable matching, namely, the one produced by the Mating Ritual. Lemma 11.6.8 implies a key property the Mating Ritual:

Theorem 11.6.10

The Mating Ritual marries every man to his optimal spouse and every woman to her pessimal spouse.

Proof

If Bob is married to Alice on the final day of the Ritual, then everyone above Alice on Bob's preference list was crossed off, and by property Q, all these crossed off women were infeasible for Bob. So Alice is Bob's highest ranked feasible spouse, that is, his optimal spouse.

Further, since Bob likes Alice better than any other feasible wife, Alice and Bob would be a rogue couple if Alice was married to a husband she liked less than Bob. So Bob must be Alice's least preferred feasible husband. ■

Applications

The Mating Ritual was first announced in a paper by D. Gale and L.S. Shapley in 1962, but ten years before the Gale-Shapley paper was published, and unknown to them, a similar algorithm was being used to assign residents to hospitals by the National Resident Matching Program (NRMP). The NRMP has, since the turn of the twentieth century, assigned each year's pool of medical school graduates to hospital residencies (formerly called "internships"), with hospitals and graduates playing the roles of men and women.[7] Before the Ritual-like algorithm was adopted, there were chronic disruptions and awkward countermeasures taken to preserve unstable assignments of graduates to residencies. The Ritual resolved these problems so successfully, that it was used essentially without change at least through 1989.[8] For this and related work, Shapley was awarded the 2012 Nobel prize in Economics.

Not surprisingly, the Mating Ritual is also used by at least one large online dating agency. Of course there is no serenading going on—everything is handled by computer.

[6]Once again, we disclaim any political statement here—it's just the way that the math works out.

[7]In this case there may be multiple women married to one man, but this is a minor complication, see Problem 11.23.

[8]Much more about the Stable Marriage Problem can be found in the very readable mathematical monograph by Dan Gusfield and Robert W. Irving, [24].

This page titled 11.6: The Stable Marriage Problem is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.7: Coloring

In Section 11.2, we used edges to indicate an affinity between a pair of nodes. But there are lots of situations in which edges will correspond to *conflicts* between nodes. Exam scheduling is a typical example.

Exam Scheduling Problem

Each term, the MIT Schedules Office must assign a time slot for each final exam. This is not easy, because some students are taking several classes with finals, and (even at MIT) a student can take only one test during a particular time slot. The Schedules Office wants to avoid all conflicts. Of course, you can make such a schedule by having every exam in a different slot, but then you would need hundreds of slots for the hundreds of courses, and the exam period would run all year! So, the Schedules Office would also like to keep exam period short.

The Schedules Office's problem is easy to describe as a graph. There will be a vertex for each course with a final exam, and two vertices will be adjacent exactly when some student is taking both courses. For example, suppose we need to schedule exams for 6.041, 6.042, 6.002, 6.003 and 6.170. The scheduling graph might appear as in Figure 11.12.

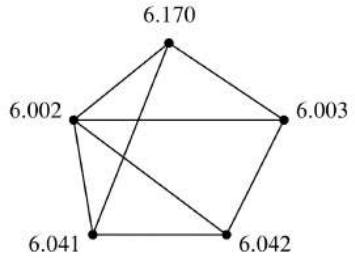


Figure 11.12 A scheduling graph for five exams. Exams connected by an edge cannot be given at the same time.

6.002 and 6.042 cannot have an exam at the same time since there are students in both courses, so there is an edge between their nodes. On the other hand, 6.042 and 6.170 can have an exam at the same time if they're taught at the same time (which they sometimes are), since no student can be enrolled in both (that is, no student *should* be enrolled in both when they have a timing conflict).

We next identify each time slot with a color. For example, Monday morning is red, Monday afternoon is blue, Tuesday morning is green, etc. Assigning an exam to a time slot is then equivalent to coloring the corresponding vertex. The main constraint is that *adjacent vertices must get different colors*—otherwise, some student has two exams at the same time. Furthermore, in order to keep the exam period short, we should try to color all the vertices using as *few different colors as possible*. As shown in Figure 11.13, three colors suffice for our example.

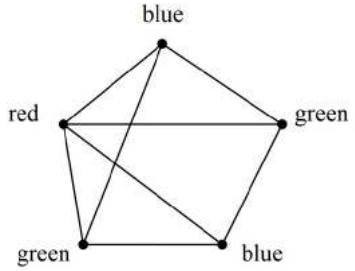


Figure 11.13 A 3-coloring of the exam graph from Figure 11.12.

The coloring in Figure 11.13 corresponds to giving one final on Monday morning (red), two Monday afternoon (blue), and two Tuesday morning (green). Can we use fewer than three colors? No! We can't use only two colors since there is a triangle in the graph, and three vertices in a triangle must all have different colors.

This is an example of a *graph coloring* problem: given a graph G, assign colors to each node such that adjacent nodes have different colors. A color assignment with this property is called a *valid coloring* of the graph—a "*coloring*," for short. A graph G is k*-colorable* if it has a coloring that uses at most k colors.

Definition 11.7.1

The minimum value of k for which a graph, G, has a valid coloring is called its *chromatic number*, $\chi(G)$.

So G is k-colorable iff $\chi(G) \leq k$.

In general, trying to figure out if you can color a graph with a fixed number of colors can take a long time. It's a classic example of a problem for which no fast algorithms are known. In fact, it is easy to check if a coloring works, but it seems really hard to find it. (If you figure out how, then you can get a $1 million Clay prize.)

Some Coloring Bounds

There are some simple properties of graphs that give useful bounds on colorability. The simplest property is being a cycle: an even-length closed cycle is 2-colorable. Cycles in simple graphs by convention have positive length and so are not 1- colorable. So

$$\chi(C_{even}) = 2.$$

On the other hand, an odd-length cycle requires 3 colors, that is,

$$\chi(C_{odd}) = 3. \tag{11.7.1}$$

You should take a moment to think about why this equality holds.

Another simple example is a complete graph K_n:

$$\chi(K_n) = n$$

since no two vertices can have the same color.

Being bipartite is another property closely related to colorability. If a graph is bipartite, then you can color it with 2 colors using one color for the nodes on the "left" and a second color for the nodes on the "right." Conversely, graphs with chromatic number 2 are all bipartite with all the vertices of one color on the "left" and those with the other color on the right. Since only graphs with no edges—the empty graphs—have chromatic number 1, we have:

Lemma 11.7.2. A graph, G, with at least one edge is bipartite iff $\chi(G) = 2$.

The chromatic number of a graph can also be shown to be small if the vertex degrees of the graph are small. In particular, if we have an upper bound on the degrees of all the vertices in a graph, then we can easily find a coloring with only one more color than the degree bound.

Theorem 11.7.4

A graph with maximum degree at most k is $k+1$-colorable.

Since k is the only nonnegative integer valued variable mentioned in the theorem, you might be tempted to try to prove this theorem using induction on k. Unfortunately, this approach leads to disaster—we don't know of any reasonable way to do this and expect it would ruin your week if you tried it on a problem set. When you encounter such a disaster using induction on graphs, it is usually best to change what you are inducting on. In graphs, typical good choices for the induction parameter are n, the number of nodes, or e, the number of edges.

Proof

We use induction on the number of vertices in the graph, which we denote by n. Let $P(n)$ be the proposition that an n-vertex graph with maximum degree at most k is $k+1$-colorable.

Base case ($n = 1$): A 1-vertex graph has maximum degree 0 and is 1-colorable, so $P(1)$ is true.

Inductive step: Now assume that $P(n)$ is true, and let G be an $(n+1)$-vertex graph with maximum degree at most k. Remove a vertex v (and all edges incident to it), leaving an n-vertex subgraph, H. The maximum degree of H is at most k, and so H is $(k+1)$-colorable by our assumption $P(n)$. Now add back vertex v. We can assign v a color (from the set of $(k+1)$ colors) that is different from all its adjacent vertices, since there are at most k vertices adjacent to v and so at least one

of the $(k+1)$ colors is still available. Therefore, G is $(k+1)$-colorable. This completes the inductive step, and the theorem follows by induction. ■

Sometimes $k+1$ colors is the best you can do. For example, $\chi(K_n) = n$and every node in K_n has degree $k = n-1$ and so this is an example where Theorem 11.7.3 gives the best possible bound. By a similar argument, we can show that Theorem 11.7.3 gives the best possible bound for *any* graph with degree bounded by k that has K_{k+1} as a subgraph.

But sometimes $k+1$ colors is far from the best that you can do. For example, the n-node *star graph* shown in Figure 11.14 has maximum degree $n-1$ but can be colored using just 2 colors.

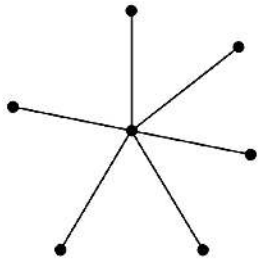


Figure 11.14 A 7-node star graph.

Why coloring?

One reason coloring problems frequently arise in practice is because scheduling conflicts are so common. For example, at Akamai, a new version of software is deployed over each of 65,000 servers every few days. The updates cannot be done at the same time since the servers need to be taken down in order to deploy the software. Also, the servers cannot be handled one at a time, since it would take forever to update them all (each one takes about an hour). Moreover, certain pairs of servers cannot be taken down at the same time since they have common critical functions. This problem was eventually solved by making a 65,000-node conflict graph and coloring it with 8 colors—so only 8 waves of install are needed!

Another example comes from the need to assign frequencies to radio stations. If two stations have an overlap in their broadcast area, they can't be given the same frequency. Frequencies are precious and expensive, so you want to minimize the number handed out. This amounts to finding the minimum coloring for a graph whose vertices are the stations and whose edges connect stations with overlapping areas.

Coloring also comes up in allocating registers for program variables. While a variable is in use, its value needs to be saved in a register. Registers can be reused for different variables but two variables need different registers if they are referenced during overlapping intervals of program execution. So register allocation is the coloring problem for a graph whose vertices are the variables: vertices are adjacent if their intervals overlap, and the colors are registers. Once again, the goal is to minimize the number of colors needed to color the graph.

Finally, there's the famous map coloring problem stated in Proposition 1.1.6. The question is how many colors are needed to color a map so that adjacent territories get different colors? This is the same as the number of colors needed to color a graph that can be drawn in the plane without edges crossing. A proof that four colors are enough for *planar* graphs was acclaimed when it was discovered about thirty years ago. Implicit in that proof was a 4-coloring procedure that takes time proportional to the number of vertices in the graph (countries in the map).

Surprisingly, it's another of those million dollar prize questions to find an efficient procedure to tell if a planar graph really *needs* four colors, or if three will actually do the job. A proof that testing 3-colorability of graphs is as hard as the million dollar SAT problem is given in Problem 11.39; this turns out to be true even for planar graphs. (It is easy to tell if a graph is 2-colorable, as explained in Section 11.9.2.) In Chapter 12, we'll develop enough planar graph theory to present an easy proof that all planar graphs are 5-colorable.

This page titled 11.7: Coloring is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.8: Simple Walks

Walks, Paths, Cycles in Simple Graphs

Walks and paths in simple graphs are esentially the same as in digraphs. We just modify the digraph definitions using undirected edges instead of directed ones. For example, the formal definition of a walk in a simple graph is a virtually the same as the Definition 9.2.1 of a walk in a digraph:

Definition 11.8.1

A walk in a *simple graph*, G, is an alternating sequence of vertices and edges that begins with a vertex, ends with a vertex, and such that for every edge $\langle u - v\rangle$ in the walk, one of the endpoints u, v is the element just before the edge, and the other endpoint is the next element after the edge. The *length of a walk* is the total number of occurrences of edges in it.

So a walk, $\mathbf{v}$, is a sequence of the form

where $\langle v_i - v_{i+1}\rangle \in E(G)$ for $i \in [0..k)$. The walk is said to *start* at v_0, to *end* at v_k, and the *length*, $|\mathbf{v}|$, of the walk is k. The walk is a *path* iff all the v_i's are different, that is, if $i \neq j$, then $v_i \neq v_j$.

A *closed walk* is a walk that begins and ends at the same vertex. A single vertex counts as a length zero closed walk as well as a length zero path.

A *cycle* is a closed walk of length three or more whose vertices are distinct except for the beginning and end vertices

Note that in contrast to digraphs, we don't count length two closed walks as cycles in simple graphs. That's because a walk going back and forth on the same edge is always possible in a simple graph, and it has no importance. Also, there are no closed walks of length one, since simple graphs don't have self loops.

As in digraphs, the length of a walk is *one less* than the number of occurrences of vertices in it. For example, the graph in Figure 11.15 has a length 6 path through the seven successive vertices \(abcdefg\). This is the longest path in the graph. The graph in Figure 11.15 also has three cycles through successive vertices $bhecb$, $cdec$, and $bcdehb$.

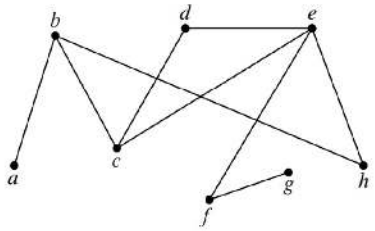


Figure 11.15 *A graph with 3 cycles:* $bhecb, cdec, bcdehb$.

Cycles as Subgraphs

A cycle does not really have a beginning or an end, so it can be described by *any* of the paths that go around it. For example, in the graph in Figure 11.15, the cycle starting at b and going through vertices $bcdehb$ can also be described as starting at d and going through $dehbcd$. Furthermore, cycles in simple graphs don't have a direction: $dcbhed$ describes the same cycle as though it started and ended at d but went in the opposite direction.

A precise way to explain which closed walks describe the same cycle is to define cycle as a subgraph instead of as a closed walk. Specifically, we could define a cycle in G to be a *subgraph* of G that looks like a length-n cycle for $n \geq 3$.

Definition 11.8.2

A graph G is said to be a *subgraph* of a graph H if $V(G) \subseteq V(H)$ and $E(G) \subseteq E(H)$.

For example, the one-edge graph G where

$$V(G) = \{g, h, i\} \text{ and } E(G) = \{\langle h - i\rangle\}$$

is a subgraph of the graph H in Figure 11.1. On the other hand, any graph containing an edge $\langle g - h \rangle$ will not be a subgraph of H because this edge is not in $E(H)$. Another example is an empty graph on n nodes, which will be a subgraph of an L_n with the same set of nodes; similarly, L_n is a subgraph of C_n, and $C_n$is a subgraph of K_n.

Definition 11.8.3

For $n \geq 3$, let C_n be the graph with vertices $1, \ldots, n$ and edges

$$\langle 1 - 2 \rangle, \langle 2 - 3 \rangle, \ldots, \langle (n-1) - n \rangle, \langle n - 1 \rangle.$$

A cycle of a graph, G, is a subgraph of G that is isomorphic to C_n for some $n \geq 3$.

This definition formally captures the idea that cycles don't have direction or beginnings or ends.

This page titled 11.8: Simple Walks is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.9: Connectivity

Definition 11.9.1

Two vertices are connected in a graph when there is a path that begins at one and ends at the other. By convention, every vertex is connected to itself by a path of length zero. A *graph is connected* when every pair of vertices are connected.

Connected Components

Being connected is usually a good property for a graph to have. For example, it could mean that it is possible to get from any node to any other node, or that it is possible to communicate between any pair of nodes, depending on the application.

But not all graphs are connected. For example, the graph where nodes represent cities and edges represent highways might be connected for North American cities, but would surely not be connected if you also included cities in Australia. The same is true for communication networks like the internet—in order to be protected from viruses that spread on the internet, some government networks are completely isolated from the internet.

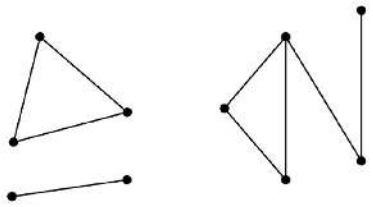


Figure 11.16 One graph with 3 connected components.

Another example is shown in Figure 11.16, which looks like a picture of three graphs, but is intended to be a picture of *one* graph. This graph consists of three pieces (subgraphs). Each piece by itself is connected, but there are no paths between vertices in different pieces. These connected pieces of a graph are called its *connected components*.

Definition 11.9.2

A *connected component* of a graph is a subgraph consisting of some vertex and every node and edge that is connected to that vertex.

So, a graph is connected iff it has exactly one connected component. At the other extreme, the empty graph on n vertices has n connected components.

Odd Cycles and 2-Colorability

We have already seen that determining the chromatic number of a graph is a challenging problem. There is one special case where this problem is very easy, namely, when the graph is 2-colorable

Theorem 11.9.3

The following graph properties are equivalent:

1. *The graph contains an odd length cycle.*
2. *The graph is not 2-colorable.*
3. *The graph contains an odd length closed walk.*

In other words, if a graph has any one of the three properties above, then it has all of the properties.

We will show the following implications among these properties:

1. IMPLIES 2. IMPLIES 3. IMPLIES 1:

So each of these properties implies the other two, which means they all are equivalent.

1 **IMPLIES** 2 *Proof*. This follows from equation 11.7.1. ■

2 **IMPLIES** 3 If we prove this implication for connected graphs, then it will hold for an arbitrary graph because it will hold for each connected component. So we can assume that G is connected.

Proof. Pick an arbitrary vertex r of G. Since G is connected, for every node $u \in V(G)$, there will be a walk $\mathbf{w}_u$ starting at u and ending at r. Assign colors to vertices of G as follows:

$$\text{color}(u) = \begin{cases} \text{black}, & \text{if } |\mathbf{w}_u| \text{ is even,} \\ \text{white}, & \text{otherwise.} \end{cases}$$

Now since G is not colorable, this can't be a valid coloring. So there must be an edge between two nodes u and v with the same color. But in that case

$$\mathbf{w}_u \hat{} \text{reverse}(\mathbf{w}_u) \hat{} \langle v - u \rangle$$

is a closed walk starting and ending at u, and its length is

$$|\mathbf{w}_u| + |\mathbf{w}_v| + 1$$

which is odd. ■

3 **IMPLIES** 1 *Proof.* Since there is an odd length closed walk, the WOP implies there is an odd length closed walk w of minimum length. We claim w must be a cycle. To show this, assume to the contrary that w is not a cycle, so there is a repeat vertex occurrence besides the start and end. There are then two cases to consider depending on whether the additional repeat is different from, or the same as, the start vertex.

In the first case, the start vertex has an extra occurrence. That is,

$$\mathbf{w} = \mathbf{f} \hat{x} \mathbf{r}$$

for some positive length walks $\mathbf{f}$ and $\mathbf{r}$ that begin and end at x. Since

$$|\mathbf{w}| = |\mathbf{f}| + |\mathbf{r}|$$

is odd, exactly one of $\mathbf{f}$ and $\mathbf{r}$ must have odd length, and that one will be an odd length closed walk shorter than $\mathbf{w}$, a contradiction.

In the second case,

$$\mathbf{w} = \mathbf{f} \hat{y} \mathbf{g} \hat{y} \mathbf{r}$$

where $\mathbf{f}$ is a walk from x to y for some $y \neq x$, and $\mathbf{r}$ is a walk from y to x, and $|\mathbf{g}| > 0$. Now $\mathbf{g}$ cannot have odd length or it would be an odd-length closed walk shorter than $\mathbf{w}$. So $\mathbf{g}$ has even length. That implies that $\mathbf{f} \hat{y} \mathbf{r}$ must be an odd-length closed walk shorter than w, again a contradiction.

This completes the proof of Theorem 11.9.3. ■

Theorem 11.9.3 turns out to be useful, since bipartite graphs come up fairly often in practice. We'll see examples when we talk about planar graphs in Chapter 12.

k-connected Graphs

If we think of a graph as modeling cables in a telephone network, or oil pipelines, or electrical power lines, then we not only want connectivity, but we want connectivity that survives component failure. So more generally, we want to define how strongly two vertices are connected. One measure of connection strength is how many links must fail before connectedness fails. In particular, two vertices are *k-edge connected* when it takes at least k "edge-failures" to disconnect them. More precisely:

Definition 11.9.4

Two vertices in a graph are *k-edge connected* when they remain connected in every subgraph obtained by deleting up to $k-1$ edges. A graph is k-edge connected when it has more than one vertex, and pair of distinct vertices in the graph are k-connected.

Notice that according to Definition 11.9.4, if a graph is k-connected, it is also j-connected for $j \leq k$. This convenient convention implies that two vertices are connected according to definition 11.9.1 iff they are 1-edge connected according to Definition 11.9.4. From now on we'll drop the "edge" modifier and just say "k-connected."[9]

For example, in the graph in figure 11.15, vertices c and e are 3-connected, b and e are 2-connected, g and e are 1 connected, and no vertices are 4-connected. The graph as a whole is only 1-connected. A complete graph, k_n, is $n-1$- connected. Every cycle is 2-connected.

The idea of a *cut edge* is a useful way to explain 2-connectivity.

Definition 11.9.5

If two vertices are connected in a graph G, but not connected when an edge e is removed, then e is called a *cut edge* of G.

So a graph with more than one vertex is 2-connected iff it is connected and has no cut edges. The following Lemma is another immediate consequence of the definition:

Lemma 11.9.6. *An edge is a cut edge iff it is not on a cycle.*

More generally, if two vertices are connected by k edge-disjoint paths—that is, no edge occurs in two paths—then they must be k-connected, since at least one edge will have to be removed from each of the paths before they could disconnect. A fundamental fact, whose ingenious proof we omit, is Menger's theorem which confirms that the converse is also true: if two vertices are k-connected, then there are k edge-disjoint paths connecting them. It takes some ingenuity to prove this just for the case $k = 2$.

The Minimum Number of Edges in a Connected Graph

The following theorem says that a graph with few edges must have many connected components.

Theorem 11.9.7

*Every graph, G, has at least $|V(G)| - |E(G)|$ connected component*s.

Of course for Theorem 11.9.7 to be of any use, there must be fewer edges than vertices.

Proof

We use induction on the number, k, of edges. Let $P(k)$ be the proposition that

every graph, G, with k edges has at least $|V(G)| - k$ connected components.

Base case ($k = 0$): In a graph with 0 edges, each vertex is itself a connected component, and so there are exactly $|V(G)| = |V(G)| - 0$ connected components. So $P(0)$ holds.

Inductive step: Let G_e be the graph that results from removing an edge, e *in* $E(G)$. So G_e has k edges, and by the induction hypothesis $P(k)$, we may assume that G_e has at least $|V(G) - k|$-connected components. Now add back the edge e to obtain the original graph G. If the endpoints of e were in the same connected component of G_e, then G has the same sets of connected vertices as G_e, so G has at least $|V(G)| - k > |V(G)| - (k+1)$ components. Alternatively, if the endpoints of e were in different connected components of G_e, then these two components are merged into one component in G, while all other components remain unchanged, so that G has one fewer connected component than G_e. That is, G has at least $(|V(G)| - k) - 1 > |V(G)| - (k+1)$ connected components. So in either case, G has at least $|V(G)| - (k+1)$ components, as claimed.

This completes the inductive step and hence the entire proof by induction. \blacksqure

Corollary 11.9.8. Every connected graph with n vertices has at least $n-1$ edges.

A couple of points about the proof of Theorem 11.9.7 are worth noticing. First, we used induction on the number of edges in the graph. This is very common in proofs involving graphs, as is induction on the number of vertices. When you're presented with a graph problem, these two approaches should be among the first you consider.

The second point is more subtle. Notice that in the inductive step, we took an arbitrary $(k+1)$-edge graph, threw out an edge so that we could apply the induction assumption, and then put the edge back. You'll see this shrink-down, grow-back process very often in the inductive steps of proofs related to graphs. This might seem like needless effort: why not start with an k-edge graph and add one more to get an $(k+1)$-edge graph? That would work fine in this case, but opens the door to a nasty logical error called *buildup error*, illustrated in Problem 11.48.

[9]There is a corresponding definition of k-vertex connectedness based on deleting vertices rather than edges. Graph theory texts usually use "k-connected" as shorthand for "k-vertex connected." But edge-connectedness will be enough for us.

This page titled 11.9: Connectivity is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

11.10: Forests and Trees

We've already made good use of digraphs without cycles, but *simple* graphs without cycles are arguably the most important graphs in computer science.

Leaves, Parents & Children

Definition: 11.10.1

An acyclic graph is called a *forest*. A connected acyclic graph is called a *tree*.

The graph shown in Figure 11.17 is a forest. Each of its connected components is by definition a tree.

One of the first things you will notice about trees is that they tend to have a lot of nodes with degree one. Such nodes are called *leaves*.

Definition: 11.10.2

A degree 1 node in a forest is called a *leaf*.

The forest in Figure 11.17 has 4 leaves. The tree in Figure 11.18 has 5 leaves.

Trees are a fundamental data structure in computer science. For example, information is often stored in tree-like data structures, and the execution of many recursive programs can be modeled as the traversal of a tree. In such cases, it is often useful to arrange the nodes in levels, where the node at the top level is identified as the *root* and where every edge joins a *parent* to a *child* one level below. Figure 11.19 shows the tree of Figure 11.18 redrawn in this way. Node d is a child of node e and the parent of nodes b and c.

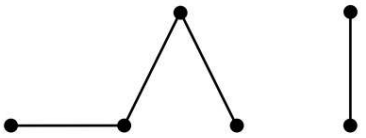


Figure 11.17 A 6-node forest consisting of 2 component trees.

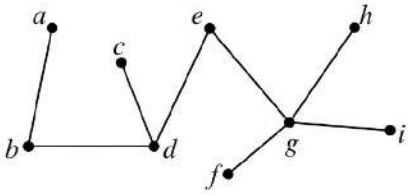


Figure 11.18 A 9-node tree with 5 leaves.

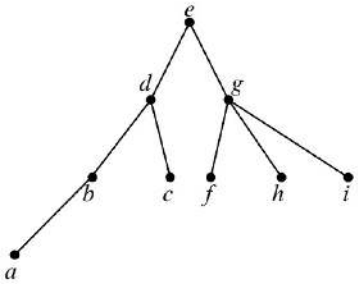


Figure 11.19 The tree from Figure 11.18 redrawn with node e as the root and the other nodes arranged in levels.

Properties

Trees have many unique properties. We have listed some of them in the following theorem.

Theorem 11.10.3

Every tree has the following properties:

1. *Every connected subgraph is a tree.*
2. *There is a unique path between every pair of vertices.*
3. *Adding an edge between nonadjacent nodes in a tree creates a graph with a cycle.*
4. *Removing any edge disconnects the graph. That is, every edge is a cut edge.*
5. *If the tree has at least two vertices, then it has at least two leaves.*
6. *The number of vertices in a tree is one larger than the number of edges.*

Proof

1. A cycle in a subgraph is also a cycle in the whole graph, so any subgraph of an acyclic graph must also be acyclic. If the subgraph is also connected, then by definition, it is a tree.
2. Since a tree is connected, there is at least one path between every pair of vertices. Suppose for the purposes of contradiction, that there are two different paths between some pair of vertices. Then there are two distinct paths $\mathbf{p} \neq \mathbf{q}$ between the same two vertices with minimum total length $|\mathbf{p}| + |\mathbf{q}|$. If these paths shared a vertex, w, other than at the start and end of the paths, then the parts of $\mathbf{p}$ and $\mathbf{q}$ from start to w, or the parts of $\mathbf{p}$ and $\mathbf{q}$ from w to the end, must be distinct paths between the same vertices with total length less than $|\mathbf{p}| + |\mathbf{q}|$, contradicting the minimality of this sum. Therefore, $\mathbf{p}$ and $\mathbf{q}$ have no vertices in common besides their endpoints, and so $\hat{\mathbf{p}}\mathbf{reverse}(\mathbf{q})$ is a cycle.
3. An additional edge b $\langle u - v\rangle$ together with the unique path between u and v forms a cycle.
4. Suppose that we remove edge $\langle u - v\rangle$. Since the tree contained a unique path between u and v, that path must have been $\langle u - v\rangle$. Therefore, when that edge is removed, no path remains, and so the graph is not connected.
5. Since the tree has at least two vertices, the longest path in the tree will have different endpoints u and v. We claim u is a leaf. This follows because, since by definition of endpoint, u is incident to at most one edge on the path. Also, if u was incident to an edge not on the path, then the path could be lengthened by adding that edge, contradicting the fact that the path was as long as possible. It follows that u is incident only to a single edge, that is u is a leaf. The same hold for v.
6. We use induction on the proposition

$$P(n) ::= \text{there are } n-1 \text{ edges in any } n\text{–vertex tree.}$$

Base case ($n = 1$): $P(n)$ is true since a tree with 1 node has 0 edges and $1 - 1 = 0$.

Inductive step: Now suppose that $P(n)$ is true and consider an $n+1$-vertex tree, T. Let v be a leaf of the tree. You can verify that deleting a vertex of degree 1 (and its incident edge) from any connected graph leaves a connected subgraph. So by Theorem 11.10.3.1, deleting v and its incident edge gives a smaller tree, and this smaller tree has $n-1$ edges by induction. If we reattach the vertex, v, and its incident edge, we find that T has $n = (n+1) - 1$ edges. Hence, $P(n+1)$ is true, and the induction proof is complete. ■

Various subsets of properties in Theorem 11.10.3 provide alternative characterizations of trees. For example,

Lemma 11.10.4. A graph G is a tree iff G is a forest and $|V(G)| = |E(G)| + 1$.

The proof is an easy consequence of Theorem 11.9.7.6 (Problem 11.55).

Spanning Trees

Trees are everywhere. In fact, every connected graph contains a subgraph that is a tree with the same vertices as the graph. This is called a spanning tree for the graph. For example, Figure 11.20 is a connected graph with a *spanning tree* highlighted.

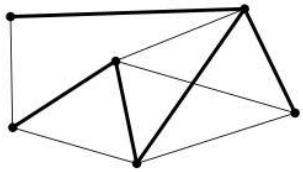


Figure 11.20 A graph where the edges of a spanning tree have been thickened.

Definition 11.10.5

Define a *spanning subgraph* of a graph, G, to be a subgraph containing all the vertices of G.

Theorem 11.10.6

Every connected graph contains a spanning tree.

Proof

Suppose G is a connected graph, so the graph G itself is a connected, spanning subgraph. So by WOP, G must have a minimum-edge connected, spanning subgraph, T. We claim T is a spanning tree. Since T is a connected, spanning subgraph by definition, all we have to show is that T is acyclic.

But suppose to the contrary that T contained a cycle C. By Lemma 11.9.6, an edge e of C will not be a cut edge, so removing it would leave a connected, spanning subgraph that was smaller than C, contradicting the minimality to T. ■

Minimum Weight Spanning Trees

Spanning trees are interesting because they connect all the nodes of a graph using the smallest possible number of edges. For example the spanning tree for the 6- node graph shown in Figure 11.20 has 5 edges.

In many applications, there are numerical costs or weights associated with the edges of the graph. For example, suppose the nodes of a graph represent buildings and edges represent connections between them. The cost of a connection may vary a lot from one pair of buildings or towns to another. Another example is where the nodes represent cities and the weight of an edge is the distance between them: the weight of the Los Angeles/New York City edge is much higher than the weight of the NYC/Boston edge. The *weight of a graph* is simply defined to be the sum of the weights of its edges. For example, the weight of the spanning tree shown in Figure 11.21 is 19.

Definition 11.10.7

A *minimum weight spanning tree* (MST) of an edge-weighted graph G is a spanning tree of G with the smallest possible sum of edge weights.

Is the spanning tree shown in Figure 11.21(a) an MST of the weighted graph shown in Figure 11.21(b)? It actually isn't, since the tree shown in Figure 11.22 is also a spanning tree of the graph shown in Figure 11.21(b), and this spanning tree has weight 17.

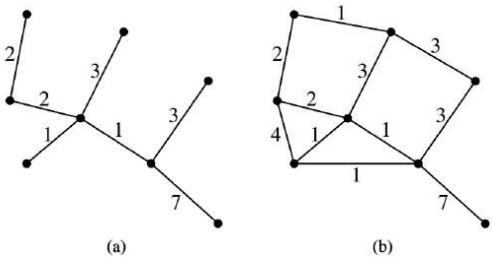


Figure 11.21 A spanning tree (a) with weight 19 for a graph (b).

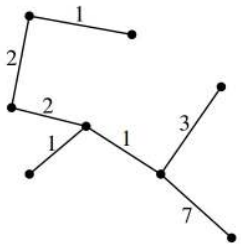


Figure 11.22 An MST with weight 17 for the graph in Figure 11.21(b).

What about the tree shown in Figure 11.22? It seems to be an MST, but how do we prove it? In general, how do we find an MST for a connected graph G? We could try enumerating all subtrees of G, but that approach would be hopeless for large graphs.

There actually are many good ways to find MST's based on a property of some subgraphs of G called *pre-MST*'s.

Definition 11.10.8

A *pre-MST* for a graph G is a spanning subgraph of G that is also a subgraph of some MST of G.

So a pre-MST will necessarily be a forest.

For example, the empty graph with the same vertices as G is guaranteed to be a pre-MST of G, and so is any actual MST of G.

If e is an edge of G and S is a spanning subgraph, we'll write $S + e$ for the spanning subgraph with edges $E(S) \cup \{e\}$.

Definition 11.10.9

If F is a pre-MST and e is a new edge, that is $e \in E(G) - E(F)$, then e *extends* F when $F + e$ is also a pre-MST.

So being a pre-MST is contrived to be an invariant under addition of extending edges, by the definition of extension.

The standard methods for finding MST's all start with the empty spanning forest and build up to an MST by adding one extending edge after another. Since the empty spanning forest is a pre-MST, and being a pre-MST is, by definition, invariant under extensions, every forest built in this way will be a pre-MST. But no spanning tree can be a subgraph of a different spanning tree. So when the pre-MST finally grows enough to become a tree, it will be an MST. By Lemma 11.10.4, this happens after exactly $|V(G)| - 1$ edge extensions.

So the problem of finding MST's reduces to the question of how to tell if an edge is an extending edge. Here's how:

Definition 11.10.10

Let F be a pre-MST, and color the vertices in each connected component of F either all black or all white. At least one component of each color is required. Call this a *solid coloring* of F. A *gray edge* of a solid coloring is an edge of G with different colored endpoints.

Any path in G from a white vertex to a black vertex obviously must include a gray edge, so for any solid coloring, there is guaranteed to be at least one gray edge. In fact, there will have to be at least as many gray edges as there are components with the same color. Here's the punchline:

Lemma 11.10.11. *An edge extends a pre-MST F if it is a minimum weight gray edge in some solid coloring of F.*

So to extend a pre-MST, choose any solid coloring, find the gray edges, and among them choose one with minimum weight. Each of these steps is easy to do, so it is easy to keep extending and arrive at an MST. For example, here are three known algorithms that are explained by Lemma 11.10.11:

Algorithm 1. [Prim] *Grow a tree one edge at a time by adding a minimum weight edge among the edges that have exactly one endpoint in the tree.*

This is the algorithm that comes from coloring the growing tree white and all the vertices not in the tree black. Then the gray edges are the ones with exactly one endpoint in the tree.

Algorithm 2. *[Kruskal] Grow a forest one edge at a time by adding a minimum weight edge among the edges with endpoints in different connected components.*

An edge does not create a cycle iff it connects different components. The edge chosen by Kruskal's algorithm will be the minimum weight gray edge when the components it connects are assigned different colors.

For example, in the weighted graph we have been considering, we might run Algorithm 1 as follows. Start by choosing one of the weight 1 edges, since this is the smallest weight in the graph. Suppose we chose the weight 1 edge on the bottom of the triangle of weight 1 edges in our graph. This edge is incident to the same vertex as two weight 1 edges, a weight 4 edge, a weight 7 edge, and a weight 3 edge. We would then choose the incident edge of minimum weight. In this case, one of the two weight 1 edges. At this point, we cannot choose the third weight 1 edge: it won't be gray because its endpoints are both in the tree, and so are both colored white. But we can continue by choosing a weight 2 edge. We might end up with the spanning tree shown in Figure 11.23, which has weight 17, the smallest we've seen so far.

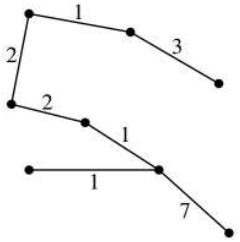


Figure 11.23 A spanning tree found by Algorithm 1.

Now suppose we instead ran Algorithm 2 on our graph. We might again choose the weight 1 edge on the bottom of the triangle of weight 1 edges in our graph. Now, instead of choosing one of the weight 1 edges it touches, we might choose the weight 1 edge on the top of the graph. This edge still has minimum weight, and will be gray if we simply color its endpoints differently, so Algorithm 2 can choose it. We would then choose one of the remaining weight 1 edges. Note that neither causes us to form a cycle. Continuing the algorithm, we could end up with the same spanning tree in Figure 11.23, though this will depend on the tie breaking rules used to choose among gray edges with the same minimum weight. For example, if the weight of every edge in G is one, then all spanning trees are MST's with weight $|V(G)| - 1$, and both of these algorithms can arrive at each of these spanning trees by suitable tie-breaking.

The coloring that explains Algorithm 1 also justifies a more flexible algorithm which has Algorithm 1 as a special case:

Algorithm 3. *Grow a forest one edge at a time by picking any component and adding a minimum weight edge among the edges leaving that component.*

This algorithm allows components that are not too close to grow in parallel and independently, which is great for "distributed" computation where separate processors share the work with limited communication between processors.

These are examples of greedy approaches to optimization. Sometimes greediness works and sometimes it doesn't. The good news is that it does work to find the MST. Therefore, we can be sure that the MST for our example graph has weight 17, since it was produced by Algorithm 2. Furthermore we have a fast algorithm for finding a minimum weight spanning tree for any graph.

Ok, to wrap up this story, all that's left is the proof that minimal gray edges are extending edges. This might sound like a chore, but it just uses the same reasoning we used to be sure there would be a gray edge when you need it.

Proof. (of Lemma 11.10.11)

Let F be a pre-MST that is a subgraph of some MST M of G, and suppose e is a minimum weight gray edge under some solid coloring of F. We want to show that $F + e$ is also a pre-MST.

If e happens to be an edge of M, then $F + e$ remains a subgraph of M, and so is a pre-MST.

The other case is when e is not an edge of M. In that case, $M + e$ will be a connected, spanning subgraph. Also M has a path $\mathbf{p}$ between the different colored endpoints of e, so $M + e$ has a cycle consisting of e together with $\mathbf{p}$. Now $\mathbf{p}$ has both a black endpoint and a white one, so it must contain some gray edge $g \neq e$. The trick is to remove g from $M + e$ to obtain a subgraph $M + e - g$.

Since gray edges by definition are not edges of F, the graph $M + e - g$ contains $F + e$. We claim that $M + e - g$ is an MST, which proves the claim that e extends F.

To prove this claim, note that $M + e$ is a connected, spanning subgraph, and g is on a cycle of $M + e$, so by Lemma 11.9.6, removing g won't disconnect anything. Therefore, $M + e - g$ is still a connected, spanning subgraph. Moreover, $M + e - g$ has the same number of edges as M, so Lemma 11.10.4 implies that it must be a spanning tree. Finally, since e is minimum weight among gray edges,

$$w(M + e - g) = w(M) + w(e) - w(g) \leq w(M).$$

This means that $M + e - g$ is a spanning tree whose weight is at most that of an MST, which implies that $M + e - g$ is also an MST.

■

Another interesting fact falls out of the proof of Lemma 11.10.11:

Corollary 11.10.12. *If all edges in a weighted graph have distinct weights, then the graph has a unique MST.*

The proof of Corollary 11.10.12 is left to Problem 11.70.

This page titled 11.10: Forests and Trees is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

12: Planar Graphs

This page titled 12: Planar Graphs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

12.1: Drawing Graphs in the Plane

Suppose there are three dog houses and three human houses, as shown in Figure 12.1. Can you find a route from each dog house to each human house such that no route crosses any other route?

A similar question comes up about a little-known animal called a *quadrapus* that looks like an octopus with four stretchy arms instead of eight. If five quadrapi are resting on the sea floor, as shown in Figure 12.2, can each quadrapus simultaneously shake hands with every other in such a way that no arms cross?

Both these puzzles can be understood as asking about drawing graphs in the plane. Replacing dogs and houses by nodes, the dog house puzzle can be rephrased as asking whether there is a planar drawing of the graph with six nodes and edges between each of the first three nodes and each of the second three nodes. This graph is called the *complete bipartite graph* $K_{3,3}$ and is shown in Figure 12.3.(a). The quadrapi puzzle asks whether there is a planar drawing of the complete graph K_5 shown in Figure 12.3.(b).

In each case, the answer is, “No —but almost!” In fact, if you remove an edge from either of these graphs, then the resulting graph *can* be redrawn in the plane so that no edges cross, as shown in Figure 12.4.

Planar drawings have applications in circuit layout and are helpful in displaying graphical data such as program flow charts, organizational charts, and scheduling conflicts. For these applications, the goal is to draw the graph in the plane with as few edge crossings as possible. (See the box on the following page for one such example.)

Steve Wozniak and a Planar Circuit Design

When wires are arranged on a surface, like a circuit board or microchip, crossings require troublesome three-dimensional structures. When Steve Wozniak designed the disk drive for the early Apple II computer, he struggled mightily to achieve a nearly planar design according to the following excerpt from apple2history.org which in turn quotes *Fire in the Valley* by Freiberger and Swaine:

For two weeks, he worked late each night to make a satisfactory design. When he was finished, he found that if he moved a connector he could cut down on feedthroughs, making the board more reliable. To make that move, however, he had to start over in his design. This time it only took twenty hours. He then saw another feedthrough that could be eliminated, and again started over on his design. “The final design was generally recognized by computer engineers as brilliant and was by engineering aesthetics beautiful. Woz later said, ’It’s something you can only do if you’re the engineer and the PC board layout person yourself. That was an artistic layout. The board has virtually no feedthroughs.’



Figure 12.1 Three dog houses and and three human houses. Is there a route from each dog house to each human house so that no pair of routes cross each other?

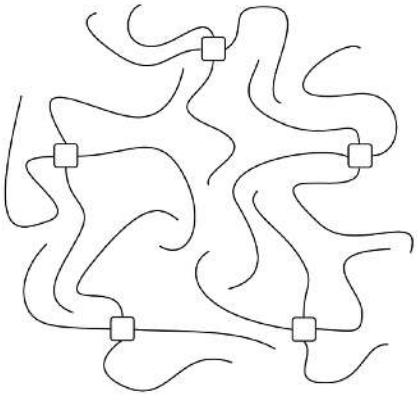


Figure 12.2 Five quadrapi (4-armed creatures).

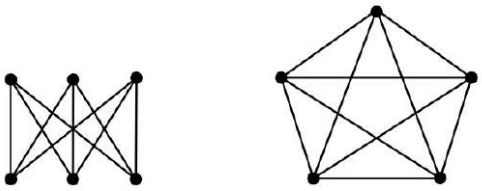


Figure 12.3 $K_{3,3}$ (a) and K5 (b). Can you redraw these graphs so that no pairs of edges cross?

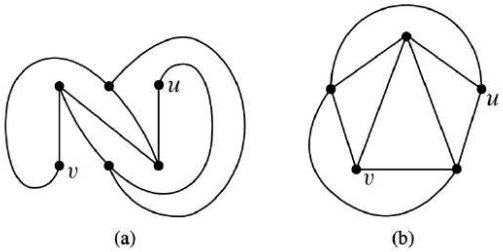


Figure 12.4 Planar drawings of (a) $K_{3,3}$ without $\langle u - v \rangle$, and (b) K_5 without $\langle u - v \rangle$.

This page titled 12.1: Drawing Graphs in the Plane is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

12.2: Definitions of Planar Graphs

We took the idea of a planar drawing for granted in the previous section, but if we're going to *prove* things about planar graphs, we better have precise definitions.

Definition 12.2.1

A *drawing* of a graph assigns to each node a distinct point in the plane and assigns to each edge a smooth curve in the plane whose endpoints correspond to the nodes incident to the edge. The drawing is *planar* if none of the curves cross themselves or other curves, namely, the only points that appear more than once on any of the curves are the node points. A graph is *planar* when it has a planar drawing.

Definition 12.2.1 is precise but depends on further concepts: "smooth planar curves" and "points appearing more than once" on them. We haven't defined these concepts —we just showed the simple picture in Figure 12.4 and hoped you would get the idea.

Pictures can be a great way to get a new idea across, but it is generally not a good idea to use a picture to replace precise mathematics. Relying solely on pictures can sometimes lead to disaster —or to bogus proofs, anyway. There is a long history of bogus proofs about planar graphs based on misleading pictures.

The bad news is that to prove things about planar graphs using the planar drawings of Definition 12.2.1, we'd have to take a chapter-long excursion into continuous mathematics just to develop the needed concepts from plane geometry and point-set topology. The good news is that there is another way to define planar graphs that uses only discrete mathematics. In particular, we can define planar graphs as a recursive data type. In order to understand how it works, we first need to understand the concept of a *face* in a planar drawing.

Faces

The curves in a planar drawing divide up the plane into connected regions called the *continuous faces*[1] of the drawing. For example, the drawing in Figure 12.5 has four continuous faces. Face IV, which extends off to infinity in all directions, is called the *outside face*.

The vertices along the boundary of each continuous face in Figure 12.5 form a cycle. For example, labeling the vertices as in Figure 12.6, the cycles for each of the face boundaries can be described by the vertex sequences

$$abca \quad abda \quad bcdb \quad acda. \tag{12.2.1}$$

These four cycles correspond nicely to the four continuous faces in Figure 12.6 — so nicely, in fact, that we can identify each of the faces in Figure 12.6 by its cycle. For example, the cycle $abca$ identifies face III. The cycles in list 12.2.1 are called the *discrete faces* of the graph in Figure 12.6. We use the term "discrete" since cycles in a graph are a discrete data type —as opposed to a region in the plane, which is a continuous data type.

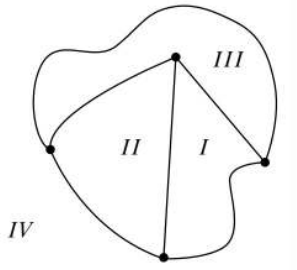


Figure 12.5 A planar drawing with four continuous faces.

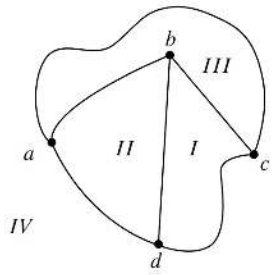


Figure 12.6 The drawing with labeled vertices.

Unfortunately, continuous faces in planar drawings are not always bounded by cycles in the graph —things can get a little more complicated. For example, the planar drawing in Figure 12.7 has what we will call a *bridge*, namely, a cut edge $\langle c - e \rangle$. The sequence of vertices along the boundary of the outer region of the drawing is

$$abcefgecda.$$

This sequence defines a closed walk, but does not define a cycle since the walk has two occurrences of the bridge $\langle c - e \rangle$ and each of its endpoints.

The planar drawing in Figure 12.8 illustrates another complication. This drawing has what we will call a *dongle*, namely, the nodes $v, x, y,$ and w, and the edges incident to them. The sequence of vertices along the boundary of the inner region is

$$rstvxyxvwvtur.$$

This sequence defines a closed walk, but once again does not define a cycle because it has two occurrences of *every* edge of the dongle —once "coming" and once "going."

It turns out that bridges and dongles are the only complications, at least for connected graphs. In particular, every continuous face in a planar drawing corresponds to a closed walk in the graph. These closed walks will be called the *discrete faces* of the drawing, and we'll define them next.

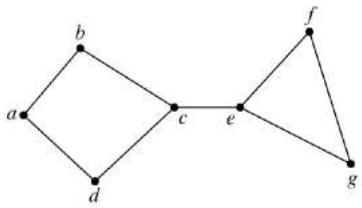


Figure 12.7 A planar drawing with a *bridge*.

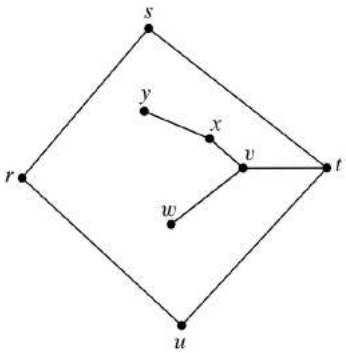


Figure 12.8 A planar drawing with a *dongle*.

Recursive Definition for Planar Embeddings

The association between the continuous faces of a planar drawing and closed walks provides the discrete data type we can use instead of continuous drawings. We'll define a *planar embedding* of *connected* graph to be the set of closed walks that are its face

boundaries. Since all we care about in a graph are the connections between vertices —not what a drawing of the graph actually looks like —planar embeddings are exactly what we need.

The question is how to define planar embeddings without appealing to continuous drawings. There is a simple way to do this based on the idea that any continuous drawing can drawn step by step:

- either draw a new point somewhere in the plane to represent a vertex,
- or draw a curve between two vertex points that have already been laid down, making sure the new curve doesn't cross any of the previously drawn curves.

A new curve won't cross any other curves precisely when it stays within one of the continuous faces. Alternatively, a new curve won't have to cross any other curves if it can go between the outer faces of two different drawings. So to be sure it's ok to draw a new curve, we just need to check that its endpoints are on the boundary of the same face, or that its endpoints are on the outer faces of different drawings. Of course drawing the new curve changes the faces slightly, so the face boundaries will have to be updated once the new curve is drawn. This is the idea behind the following recursive definition.

Definition 12.2.2

A *planar embedding* of a *connected* graph consists of a nonempty set of closed walks of the graph called the *discrete faces* of the embedding. Planar embeddings are defined recursively as follows:

Base case: If G is a graph consisting of a single vertex, v, then a planar embedding of G has one discrete face, namely, the length zero closed walk, v.

Constructor case (split a face): Suppose G is a connected graph with a planar embedding, and suppose a and b are distinct, nonadjacent vertices of G that occur in some discrete face, γ, of the planar embedding. That is, γ is a closed walk of the form

$$\gamma = \alpha\hat{}\beta$$

where α is a walk from a to b and β is a walk from b to a. Then the graph obtained by adding the edge $\langle a - b\rangle$ to the edges of G has a planar embedding with the same discrete faces as G, except that face γ is replaced by the two discrete faces[2]

$$\alpha\hat{}\langle b - a\rangle \quad \text{and} \quad \langle a - b\rangle\hat{}\beta \tag{12.2.2}$$

as illustrated in Figure 12.9.[3]

Constructor case (add a bridge): Suppose G and H are connected graphs with planar embeddings and disjoint sets of vertices. Let γ be a discrete face of the embedding of G and suppose that γ begins and ends at vertex a.

Similarly, let δ be a discrete face of the embedding of H that begins and ends at vertex b.

Then the graph obtained by connecting G and H with a new edge, $\langle a - b\rangle$, has a planar embedding whose discrete faces are the union of the discrete faces of G and H, except that faces γ and δ are replaced by one new face

$$\gamma\hat{}\langle a - b\rangle\hat{}\delta\hat{}\langle b - a\rangle.$$

This is illustrated in Figure 12.10, where the vertex sequences of the faces of G and H are:

$$G : \{axyza, axya, ayza\} \quad H : \{btuvwb, btvwb, tuvt\},$$

and after adding the bridge $\langle a - b\rangle$, there is a single connected graph whose faces have the vertex sequences

$$axyzabtuvwba, axya, ayza, btvwb, tuvt.$$

A bridge is simply a cut edge, but in the context of planar embeddings, the bridges are precisely the edges that occur *twice on the same discrete face* —as opposed to once on each of two faces. Dongles are trees made of bridges; we only use dongles in illustrations, so there's no need to define them more precisely.

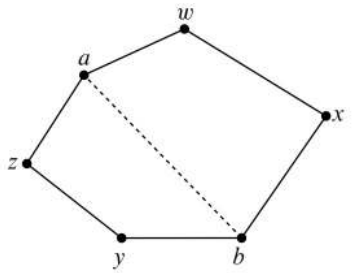


Figure 12.9 The “split a face” case: $awxbyza$ splits into $awxba$ and $abyza$.

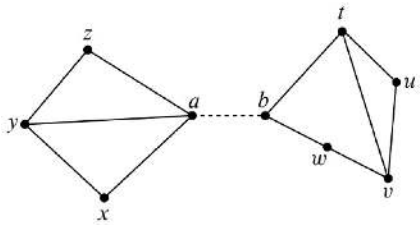


Figure 12.10 The “add a bridge” case.

Does It Work?

Yes! In general, a graph is planar because it has a planar drawing according to Definition 12.2.1 if and only if each of its connected components has a planar embedding as specified in Definition 12.2.2. Of course we can’t prove this without an excursion into exactly the kind of continuous math that we’re trying to avoid. But now that the recursive definition of planar graphs is in place, we won’t ever need to fall back on the continuous stuff. That’s the good news.

The bad news is that Definition 12.2.2 is a lot more technical than the intuitively simple notion of a drawing whose edges don’t cross. In many cases it’s easier to stick to the idea of planar drawings and give proofs in those terms. For example, erasing edges from a planar drawing will surely leave a planar drawing. On the other hand, it’s not so obvious, though of course it is true, that you can delete an edge from a planar embedding and still get a planar embedding (see Problem 12.9).

In the hands of experts, and perhaps in your hands too with a little more experience, proofs about planar graphs by appeal to drawings can be convincing and reliable. But given the long history of mistakes in such proofs, it’s safer to work from the precise definition of planar embedding. More generally, it’s also important to see how the abstract properties of curved drawings in the plane can be modelled successfully using a discrete data type.

Where Did the Outer Face Go?

Every planar drawing has an immediately-recognizable outer face —it’s the one that goes to infinity in all directions. But where is the outer face in a planar embedding?

There isn’t one! That’s because there really isn’t any need to distinguish one face from another. In fact, a planar embedding could be drawn with any given face on the outside. An intuitive explanation of this is to think of drawing the embedding on a *sphere* instead of the plane. Then any face can be made the outside face by “puncturing” that face of the sphere, stretching the puncture hole to a circle around the rest of the faces, and flattening the circular drawing onto the plane.

So pictures that show different “outside” boundaries may actually be illustrations of the same planar embedding. For example, the two embeddings shown in Figure 12.11 are really the same —check it: they have the same boundary cycles.

This is what justifies the “add bridge” case in Definition 12.2.2: whatever face is chosen in the embeddings of each of the disjoint planar graphs, we can draw a bridge between them without needing to cross any other edges in the drawing, because we can assume the bridge connects two “outer” faces.

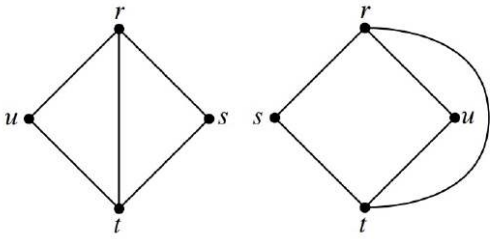


Figure 12.11 Two illustrations of the same embedding.

[1]Most texts drop the adjective *continuous* from the definition of a face as a connected region. We need the adjective to distinguish continuous faces from the *discrete* faces we're about to define.

[2]There is a minor exception to this definition of embedding in the special case when G is a line graph beginning with a and ending with b. In this case the cycles into which γ splits are actually the same. That's because adding edge $\langle a - b \rangle$ creates a cycle that divides the plane into "inner" and "outer" continuous faces that are both bordered by this cycle. In order to maintain the correspondence between continuous faces and discrete faces in this case, we define the two discrete faces of the embedding to be two "copies" of this same cycle.

[3]Formally, merge is an operation on walks, not a walk and an edge, so in (12.2.2), we should have used a walk $\langle a \langle a - b \rangle b \rangle$ instead of an edge $\langle a - b \rangle$ and written

$$\alpha\hat{\ }(b\langle b - a\rangle a) \text{ and } (a\langle a - b\rangle b)\hat{\ }\beta$$

This page titled 12.2: Definitions of Planar Graphs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

12.3: Euler's Formula

The value of the recursive definition is that it provides a powerful technique for proving properties of planar graphs, namely, structural induction. For example, we will now use Definition 12.2.2 and structural induction to establish one of the most basic properties of a connected planar graph, namely, that the number of vertices and edges completely determines the number of faces in every possible planar embedding of the graph.

Theorem 12.3.1

(Euler's Formula). *If a connected graph has a planar embedding, then*

$$v - e + f = 2$$

where v is the number of vertices, e is the number of edges, and f is the number of faces.

For example, in Figure 12.5, $v = 4, e = 6$, and $f = 4$. Sure enough, $4 - 6 + 4 = 2$, as Euler's Formula claims.

Proof

The proof is by structural induction on the definition of planar embeddings. Let $P(\epsilon)$ be the proposition that $v - e + f = 2$ for an embedding, ϵ.

Base case (ϵ is the one-vertex planar embedding): By definition, $v = 1, e = 0$, and $f = 1$, and $1 - 0 + 1 = 2$, so $P(\epsilon)$ indeed holds.

Constructor case (split a face): Suppose G is a connected graph with a planar embedding, and suppose a and b are distinct, nonadjacent vertices of G that appear on some discrete face, $\gamma = a \ldots b \cdots a$, of the planar embedding.

Then the graph obtained by adding the edge $\langle a - b \rangle$ to the edges of G has a planar embedding with one more face and one more edge than G. So the quantity $v - e + f$ will remain the same for both graphs, and since by structural induction this quantity is 2 for G's embedding, it's also 2 for the embedding of G with the added edge. So P holds for the constructed embedding.

Constructor case (add bridge): Suppose G and H are connected graphs with planar embeddings and disjoint sets of vertices. Then connecting these two graphs with a bridge merges the two bridged faces into a single face, and leaves all other faces unchanged. So the bridge operation yields a planar embedding of a connected graph with $v_G + v_H$ vertices, $e_G + e_H + 1$ edges, and $f_G + f_H - 1$ faces. Since

$$\begin{aligned}
&(v_G + v_H) - (e_G + e_H + 1) + (f_G + f_H - 1) \\
&\quad = (v_G - e_G + f_G) + (v_H - e_H + f_H) - 2 \\
&\quad = (2) + (2) - 2 \qquad \text{(by structural induction hypothesis)} \\
&\quad = 2,
\end{aligned}$$

$v - e + f$ remains equal to 2 for the constructed embedding. That is, $P(\epsilon)$ also holds in this case. This completes the proof of the constructor cases, and the theorem follows by structural induction. ■

This page titled 12.3: Euler's Formula is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

12.4: Bounding the Number of Edges in a Planar Graph

Like Euler's formula, the following lemmas follow by structural induction directly from Definition 12.2.2.

Lemma 12.4.1. *In a planar embedding of a connected graph, each edge occurs once in each of two different faces, or occurs exactly twice in one face.*

Lemma 12.4.2. *In a planar embedding of a connected graph with at least three vertices, each face is of length at least three.*

Combining Lemmas 12.4.1 and 12.4.2 with Euler's Formula, we can now prove that planar graphs have a limited number of edges:

Theorem 12.4.3

Suppose a connected planar graph has $v \geq 3$ vertices and e edges. Then

$$e \leq 3v - 6. \tag{12.4.1}$$

Proof

By definition, a connected graph is planar iff it has a planar embedding. So suppose a connected graph with v vertices and e edges has a planar embedding with f faces. By Lemma 12.4.1, every edge has exactly two occurrences in the face boundaries. So the sum of the lengths of the face boundaries is exactly $2e$. Also by Lemma 12.4.2, when $v \geq 3$, each face boundary is of length at least three, so this sum is at least $3f$. This implies that

$$3f \leq 2e. \tag{12.4.2}$$

But $f = e - v + 2$ by Euler's formula, and substituting into (12.4.2) gives

$$\begin{aligned} 3(e - v + 2) &\leq 2e \\ e - 3v + 6 &\leq 0 \\ e &\leq 3v - 6 \end{aligned} \tag{12.4.3}$$

■

This page titled 12.4: Bounding the Number of Edges in a Planar Graph is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

12.5: Returning to K5 and K3,3

Finally we have a simple way to answer the quadrapi question at the beginning of this chapter: the five quadrapi can't all shake hands without crossing. The reason is that we know the quadrupi question is the same as asking whether a complete graph K_5 is planar, and Theorem 12.4.3 has the immediate:

Corollary 12.5.1. *K_5 is not planar.*

Proof. K_5 is connected and has 5 vertices and 10 edges. But since $10 > 3 \cdot 5 - 6$, K_5 does not satisfy the inequality (12.4.1.) that holds in all planar graphs. ■

We can also use Euler's Formula to show that $K_{3,3}$ is not planar. The proof is similar to that of Theorem 12.4.1. except that we use the additional fact that $K_{3,3}$ is a bipartite graph.

Lemma 12.5.2. *In a planar embedding of a connected bipartite graph with at least 3 vertices, each face has length at least 4.*

Proof. By Lemma 12.4.2, every face of a planar embedding of the graph has length at least 3. But by Lemma 11.7.2 and Theorem 11.9.3.3, a bipartite graph can't have odd length closed walks. Since the faces of a planar embedding are closed walks, there can't be any faces of length 3 in a bipartite embedding. So every face must have length at least 4. ■

Theorem 12.5.3. *Suppose a connected bipartite graph with $v \geq 3$ vertices and e edges is planar. Then*

$$e \leq 2v - 4. \tag{12.5.1}$$

Proof. Lemma 12.5.2. implies that all the faces of an embedding of the graph have length at least 4. Now arguing as in the proof of Theorem 12.4.3, we find that the sum of the lengths of the face boundaries is exactly $2e$ and at least $4f$. Hence,

$$4f \leq 2e \tag{12.5.2}$$

for any embedding of a planar bipartite graph. By Euler's theorem, $f = 2 - v + e$. Substituting $2 - v + e$ for f in (12.5.2), we have

$$4(2 - v + e) \leq 2e,$$

which simplies to (12.5.1). ■

Corollary 12.5.4. *$K_{3,3}$ is not planar.*

Proof. $K_{3,3}$ is connected, bipartite and has 6 vertices and 9 edges. But since $9 > 2 \cdot 6 - 4$, $K_{3,3}$ does not satisfy the inequality (12.4.1.) that holds in all bipartite planar graphs. ■

This page titled 12.5: Returning to K5 and K3,3 is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

12.6: Coloring Planar Graphs

We've covered a lot of ground with planar graphs, but not nearly enough to prove the famous 4-color theorem. But we can get awfully close. Indeed, we have done almost enough work to prove that every planar graph can be colored using only 5 colors.

There are two familiar facts about planarity that we will need.

Lemma 12.6.1. *Any subgraph of a planar graph is planar.*

Lemma 12.6.2. *Merging two adjacent vertices of a planar graph leaves another planar graph.*

Merging two adjacent vertices, n_1 and n_2 of a graph means deleting the two vertices and then replacing them by a new "merged" vertex, m, adjacent to all the vertices that were adjacent to either of n_1 or n_2, as illustrated in Figure 12.12.

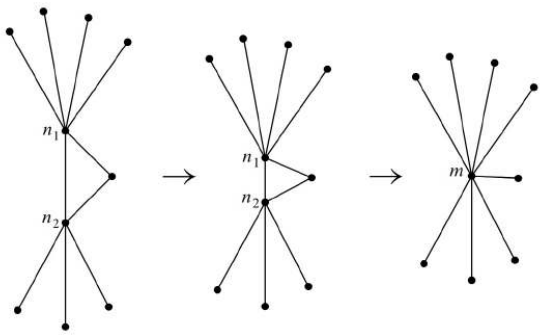


Figure 12.12 Merging adjacent vertices n_1 and n_2 into new vertex, m.

Many authors take Lemmas 12.6.1 and 12.6.2 for granted for continuous drawings of planar graphs described by Definition 12.2.1. With the recursive Definition 12.2.2 both Lemmas can actually be proved using structural induction (see Problem 12.9).

We need only one more lemma:

Lemma 12.6.3. *Every planar graph has a vertex of degree at most five.*

Proof. Assuming to the contrary that every vertex of some planar graph had degree at least 6, then the sum of the vertex degrees is at least $6v$. But the sum of the vertex degrees equals $2e$ by the Handshake Lemma 11.2.1, so we have $e \geq 3v$ contradicting the fact that $e \leq 3v - 6 < 3v$ by Theorem 12.4.3. ■

Theorem 12.6.4

Every planar graph is five-colorable.

Proof

The proof will be by strong induction on the number, v, of vertices, with induction hypothesis:

Every planar graph with v vertices is five-colorable.

Base cases ($v \leq 5$): immediate.

Inductive case: Suppose G is a planar graph with $v + 1$ vertices. We will describe a five-coloring of G.

First, choose a vertex, g, of G with degree at most 5; Lemma 12.6.3 guarantees there will be such a vertex.

Case 1: ($\deg(g) < 5$): Deleting g from G leaves a graph, H, that is planar by Lemma 12.6.1, and, since H has v vertices, it is five-colorable by induction hypothesis. Now define a five coloring of G as follows: use the five-coloring of H for all the vertices besides g, and assign one of the five colors to g that is not the same as the color assigned to any of its neighbors. Since there are fewer than 5 neighbors, there will always be such a color available for g.

Case 2: ($\deg(g) = 5$): If the five neighbors of g in G were all adjacent to each other, then these five vertices would form a nonplanar subgraph isomorphic to K_5, contradicting Lemma 12.6.1 (since K_5 is not planar). So there must be two neighbors, n_1 and n_2, of g that are not adjacent. Now merge n_1 and g into a new vertex, m. In this new graph, n_2 is adjacent to m, and the graph is planar by Lemma 12.6.2. So we can then merge m and n_2 into a another new vertex, m', resulting in a

new graph, G', which by Lemma 12.6.2 is also planar. Since G' has v 1 vertices, it is five-colorable by the induction hypothesis.

Now define a five coloring of G as follows: use the five-coloring of G' for all the vertices besides g, n_1 and n_2. Next assign the color of m' in G' to be the color of the neighbors n_1 and n_2. Since n_1 and n_2 are not adjacent in G, this defines a proper five-coloring of G except for vertex g. But since these two neighbors of g have the same color, the neighbors of g have been colored using fewer than five colors altogether. So complete the five-coloring of G by assigning one of the five colors to g that is not the same as any of the colors assigned to its neighbors. ■

This page titled 12.6: Coloring Planar Graphs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

12.7: Classifying Polyhedra

The Pythagoreans had two great mathematical secrets, the irrationality of $\sqrt{2}$ and a geometric construct that we're about to rediscover!

A *polyhedron* is a convex, three-dimensional region bounded by a finite number of polygonal faces. If the faces are identical regular polygons and an equal number of polygons meet at each corner, then the polyhedron is *regular*. Three examples of regular polyhedra are shown in Figure 12.13: the tetrahedron, the cube, and the octahedron.

We can determine how many more regular polyhedra there are by thinking about planarity. Suppose we took *any* polyhedron and placed a sphere inside it. Then we could project the polyhedron face boundaries onto the sphere, which would give an image that was a planar graph embedded on the sphere, with the images of the corners of the polyhedron corresponding to vertices of the graph. We've already observed that embeddings on a sphere are the same as embeddings on the plane, so Euler's formula for planar graphs can help guide our search for regular polyhedra.

For example, planar embeddings of the three polyhedra in Figure 12.1 are shown in Figure 12.14.

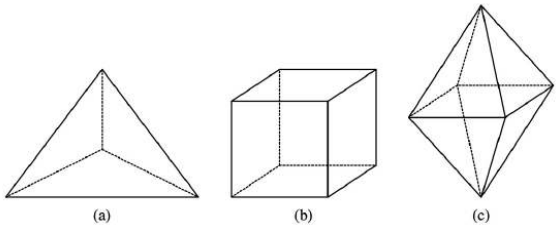


Figure 12.13 The tetrahedron (a), cube (b), and octahedron (c).

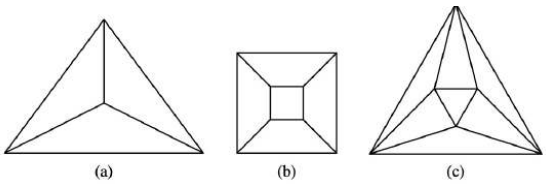


Figure 12.14 Planar embeddings of the tetrahedron (a), cube (b), and octahedron (c).

Let m be the number of faces that meet at each corner of a polyhedron, and let n be the number of edges on each face. In the corresponding planar graph, there are m edges incident to each of the v vertices. By the Handshake Lemma 11.2.1, we know:

$$mv = 2e.$$

Also, each face is bounded by n edges. Since each edge is on the boundary of two faces, we have:

$$nf = 2e$$

Solving for v and f in these equations and then substituting into Euler's formula gives:

$$\frac{2e}{m} - e + \frac{2e}{n} = 2$$

which simplifies to

$$\frac{1}{m} + \frac{1}{n} = \frac{1}{e} + \frac{1}{2} \tag{12.7.1}$$

Equation 12.7.1 places strong restrictions on the structure of a polyhedron. Every nondegenerate polygon has at least 3 sides, so $n \geq 3$. And at least 3 polygons must meet to form a corner, so $m \geq 3$. On the other hand, if either n or m were 6 or more, then the left side of the equation could be at most $\frac{1}{3} + \frac{1}{6} = \frac{1}{2}$, which is less than the right side. Checking the finitely-many cases that remain turns up only five solutions, as shown in Figure 12.15. For each valid combination of n and m, we can compute the associated number of vertices v, edges e, and faces f. And polyhedra with these properties do actually exist. The largest polyhedron, the dodecahedron, was the other great mathematical secret of the Pythagorean sect.

n	m	v	e	f	polyhedron
3	3	4	6	4	tetrahedron
4	3	8	12	6	cube
3	4	6	12	8	octahedron
3	5	12	30	20	icosahedron
5	3	20	30	12	dodecahedron

Figure 12.15 The only possible regular polyhedra.

The 5 polyhedra in Figure 12.15 are the only possible regular polyhedra. So if you want to put more than 20 geocentric satellites in orbit so that they *uniformly* blanket the globe—tough luck!

This page titled 12.7: Classifying Polyhedra is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

12.8: Another Characterization for Planar Graphs

We did not pick K_5 and $K_{3,3}$ as examples because of their application to dog houses or quadrapi shaking hands. We really picked them because they provide another, famous, discrete characterizarion of planar graphs:

Theorem 12.8.1

(Kuratowski). A graph is not planar if and only if it contains K_5 or $K_{3,3}$ as a minor.

Definition 12.8.2

A *minor* of a graph G is a graph that can be obtained by repeatedly[4] deleting vertices, deleting edges, and merging *adjacent* vertices of G.

For example, Figure 12.16 illustrates why C3 is a minor of the graph in Figure 12.16(a). In fact C3 is a minor of a connected graph G if and only if G is not a tree. The known proofs of Kuratowski's Theorem 12.8.1 are a little too long to include in an introductory text, so we won't give one.

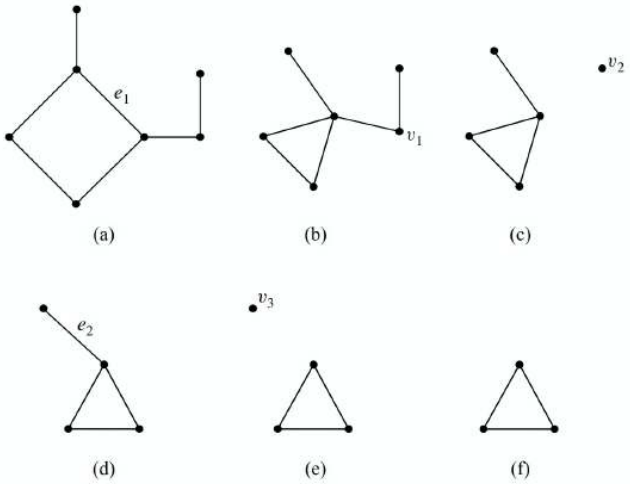


Figure 12.16 One method by which the graph in (a) can be reduced to C_3 (f), thereby showing that C_3 is a minor of the graph. The steps are: merging the nodes incident to e_1 (b), deleting v_1 and all edges incident to it (c), deleting v_2 (d), deleting e_2, and deleting v_3 (f).

[4]The three operations can each be performed any number of times in any order.

This page titled 12.8: Another Characterization for Planar Graphs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

SECTION OVERVIEW

3: Counting

13: Sums and Asymptotics

13.1: The Value of an Annuity
13.2: Sums of Powers
13.3: Approximating Sums
13.4: Hanging Out Over the Edge
13.5: Products
13.6: Double Trouble
13.7: Asymptotic Notation

14: Cardinality Rules

14.1: Counting One Thing by Counting Another
14.2: Counting Sequences
14.3: The Generalized Product Rule
14.4: The Division Rule
14.5: Counting Subsets
14.6: Sequences with Repetitions
14.7: Counting Practice - Poker Hands
14.8: The Pigeonhole Principle
14.9: Inclusion-Exclusion
14.10: Combinatorial Proofs

15: Generating Functions

15.1: Infinite Series
15.2: Counting with Generating Functions
15.3: Partial Fractions
15.4: Solving Linear Recurrences
15.5: Formal Power Series

This page titled 3: Counting is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

13: Sums and Asymptotics

Sums and products arise regularly in the analysis of algorithms, financial applications, physical problems, and probabilistic systems. For example, according to Theorem 2.2.1,

$$1+2+3+\cdots+n=\frac{n(n+1)}{2}. \tag{13.1}$$

Of course, the lefthand sum could be expressed concisely as a subscripted summation

$$\sum_{i=1}^{n} i$$

but the right hand expression $n(n+1)/2$ is not only concise but also easier to evaluate. Furthermore, it more clearly reveals properties such as the growth rate of the sum. Expressions like $n(n+1)/2$ that do not make use of subscripted summations or products—or those handy but sometimes troublesome sequences of three dots—are called *closed forms*.

Another example is the closed form for a *geometric sum*

$$1+x+x^2+x^3+\cdots+x^n=\frac{1-x^{n+1}}{1-x} \tag{13.2}$$

given in Problem 5.4. The sum as described on the left hand side of (13.2) involves n additions and $1+2+\cdots+(n-1)=(n-1)n/2$ multiplications, but its closed form on the right hand side can be evaluated using fast exponentiation with at most $2\log n$ multiplications, a division, and a couple of subtractions. Also, the closed form makes the growth and limiting behavior of the sum much more apparent.

Equations (13.1) and (13.2) were easy to verify by induction, but, as is often the case, the proofs by induction gave no hint about how these formulas were found in the first place. Finding them is part math and part art, which we'll start examining in this chapter.

Our first motivating example will be the value of a financial instrument known as an annuity. This value will be a large and nasty-looking sum. We will then describe several methods for finding closed forms for several sorts of sums, including those for annuities. In some cases, a closed form for a sum may not exist, and so we will provide a general method for finding closed forms for good upper and lower bounds on the sum.

The methods we develop for sums will also work for products, since any product can be converted into a sum by taking its logarithm. For instance, later in the chapter we will use this approach to find a good closed-form approximation to the *factorial function*

$$n! ::= 1\cdot 2\cdot 3\cdots n.$$

We conclude the chapter with a discussion of asymptotic notation, especially "Big Oh" notation. Asymptotic notation is often used to bound the error terms when there is no exact closed form expression for a sum or product. It also provides a convenient way to express the growth rate or order of magnitude of a sum or product.

13.1: The Value of an Annuity
13.2: Sums of Powers
13.3: Approximating Sums
13.4: Hanging Out Over the Edge
13.5: Products
13.6: Double Trouble
13.7: Asymptotic Notation

This page titled 13: Sums and Asymptotics is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

13.1: The Value of an Annuity

Would you prefer a million dollars today or $50,000 a year for the rest of your life? On the one hand, instant gratification is nice. On the other hand, the *total dollars* received at $50K per year is much larger if you live long enough.

Formally, this is a question about the value of an annuity. An *annuity* is a financial instrument that pays out a fixed amount of money at the beginning of every year for some specified number of years. In particular, an n-year, m-payment annuity pays m dollars at the start of each year for n years. In some cases, n is finite, but not always. Examples include lottery payouts, student loans, and home mortgages. There are even firms on Wall Street that specialize in trading annuities.[1]

A key question is, "What is an annuity worth?" For example, lotteries often pay out jackpots over many years. Intuitively, $50,000 a year for 20 years ought to be worth less than a million dollars right now. If you had all the cash right away, you could invest it and begin collecting interest. But what if the choice were between $50,000 a year for 20 years and a *half* million dollars today? Suddenly, it's not clear which option is better.

The Future Value of Money

In order to answer such questions, we need to know what a dollar paid out in the future is worth today. To model this, let's assume that money can be invested at a fixed annual interest rate p. We'll assume an 8% rate[2] for the rest of the discussion, so $p = 0.08$.

Here is why the interest rate p matters. Ten dollars invested today at interest rate p will become $(1+p)\cdot 10 = 10.80$ dollars in a year, $(1+p)^2 \cdot 10 \approx 11.66$ dollars in two years, and so forth. Looked at another way, ten dollars paid out a year from now is only really worth $1/(1+p)\cdot 10 \approx 9.26$ dollars today, because if we had the $9.26 today, we could invest it and would have $10.00 in a year anyway. Therefore, p determines the value of money paid out in the future.

So for an n-year, m-payment annuity, the first payment of m dollars is truly worth m dollars. But the second payment a year later is worth only $m/(1+p)$ dollars. Similarly, the third payment is worth $m/(1+p)^2$, and the n-th payment is worth only $m/(1+p)^{n-1}$. The total value, V, of the annuity is equal to the sum of the payment values. This gives:

$$\begin{aligned} V &= \sum_{i=1}^{n} \frac{m}{(1+p)^{i-1}} \\ &= m \cdot \sum_{j=0}^{n-1} \left(\frac{1}{1+p}\right)^j && \text{(substitute } j = i-1\text{)} \\ &= m \cdot \sum_{j=0}^{n-1} x^j && \text{(substitute } x = 1/(1+p)\text{).} \end{aligned} \tag{13.1.1}$$

The goal of the preceding substitutions was to get the summation into the form of a simple geometric sum. This leads us to an explanation of a way you could have discovered the closed form (13.2) in the first place using the *Perturbation Method*.

The Perturbation Method

Given a sum that has a nice structure, it is often useful to "perturb" the sum so that we can somehow combine the sum with the perturbation to get something much simpler. For example, suppose

$$S = 1 + x + x^2 + \cdots + x^n.$$

An example of a perturbation would be

$$xS = x + x^2 + \cdots + x^{n+1}.$$

The difference between S and xS is not so great, and so if we were to subtract xS from S, there would be massive cancellation:

$$\begin{aligned} S &= 1 + x + x^2 + x^3 + \cdots + x^n \\ -xS &= -x - x^2 - x^3 - \cdots - x^n - x^{n+1}. \end{aligned} \tag{13.1.2}$$

The result of the subtraction is

$$S - xS = 1 - x^{n+1}.$$

Solving for S gives the desired closed-form expression in equation 13.2, namely,

$$S = \frac{1 - x^{n+1}}{1 - x}.$$

We'll see more examples of this method when we introduce *generating functions* in Chapter 15.

13.1.3 A Closed Form for the Annuity Value

Using equation 13.2, we can derive a simple formula for V, the value of an annuity that pays m dollars at the start of each year for n years.

$$V = m\left(\frac{1 - x^n}{1 - x}\right) \qquad \text{(by equations 13.1.1 and 13.2)} \tag{13.1.3}$$

$$= m\left(\frac{1 + p - (1/(1+p))^{n-1}}{p}\right) \qquad \text{(substituting } x = 1/(1+p)\text{)}. \tag{13.1.4}$$

Equation 13.1.4 is much easier to use than a summation with dozens of terms. For example, what is the real value of a winning lottery ticket that pays \$50,000 per year for 20 years? Plugging in $m = \$50,000, n = 20$, and $p = 0.08$ gives $V \approx \$530,180$. So because payments are deferred, the million dollar lottery is really only worth about a half million dollars! This is a good trick for the lottery advertisers.

Infinite Geometric Series

We began this chapter by asking whether you would prefer a million dollars today or \$50,000 a year for the rest of your life. Of course, this depends on how long you live, so optimistically assume that the second option is to receive \$50,000 a year forever. This sounds like infinite money! But we can compute the value of an annuity with an infinite number of payments by taking the limit of our geometric sum in equation 13.2 as n tends to infinity

Theorem 13.1.1

If $|x| < 1$, *then*

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1 - x}.$$

Proof

$$\begin{aligned} \sum_{i=0}^{\infty} x^i &::= \lim_{n\to\infty} x^i \\ &= \lim_{n\to\infty} \frac{1 - x^{n+1}}{1 - x} \qquad \text{(by equation 13.2)} \\ &= \frac{1}{1 - x}. \end{aligned} \tag{13.1.5}$$

The final line follows from the fact that $\lim_{n\to\infty} x^{n+1} = 0$ when $|x| < 1$. ■

In our annuity problem, $x = 1/(1+p) < 1$, so Theorem 13.1.1 applies, and we get

$$\begin{aligned} V &= m \cdot \sum_{j=0}^{\infty} x^j \qquad \text{(by equation 13.1.1.)} \\ &= m \cdot \frac{1}{1 - x} \qquad \text{(by Theorem 13.1.1.)} \\ &= m \cdot \frac{1 + p}{p} \qquad (x = 1/(1+p)). \end{aligned} \tag{13.1.6}$$

Plugging in $m = \$50,000$ and $p = 0:08$, we see that the value V is only $\$675,000$. It seems amazing that a million dollars today is worth much more than \$50,000 paid every year for eternity! But on closer inspection, if we had a million dollars today in the bank earning 8% interest, we could take out and spend \$80,000 a year, forever. So as it turns out, this answer really isn't so amazing after all.

Examples

Equation 13.2 and Theorem 13.1.1 are incredibly useful in computer science.

Here are some other common sums that can be put into closed form using equation 13.2 and Theorem 13.1.1:

$$1+1/2+1/4+\cdots=\sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i=\frac{1}{1-(1/2)}=2 \tag{13.1.7}$$

$$0.99999\cdots=0.9\sum_{i=0}^{\infty}\left(\frac{1}{10}\right)^i=0.9\left(\frac{1}{1-1/10}\right)=0.9\left(\frac{10}{9}\right)=1 \tag{13.1.8}$$

$$1-1/2+1/4-\cdots=\sum_{i=0}^{\infty}\left(\frac{-1}{2}\right)^i=\frac{1}{1-(-1/2)}=\frac{2}{3} \tag{13.1.9}$$

$$1+2+4+\cdots+2^{n-1}=\sum_{i=0}^{n-1}2^i=\frac{1-2^n}{1-2}=2^n-1 \tag{13.1.10}$$

$$1+3+9+\cdots+3^{n-1}=\sum_{i=0}^{n-1}3^i=\frac{1-3^n}{1-3}=\frac{3^n-1}{2} \tag{13.1.11}$$

If the terms in a geometric sum grow smaller, as in equation 13.1.7, then the sum is said to be *geometrically decreasing*. If the terms in a geometric sum grow progressively larger, as in equations 13.1.10 and 13.1.11, then the sum is said to be *geometrically increasing*. In either case, the sum is usually approximately equal to the term in the sum with the greatest absolute value. For example, in equations 13.1.7 and 13.1.9, the largest term is equal to 1 and the sums are 2 and 2/3, both relatively close to 1. In equation 13.1.10, the sum is about twice the largest term. In equation 13.1.11, the largest term is 3^{n-1} and the sum is $(3^n-1)/2$, which is only about a factor of 1:5 greater. You can see why this rule of thumb works by looking carefully at equation 13.2 and Theorem 13.1.1.

Variations of Geometric Sums

We now know all about geometric sums—if you have one, life is easy. But in practice one often encounters sums that cannot be transformed by simple variable substitutions to the form $\sum x^i$.

A non-obvious but useful P way to obtain new summation formulas from old ones is by differentiating or integrating with respect to x. As an example, consider the following sum:

$$\sum_{i=1}^{n-1}ix^i=x+2x^2+3x^3+\cdots+(n-1)x^{n-1}$$

This is not a geometric sum. The ratio between successive terms is not fixed, and so our formula for the sum of a geometric sum cannot be directly applied. But differentiating equation 13.2 leads to:

$$\frac{d}{dx}\left(\sum_{i=0}^{n-1}x^i\right)=\frac{d}{dx}\left(\frac{1-x^n}{1-x}\right). \tag{13.1.12}$$

The left-hand side of equation 13.1.12 is simply

$$\sum_{i=0}^{n-1}\frac{d}{dx}(x^i)=\sum_{i=0}^{n-1}i(x^{i-1})$$

The right-hand side of equation 13.1.12 is

$$\begin{aligned}\frac{-nx^{n-1}(1-x)-(-1)(1-x^n)}{(1-x)^2}&=\frac{-nx^{n-1}+nx^n+1-x^n)}{(1-x)^2}\\&=\frac{1-nx^{n-1}+(n-1)x^n}{(1-x)^2}.\end{aligned} \tag{13.1.13}$$

Hence, equation 13.1.12 means that

$$\sum_{i=0}^{n-1}ix^{i-1}=\frac{1-nx^{n-1}+(n-1)x^n}{(1-x)^2}.$$

Incidentally, Problem 13.2 shows how the perturbation method could also be applied to derive this formula.

Often, differentiating or integrating messes up the exponent of x in every term. In this case, we now have a formula for a sum of the form $\sum ix^{i-1}$, but we want a formula for the series$\sum ix^{i-1}$. The solution is simple: multiply by x. This gives:

$$\sum_{i=0}^{n-1} ix^i = \frac{x - nx^n + (n-1)x^{n+1}}{(1-x)^2} \tag{13.1.14}$$

and we have the desired closed-form expression for our sum. It seems a little complicated, but it's easier to work with than the sum.

Notice that if $|x| < 1$, then this series converges to a finite value even if there are infinitely many terms. Taking the limit of equation 13.1.14 as n tends to infinity gives the following theorem:

Theorem 13.1.2

If $|x| < 1$, *then*

$$\sum_{i=0}^{\infty} ix^i = \frac{x}{(1-x)^2}. \tag{13.1.15}$$

As a consequence, suppose that there is an annuity that pays im dollars at the end of each year i, forever. For example, if $m = \$50,000$, then the payouts are \$50,000 and then \$100,000 and then \$150,000 and so on. It is hard to believe that the value of this annuity is finite! But we can use Theorem 13.1.2 to compute the value:

$$\begin{aligned} V &= \sum_{i=1}^{\infty} \frac{im}{(1+p)^i} \\ &= m \cdot \frac{1/(1+p)}{(1-\frac{1}{1+p})^2} \\ &= m \cdot \frac{1+p}{p^2}. \end{aligned} \tag{13.1.16}$$

The second line follows by an application of Theorem 13.1.2. The third line is obtained by multiplying the numerator and denominator by $(1+p)^2$.

For example, if $m = \$50,000$, and $p = 0.08$ as usual, then the value of the annuity is $V = \$8,437,500$. Even though the payments increase every year, the increase is only additive with time; by contrast, dollars paid out in the future decrease in value exponentially with time. The geometric decrease swamps out the additive increase. Payments in the distant future are almost worthless, so the value of the annuity is finite.

The important thing to remember is the trick of taking the derivative (or integral) of a summation formula. Of course, this technique requires one to compute nasty derivatives correctly, but this is at least theoretically possible!

[1]Such trading ultimately led to the subprime mortgage disaster in 2008–2009. We'll talk more about that in a later chapter.

[2]U.S. interest rates have dropped steadily for several years, and ordinary bank deposits now earn around 1.0%. But just a few years ago the rate was 8%; this rate makes some of our examples a little more dramatic. The rate has been as high as 17% in the past thirty years.

This page titled 13.1: The Value of an Annuity is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

13.2: Sums of Powers

In Chapter 5, we verified the formula (13.1), but the source of this formula is still a mystery. Sure, we can prove that it's true by using well ordering or induction, but where did the expression on the right come from in the first place? Even more inexplicable is the closed form expression for the sum of consecutive squares:

$$\sum_{i=1}^{n} i^2 = \frac{(2n+1)(n+1)n}{6}. \tag{13.2.1}$$

It turns out that there is a way to derive these expressions, but before we explain it, we thought it would be fun—OK, our definition of "fun" may be different than yours—to show you how Gauss is supposed to have proved equation 13.1 when he was a young boy

Gauss's idea is related to the perturbation method we used in Section 13.1.2. Let

$$S = \sum_{i=1}^{n} i.$$

Then we can write the sum in two orders:

$$S = 1 + 2 + \ldots + (n-1) + n,$$

$$S = n + (n-1) + \ldots + 2 + 1.$$

Adding these two equations gives

$$\begin{aligned} 2S &= (n+1) + (n+1) + \cdots + (n+1) + (n+1) \\ &= n(n+1). \end{aligned} \tag{13.2.2}$$

Hence,

$$S = \frac{n(n+1)}{2}.$$

Not bad for a young child—Gauss showed some potential....

Unfortunately, the same trick does not work for summing consecutive squares. However, we can observe that the result might be a third-degree polynomial in n, since the sum contains n terms that average out to a value that grows quadratically in n. So we might guess that

$$\sum_{i=1}^{n} i^2 = an^3 + bn^2 + cn + d.$$

If our guess is correct, then we can determine the parameters a, b, c, and d by plugging in a few values for n. Each such value gives a linear equation in a, b, c, and d. If we plug in enough values, we may get a linear system with a unique solution. Applying this method to our example gives:

$$\begin{aligned} &n = 0 \text{ implies } 0 = d \\ &n = 1 \text{ implies } 1 = a + b + c + d \\ &n = 2 \text{ implies } 5 = 8a + 4b + 2c + d \\ &n = 3 \text{ implies } 14 = 27a + 9b + 3c + d. \end{aligned} \tag{13.2.3}$$

Solving this system gives the solution $a = 1/3, b = 1/2, c = 1/6, d = 0$. Therefore, *if* our initial guess at the form of the solution was correct, then the summation is equal to $n^3/3 + n^2/2 + n/6$, which matches equation 13.2.1.

The point is that if the desired formula turns out to be a polynomial, then once you get an estimate of the *degree* of the polynomial, all the coefficients of the polynomial can be found automatically.

Be careful! This method lets you discover formulas, but it doesn't guarantee they are right! After obtaining a formula by this method, it's important to go back and *prove* it by induction or some other method. If the initial guess at the solution was not of the right form, then the resulting formula will be completely wrong! A later chapter will describe a method based on generating functions that does not require any guessing at all.

This page titled 13.2: Sums of Powers is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

13.3: Approximating Sums

Unfortunately, it is not always possible to find a closed-form expression for a sum. For example, no closed form is known for

$$S = \sum_{i=1}^{n} \sqrt{i}.$$

In such cases, we need to resort to approximations for S if we want to have a closed form. The good news is that there is a general method to find closed-form upper and lower bounds that works well for many sums. Even better, the method is simple and easy to remember. It works by replacing the sum by an integral and then adding either the first or last term in the sum.

Definition 13.3.1

A function $f : \mathbb{R}^+ \to \mathbb{R}^+$ is *strictly increasing* when

$$x < y \text{ IMPLIES } f(x) < f(y),$$

and it is *weakly increasing* [3] when

$$x < y \text{ IMPLIES } f(x) \leq f(y).$$

Similarly, f is *strictly increasing* when

$$x < y \text{ IMPLIES } f(x) > f(y),$$

and it is *weakly decreasing* [4] when

$$x < y \text{ IMPLIES } f(x) \geq f(y).$$

For example, 2^x and $\sqrt{x}$ are strictly increasing functions, while $\max\{x, 2\}$ and $\lceil x \rceil$ are weakly increasing functions. The functions $1/x$ and 2^{-x} are strictly decreasing, while $\min\{1/x, 1/2\}$ and $\lfloor 1/x \rfloor$ are weakly decreasing.

Theorem 13.3.1

Let $f : \mathbb{R}^+ \to \mathbb{R}^+$ be a weakly increasing function. Define

$$S ::= \sum_{i=1}^{n} f(i) \tag{13.3.1}$$

and

$$I ::= \int_{1}^{n} f(x)dx.$$

Then

$$I + f(1) \leq S \leq I + f(n). \tag{13.3.2}$$

Similarly, if f is a weakly decreasing, then

$$I + f(n) \leq S \leq I + f(1).$$

Proof

Suppose $f : \mathbb{R}^+ \to \mathbb{R}^+$ is weakly increasing. The value of the sum S in 13.3.1 is the sum of the areas of n unit-width rectangles of heights $f(1), f(2), \ldots, f(n)$. This area of these rectangles is shown shaded in Figure 13.1.

The value of

$$I = \int_1^n f(x)dx$$

is the shaded area under the curve of $f(x)$ from 1 to n shown in Figure 13.2.

Comparing the shaded regions in Figures 13.1 and 13.2 shows that S is at least I plus the area of the leftmost rectangle. Hence,

$$S \geq I + f(1) \tag{13.3.3}$$

This is the lower bound for S given in 13.3.2.

To derive the upper bound for S given in 13.3.2, we shift the curve of $f(x)$ from 1 to n one unit to the left as shown in Figure 13.3.

Comparing the shaded regions in Figures 13.1 and 13.3 shows that S is at most I plus the area of the rightmost rectangle. That is,

$$S \leq I + f(n)$$

which is the upper bound for S given in 13.3.2.

The very similar argument for the weakly decreasing case is left to Problem 13.10. ■

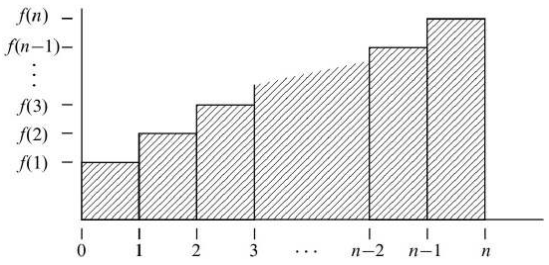


Figure 13.1 The area of the ith rectangle is $f(i)$. The shaded region has area $\sum_{i=1}^{n} f(i)$.

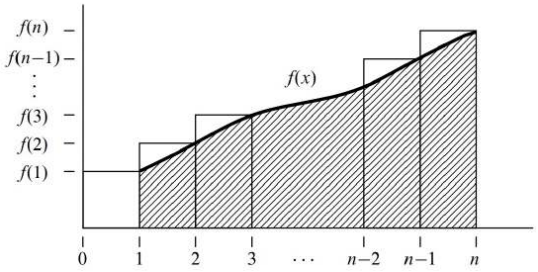


Figure 13.2 The shaded area under the curve of $f(x)$ from 1 to n (shown in bold) is $I = \int_1^n f(x)dx$.

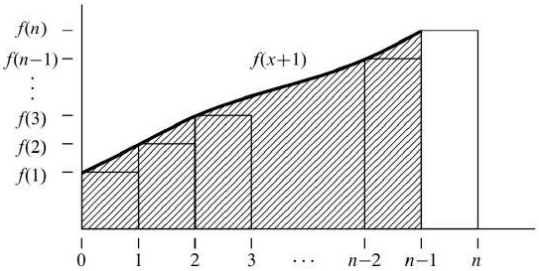


Figure 13.3 This curve is the same as the curve in Figure 13.2 shifted left by 1.

Theorem 13.3.2 provides good bounds for most sums. At worst, the bounds will be off by the largest term in the sum. For example, we can use Theorem 13.3.2 to bound the sum

$$S = \sum_{i=1}^{n} \sqrt{i}$$

as follows.

We begin by computing

$$\begin{aligned} I &= \int_1^n \sqrt{x}dx \\ &= \frac{x^{3/2}}{3/2}\Big|_1^n \\ &= \frac{2}{3}(n^{3/2} - 1). \end{aligned} \tag{13.3.4}$$

We then apply Theorem 13.3.2 to conclude that

$$\frac{2}{3}(n^{3/2} - 1) + 1 \leq S \leq \frac{2}{3}(n^{3/2} - 1) + \sqrt{n}$$

and thus that

$$\frac{2}{3}n^{3/2} + \frac{1}{3} \leq S \leq \frac{2}{3}n^{3/2} + \sqrt{n} - \frac{2}{3}.$$

In other words, the sum is very close to $\frac{2}{3}n^{3/2}$. We'll define several ways that one thing can be "very close to" something else at the end of this chapter.

As a first application of Theorem 13.3.2, we explain in the next section how it helps in resolving a classic paradox in structural engineering.

[3]Weakly increasing functions are usually called *nondecreasing* functions. We will avoid this terminology to prevent confusion between being a nondecreasing function and the much weaker property of *not* being a decreasing function.

[4]Weakly decreasing functions are usually called *nonincreasing*.

This page titled 13.3: Approximating Sums is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

13.4: Hanging Out Over the Edge

Suppose you have a bunch of books and you want to stack them up, one on top of another in some off-center way, so the top book sticks out past books below it without falling over. If you moved the stack to the edge of a table, how far past the edge of the table do you think you could get the top book to go? Could the top book stick out completely beyond the edge of table? You're not supposed to use glue or any other support to hold the stack in place.

Most people's first response to the Book Stacking Problem—sometimes also their second and third responses—is "No, the top book will never get completely past the edge of the table." But in fact, you can get the top book to stick out as far as you want: one booklength, two booklengths, any number of booklengths!

Formalizing the Problem

We'll approach this problem recursively. How far past the end of the table can we get one book to stick out? It won't tip as long as its center of mass is over the table, so we can get it to stick out half its length, as shown in Figure 13.4.

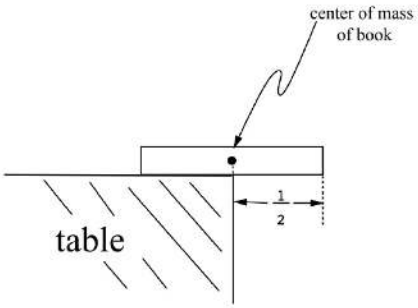


Figure 13.4 One book can overhang half a book length.

Now suppose we have a stack of books that will not tip over if the bottom book rests on the table—call that a stable stack. Let's define the overhang of a stable stack to be the horizontal distance from the center of mass of the stack to the furthest edge of the top book. So the overhang is purely a property of the stack, regardless of its placement on the table. If we place the center of mass of the stable stack at the edge of the table as in Figure 13.5, the overhang is how far we can get the top book in the stack to stick out past the edge.

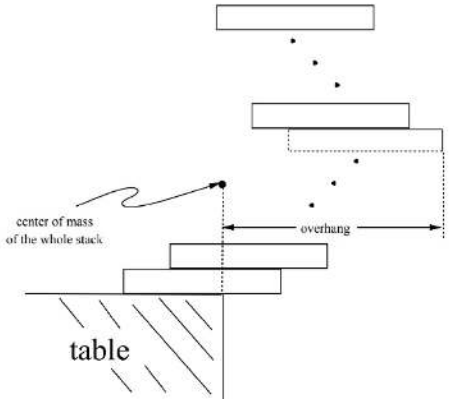


Figure 13.5 Overhanging the edge of the table.

In general, a stack of n books will be stable if and only if the center of mass of the top i books sits over the $(i+1)$st book for $i = 1, 2, \ldots, n-1$.

So we want a formula for the maximum possible overhang, B_n, achievable with a stable stack of n books.

We've already observed that the overhang of one book is 1/2 a book length. That is,

$$B_1 = \frac{1}{2}.$$

Now suppose we have a stable stack of $n+1$ books with maximum overhang. If the overhang of the n books on top of the bottom book was not maximum, we could get a book to stick out further by replacing the top stack with a stack of n books with larger overhang. So the maximum overhang, B_{n+1}, of a stack of $n+1$ books is obtained by placing a maximum overhang stable stack of n books on top of the bottom book. And we get the biggest overhang for the stack of $n+1$ books by placing the center of mass of the n books right over the edge of the bottom book as in Figure 13.6.

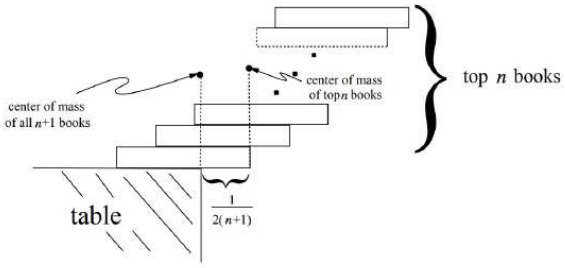


Figure 13.6 Additional overhang with $n+1$ books.

So we know where to place the $n+1$st book to get maximum overhang. In fact, the reasoning above actually shows that this way of stacking $n+1$ books is the unique way to build a stable stack where the top book extends as far as possible. All we have to do is calculate what this extension is.

The simplest way to do that is to let the center of mass of the top n books be the origin. That way the horizontal coordinate of the center of mass of the whole stack of $n+1$ books will equal the increase in the overhang. But now the center of mass of the bottom book has horizontal coordinate $1/2$, so the horizontal coordinate of center of mass of the whole stack of $n+1$ books is

$$\frac{0\cdot n+(1/2)\cdot 1}{n+1}=\frac{1}{2(n+1)}.$$

In other words,

$$B_{n+1}=B_n+\frac{1}{2(n+1)}, \tag{13.4.1}$$

as shown in Figure 13.6.

Expanding equation(13.4.1), we have

$$\begin{aligned} B_{n+1} &= B_{n-1}+\frac{1}{2n}+\frac{1}{2(n+1)} \\ &= B_1+\frac{1}{2\cdot 2}+\cdots+\frac{1}{2n}+\frac{1}{2(n+1)} \\ &= \frac{1}{2}\sum_{i=1}^{n+1}\frac{1}{i}. \end{aligned} \tag{13.4.2}$$

So our next task is to examine the behavior of B_n as n grows.

Harmonic Numbers

Definition 13.4.1

The nth harmonic number, H_n, is

$$H_n ::= \sum_{i=1}^{n}\frac{1}{i}.$$

So (13.4.2) means that

$$B_n=\frac{H_n}{2}.$$

The first few harmonic numbers are easy to compute. For example, $H_4 = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = \frac{25}{12} > 2$. The fact that H_4 is greater than 2 has special significance: it implies that the total extension of a 4-book stack is greater than one full book! This is the situation shown in Figure 13.7.

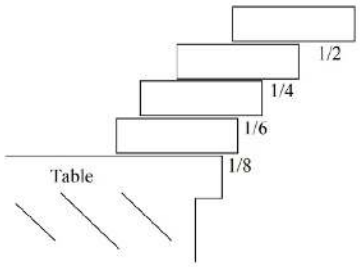


Figure 13.7 Stack of four books with maximum overhang.

There is good news and bad news about harmonic numbers. The bad news is that there is no known closed-form expression for the harmonic numbers. The good news is that we can use Theorem 13.3.2 to get close upper and lower bounds on H_n. In particular, since

$$\int_1^n \frac{1}{x}dx = \ln(x)\Big|_1^n = \ln(n),$$

Theorem 13.3.2 means that

$$\ln(n) + \frac{1}{n} \leq H_n \leq \ln(n) + 1. \tag{13.4.3}$$

In other words, the nth harmonic number is very close to $\ln(n)$.

Because the harmonic numbers frequently arise in practice, mathematicians have worked hard to get even better approximations for them. In fact, it is now known that

$$H_n = \ln(n) + \gamma + \frac{1}{2n} + \frac{1}{12n^2} + \frac{\epsilon(n)}{120n^4} \tag{13.4.4}$$

Here γ is a value 0.577215664... called *Euler's constant*, and $\epsilon(n)$ is between 0 and 1 for all n. We will not prove this formula.

We are now finally done with our analysis of the book stacking problem. Plugging the value of H_n into (13.4.2), we find that the maximum overhang for n books is very close to $1/2\ln(n)$. Since $\ln(n)$ grows to infinity as n increases, this means that if we are given enough books we can get a book to hang out arbitrarily far over the edge of the table. Of course, the number of books we need will grow as an exponential function of the overhang; it will take 227 books just to achieve an overhang of 3, never mind an overhang of 100.

Extending Further Past the End of the Table

The overhang we analyzed above was the furthest out the *top* book could extend past the table. This leaves open the question of if there is some better way to build a stable stack where some book other than the top stuck out furthest. For example, Figure 13.8 shows a stable stack of two books where the bottom book extends further out than the top book. Moreover, the bottom book extends 3/4 of a book length past the end of the table, which is the same as the maximum overhang for the top book in a two book stack.

Since the two book arrangement in Figure 13.8(a) ties the maximum overhang stack in Figure 13.8(b), we could take the unique stable stack of n books where the top book extends furthest, and switch the top two books to look like Figure 13.8(a). This would give a stable stack of n books where the second from the top book extends the same maximum overhang distance. So for $n > 1$, there are at least two ways of building a stable stack of n books which both extend the maximum overhang distance—one way where the top book is furthest out, and another way where the second from the top book is furthest out.

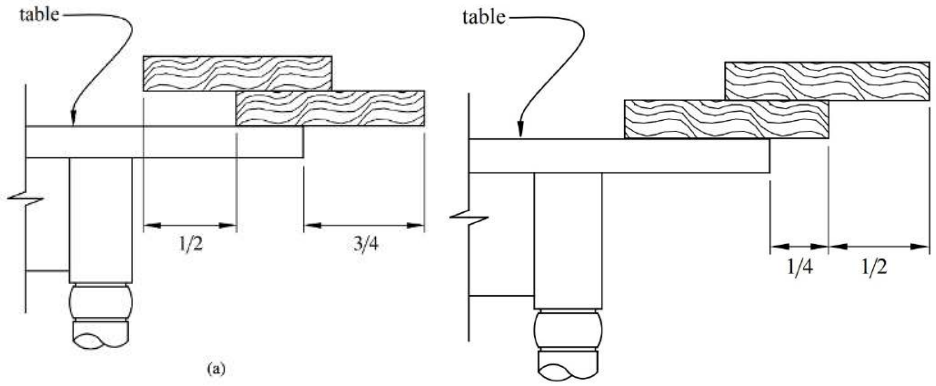


Figure 13.8 Figure (a) shows a stable stack of two books where the bottom book extends the same amount past the end of the table as the maximum overhang twobook stack shown in Figure (b).

It turns out that there is no way to beat these two ways of making stable stacks. In fact, it's not too hard to show that these are the *only* two ways to get a stable stack of books that achieves maximum overhang.

But there is more to the story. All our reasoning above was about stacks in which *one* book rests on another. It turns out that by building structures in which more than one book rests on top of another book—think of an inverted pyramid—it is possible to get a stack of n books to extend proportional to $\sqrt[3]{n}$—much more than $\ln n$—book lengths without falling over. See [13], *Maximum Overhang*.

Asymptotic Equality

For cases like Equation **13.4.4** where we understand the growth of a function like H_n up to some (unimportant) error terms, we use a special notation, ~, to denote the leading term of the function. For example, we say that $H_n \sim \ln(n)$ to indicate that the leading term of H_n is $\ln(n)$. More precisely:

Definition 13.4.2

For functions $f, g : \mathbb{R} \to \mathbb{R}$, we say f is *asymptotically equal* to g, in symbols,

$$f(x) \sim g(x)$$

\iff

$$\lim_{x\to\infty} f(x)/g(x) = 1.$$

Although it is tempting to write $H_n \sim \ln(n) + \gamma$ to indicate the two leading terms, this is not really right. According to Definition 13.4.2, $H_n \sim \ln(n) + c$ where c is *any constant*. The correct way to indicate that γ is the second-largest term is $H_n - \ln(n) \sim \gamma$.

The reason that the ~ notation is useful is that often we do not care about lower order terms. For example, if $n = 100$, then we can compute $H(n)$ to great precision using only the two leading terms:

$$|H_n - \ln(n) - \gamma| \le |\frac{1}{200} - \frac{1}{120000} + \frac{1}{120 \cdot 100^4}| < \frac{1}{200}.$$

We will spend a lot more time talking about asymptotic notation at the end of the chapter. But for now, let's get back to using sums.

This page titled 13.4: Hanging Out Over the Edge is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

13.5: Products

We've covered several techniques for finding closed forms for sums but no methods for dealing with products. Fortunately, we do not need to develop an entirely new set of tools when we encounter a product such as

$$n! ::= \prod_{i=1}^{n} i. \tag{13.5.1}$$

That's because we can convert any product into a sum by taking a logarithm. For example, if

$$P = \prod_{i=1}^{n} f(i),$$

then

$$\ln(P) = \sum_{i=1}^{n} \ln(f(i)).$$

We can then apply our summing tools to find a closed form (or approximate closed form) for $\ln(P)$ and then exponentiate at the end to undo the logarithm.

For example, let's see how this works for the factorial function $n!$. We start by taking the logarithm:

$$\begin{aligned} \ln(n!) &= \ln(1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n) \\ &= \ln(1) + \ln(2) + \ln(3) + \cdots + \ln(n-1) + \ln(n) \\ &= \sum_{i=1}^{n} \ln(i). \end{aligned} \tag{13.5.2}$$

Unfortunately, no closed form for this sum is known. However, we can apply Theorem 13.3.2 to find good closed-form bounds on the sum. To do this, we first compute

$$\begin{aligned} \int_1^n \ln(x)dx &= x\ln(x) - x\Big|_1^n \\ &= n\ln(n) - n + 1. \end{aligned} \tag{13.5.3}$$

Plugging into Theorem 13.3.2, this means that

$$n\ln(n) - n + 1 \leq \sum_{i=1}^{n} \ln(i) \leq n\ln(n) - n + 1 + \ln(n).$$

Exponentiating then gives

$$\frac{n^n}{e^{n-1}} \leq n! \leq \frac{n^{n+1}}{e^{n-1}} \tag{13.5.4}$$

This means that $n!$ is within a factor of n of n^n/e^{n-1}.

Stirling's Formula

The most commonly used product in discrete mathematics is probably $n!$, and mathematicians have workedto find tight closed-form bounds on its value. The most useful bounds are given in Theorem 13.5.1.

Theorem 13.5.1

(Stirling's Formula). For all $n \geq 1$,

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\epsilon(n)}$$

where

$$\frac{1}{12n+1} \leq \epsilon(n) \leq \frac{1}{12n}.$$

Theorem 13.5.1 can be proved by induction (with some pain), and there are lots of proofs using elementary calculus, but we won't go into them.

There are several important things to notice about Stirling's Formula. First, $\epsilon(n)$ is always positive. This means that

$$n! > \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{13.5.5}$$

for all $n \in \mathbb{N}^+$.

Second, $\epsilon(n)$ tends to zero as n gets large. This means that

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{13.5.6}$$

which is impressive. After all, who would expect both π and e to show up in a closed-form expression that is asymptotically equal to $n!$?

Third, $\epsilon(n)$is small even for small values of n. This means that Stirling's Formula provides good approximations for $n!$ for most all values of n. For example, if we use

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

as the approximation for $n!$, as many people do, we are guaranteed to be within a factor of

$$e^{\epsilon(n)} \leq e^{\frac{1}{12n}}$$

of the correct value. For $n \geq 10$, this means we will be within 1% of the correct value. For $n \geq 100$, the error will be less than 0.1%.

If we need an even closer approximation for $n!$, then we could use either

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/12n}$$

or

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/(12n+1)}$$

depending on whether we want an upper, or a lower, bound. By Theorem 13.5.1, we know that both bounds will be within a factor of

$$e^{\frac{1}{12n}-\frac{1}{12n+1}} = e^{\frac{1}{144n^2+12n}}$$

of the correct value. For $n \geq 10$, this means that either bound will be within 0.01% of the correct value. For $n \geq 100$, the error will be less than 0.0001%.

For quick future reference, these facts are summarized in Corollary 13.5.2 and Table 13.1.

Approximation	$n \geq 1$	$n \geq 10$	$n \geq 100$	$n \geq 1000$
$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$	$< 10\%$	$< 1\%$	$< 0.1\%$	$< 0.01\%$
$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{1/12n}$	$< 1\%$	$< 0.01\%$	$< 0.0001\%$	$< 0.000001\%$

Table 13.1 Error bounds on common approximations for $n!$ from Theorem 13.5.1. For example, if $n > 100$, then $\sqrt{2\pi n}\frac{n}{e}^n$ approximates $n!$ to within 0.1%.

Corollary 13.5.2.

$$n! < \sqrt{2\pi n}\frac{n}{e}^n \cdot \begin{cases} 1.09 & for\, n \geq 1, \\ 1.009 & for\, n \geq 10, \\ 1.0009 & for\, n \geq 100. \end{cases}$$

This page titled 13.5: Products is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

13.6: Double Trouble

Sometimes we have to evaluate sums of sums, otherwise known as *double summations*. This sounds hairy, and sometimes it is. But usually, it is straightforward— you just evaluate the inner sum, replace it with a closed form, and then evaluate the outer sum (which no longer has a summation inside it). For example,[5]

$$\begin{aligned}
\sum_{n=0}^{\infty}\left(y^n\sum_{i=0}^{n}x^i\right) &= \sum_{n=0}^{\infty}\left(y^n\frac{1-x^{n+1}}{1-x}\right) && \text{equation 13.2}\\
&= \frac{1}{1-x}\sum_{n=0}^{\infty}y^n-\left(\frac{1}{1-x}\right)\sum_{n=0}^{\infty}y^nx^{n+1}\\
&= \frac{1}{(1-x)(1-y)}-\left(\frac{x}{1-x}\right)\sum_{n=0}^{\infty}(xy)^n && \text{Theorem 13.1.1}\\
&= \frac{1}{(1-x)(1-y)}-\frac{x}{(1-x)(1-xy)} && \text{Theorem 13.1.1}\\
&= \frac{(1-xy)-x(1-y)}{(1-x)(1-y)(1-xy)}\\
&= \frac{1-x}{(1-x)(1-y)(1-xy)}\\
&= \frac{1}{(1-y)(1-xy)}.
\end{aligned} \tag{13.6.1}$$

When there's no obvious closed form for the inner sum, a special trick that is often useful is to try *exchanging the order of summation*. For example, suppose we want to compute the sum of the first n harmonic numbers

$$\sum_{k=1}^{n}H_k=\sum_{k=1}^{n}\sum_{j=1}^{k}\frac{1}{j} \tag{13.6.2}$$

For intuition about this sum, we can apply Theorem 13.3.2 to equation 13.4.3 to conclude that the sum is close to

$$\int_1^n \ln(x)dx = x\ln(x)-x\Big|_1^n = n\ln(n)-n+1.$$

Now let's look for an exact answer. If we think about the pairs (k,j) over which we are summing, they form a triangle:

		j						
		1	2	3	4	5	...	n
k	1	1						
	2	1	1/2					
	3	1	1/2	1/3				
	4	1	1/2	1/3	1/4			
		...						
	n	1	1/2		...			$1/n$

The summation in Equation 13.6.2 is summing each row and then adding the row sums. Instead, we can sum the columns and then add the column sums. Inspecting the table we see that this double sum can be written as

$$\begin{aligned}\sum_{k=1}^{n} H_k &= \sum_{k=1}^{n}\sum_{j=1}^{n}\frac{1}{j}\\ &= \sum_{j=1}^{n}\sum_{k=j}^{n}\frac{1}{j}\\ &= \sum_{j=1}^{n}\frac{1}{j}\sum_{k=j}^{n}1\\ &= \sum_{j=1}^{n}\frac{1}{j}(n-j+1)\\ &= \sum_{j=1}^{n}\frac{n+1}{j}-\sum_{j=1}^{n}\frac{j}{j}\\ &= (n+1)\sum_{j=1}^{n}\frac{1}{j}-\sum_{j=1}^{n}1\\ &= (n+1)H_n - n. \end{aligned} \tag{13.6.3}$$

[5]OK, so maybe this one is a little hairy, but it is also fairly straightforward. Wait till you see the next one!

This page titled 13.6: Double Trouble is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

13.7: Asymptotic Notation

Asymptotic notation is a shorthand used to give a quick measure of the behavior of a function $f(n)$ as n grows large. For example, the asymptotic notation ~ of Definition 13.4.2 is a binary relation indicating that two functions grow at the *same* rate. There is also a binary relation "little oh" indicating that one function grows at a significantly *slower* rate than another and "Big Oh" indicating that one function grows not much more rapidly than another.

Little O

Definition 13.7.1

For functions $f, g : \mathbb{R} \to \mathbb{R}$, with g nonnegative, we say f is *asymptotically smaller* than g, in symbols,

$$f(x) = o(g(x)),$$

iff

$$\lim_{x\to\infty} f(x)/g(x) = 0.$$

For example, $1000x^{1.9} = o(x^2)$, because $1000x^{1.9}/(x^2) = 1000/x^{0.1}$ and since $x^{0.1}$ goes to infinity with x and 1000 is constant, we have $\lim_{x\to\infty} 1000x^{1.9}/x^2 = 0$. This argument generalizes directly to yield

Lemma 13.7.2. $x^a = o(x^b)$ *for all nonnegative constants* $a < b$.

Using the familiar fact that $\log x < x$ for all $x > 1$, we can prove

Lemma 13.7.3. $\log x = o(x^\epsilon)$ *for all* $\epsilon > 0$.

Proof. Choose $\epsilon > \delta > 0$ and let $x = z^\delta$ in the equality $\log x < x$. This implies

$$\log z < z^\delta/\delta = o(z^\epsilon) \qquad \text{by Lemma 13.7.2.} \tag{13.7.1}$$

Corollary 13.7.4. $x^b = o(a^x)$ *for any* $a, b \in \mathbb{R}$ *with* $a > 1$.

Lemma 13.7.3 and Corollary 13.7.4 can also be proved using l'Hôpital s Rule or the Maclaurin Series for $\log x$ and e^x. Proofs can be found in most calculus texts.

Big O

Big O is the most frequently used asymptotic notation. It is used to give an upper bound on the growth of a function, such as the running time of an algorithm. There is a standard definition of Big Oh given below in 13.7.9, but we'll begin with an alternative definition that makes apparent several basic properties of Big Oh.

Definition 13.7.5

Given functions $f, g : \mathbb{R} \to \mathbb{R}$ with g nonnegative, we say that

$$f = O(g)$$

iff

$$\limsup_{x\to\infty} |f(x)|/g(x) < \infty.$$

Here we're using the technical notion of *limit superior*[6] instead of just limit. But because limits and lim sup's are the same when limits exist, this formulation makes it easy to check basic properties of Big Oh. We'll take the following Lemma for granted.

Lemma 13.7.6. *If a function* $f : \mathbb{R} \to \mathbb{R}$ *has a finite or infinite limit as its argument approaches infinity, then its limit and limit superior are the same.*

Now Definition 13.7.5 immediately implies:

Lemma 13.7.7. *If f = o(g) or* $f \sim g$, *then* $f = O(g)$.

Proof. $\lim f/g = 0$ or $\lim f/g = 1$ implies $\lim f/g < \infty$, so by Lemma 13.7.6, $\limsup f/g < \infty$.

Note that the converse of Lemma 13.7.7 is not true. For example, $2x = O(x)$, but $2x \not\sim x$ and $2x \neq o(x)$.

We also have:

Lemma 13.7.8. *If $f = o(g)$, then it is* not *true that $g = O(f)$.*

Proof.

$$\lim_{x\to\infty} \frac{g(x)}{f(x)} = \frac{1}{\lim_{x\to\infty} f(x)/g(x)} = \frac{1}{0} = \infty.$$

so by lemma 13.7.6, $g \neq O(f)$. ■

We need lim sup's in Definition 13.7.5 to cover cases when limits don't exist. For example, if $f(x)/g(x)$ oscillates between 3 and 5 as x grows, then $\lim_{x\to\infty} f(x)/g(x)$ does not exist, but $f = O(g)$ because $\limsup_{x\to\infty} f(x)/g(x) = 5$.

An equivalent, more usual formulation of big O does not mention $\limsup's$.

Definition 13.7.9

Given functions $f, g : \mathbb{R} \to \mathbb{R}$ with g nonnegative, we say

$$f = O(g)$$

iff there exists a constant $c \geq 0$ and an x_0 such that for all $x \geq x_0$, $|f(x) \leq cg(x)$.

This definition is rather complicated, but the idea is simple: $f(x) = O(g(x))$ means $f(x)$ is less than or equal to $g(x)$, except that we're willing to ignore a constant factor, namely, c, and to allow exceptions for small x, namely, $x < x_0$. So in the case that $f(x)/g(x)$ oscillates between 3 and 5, $f = O(g)$ according to Definition 13.7.9 because $f \leq 5g$.

Proposition 13.7.10. $100x^2 = O(x^2)$.

Proof. Choose $c = 100$ and $x_0 = 1$. Then the proposition holds, since for all $x \geq 1$, $100x^2 \leq 100x^2$. ■

Proposition 13.7.11. $x^2 + 100x + 10 = O(x^2)$.

Proof. $x^2 + 100x + 10)/x^2 = 1 + 100/x + 10/x^2$ and so its limit as x approaches infinity is $1 + 0 + 0 = 1$. So in fact, $x^2 + 100x + 10 \sim x^2$, and therefore $x^2 + 100x + 10 = O(x^2)$. Indeed, it's conversely true that $x^2 = O(x^2 + 100x + 10)$. ■

Proposition 13.7.11 generalizes to an arbitrary polynomial:

Proposition 13.7.12. $a_k x^k + a_{k-1}x^{k-1} + \cdots + a_1 x + a_0 = O(x^k)$.

We'll omit the routine proof.

Big O notation is especially useful when describing the running time of an algorithm. For example, the usual algorithm for multiplying $n \times n$ matrices uses a number of operations proportional to n^3 in the worst case. This fact can be expressed concisely by saying that the running time is $O(n^3)$. So this asymptotic notation allows the speed of the algorithm to be discussed without reference to constant factors or lower-order terms that might be machine specific. It turns out that there is another matrix multiplication procedure that uses $O(n^{2.55})$ operations. The fact that this procedure is asymptotically faster indicates that it involves new ideas that go beyond a simply more efficient implementation of the $O(n^3)$ method.

Of course the asymptotically faster procedure will also definitely be much more efficient on large enough matrices, but being asymptotically faster does not mean that it is a better choice. The $O(n^{2.55})$-operation multiplication procedure is almost never used in practice because it only becomes more efficient than the usual $O(n^3)$ procedure on matrices of impractical size.[7]

Theta

Sometimes we want to specify that a running time $T(n)$ is precisely quadratic up to constant factors (both upper bound *and* lower bound). We could do this by saying that $T(n) = O(n^2)$ and $n^2 = O(T(n))$, but rather than say both, mathematicians have devised yet another symbol, Θ, to do the job.

Definition 13.7.13

$$f = \Theta(g) \text{ iff } f = O(g) \text{ and } g = O(f).$$

The statement $f = \Theta(g)$ can be paraphrased intuitively as "f and g are equal to within a constant factor."

The Theta notation allows us to highlight growth rates and suppress distracting factors and low-order terms. For example, if the running time of an algorithm is

$$T(n) = 10n^3 - 20n^2 + 1,$$

then we can more simply write

$$T(n) = \Theta(n^3).$$

In this case, we would say that T *is of order* n^3 or that $T(n)$ *grows cubically*, which is often the main thing we really want to know. Another such example is

$$\pi^2 3^{x-7} + \frac{(2.7x^{113} + x^9 - 86)^4}{\sqrt{x}} - 1.08^{3x} = \Theta(3^x).$$

Just knowing that the running time of an algorithm is $\Theta(3^x)$, for example, is useful, because if n doubles we can predict that the running time will *by and large*[8] increase by a factor of at most 8 for large n. In this way, Theta notation preserves information about the scalability of an algorithm or system. Scalability is, of course, a big issue in the design of algorithms and systems.

Pitfalls with Asymptotic Notation

There is a long list of ways to make mistakes with asymptotic notation. This section presents some of the ways that big O notation can lead to trouble. With minimal effort, you can cause just as much chaos with the other symbols.

The Exponential Fiasco

Sometimes relationships involving big O are not so obvious. For example, one might guess that $4^x = O(2^x)$ since 4 is only a constant factor larger than 2. This reasoning is incorrect, however; 4^x actually grows as the square of 2^x.

Constant Confusion

Every constant is $O(1)$. For example, $17 = O(1)$. This is true because if we let $f(x) = 17$ and $g(x) = 1$, then there exists a $c > 0$ and an x_0 such that $|f(x)| \leq cg(x)$. In particular, we could choose $c = 17$ and $x_0 = 1$, since $|17| \leq 17 \cdot 1$ for all $x \geq 1$. We can construct a false theorem that exploits this fact.

False Theorem 13.7.14.

$$\sum_{i=1}^{n} i = O(n)$$

Bogus proof. Define $f(n) = \sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n$. Since we have shown that every constant i is $O(1)$, $f(n) = O(1) + O(1) + \cdots + O(1) = O(n)$. ■

Of course in reality $\sum_{i=1}^{n} i = n(n+1)/2 \neq O(n)$.

The error stems from confusion over what is meant in the statement $i = O(1)$. For any *constant* $i \in \mathbb{N}$ it is true that $i = O(1)$. More precisely, if f is any constant function, then $f = O(1)$. But in this False Theorem, i is not constant—it ranges over a set of values $0, 1, \ldots, n$ that depends on n.

And anyway, we should not be adding $O(1)$'s as though they were numbers. We never even defined what $O(g)$ means by itself; it should only be used in the context "$f = O(g)$" to describe a relation between functions f and g.

Equality Blunder

The notation $f = O(g)$ is too firmly entrenched to avoid, but the use of "=" is regrettable. For example, if $f = O(g)$, it seems quite reasonable to write $O(g) = f$. But doing so might tempt us to the following blunder: because $2n = O(n)$, we can say $O(n) = 2n$. But

$n = O(n)$, so we conclude that $n = O(n) = 2n$, and therefore $n = 2n$. To avoid such nonsense, we will never write "$O(f) = g$."

Similarly, you will often see statements like

$$H_n = \ln(n) + \gamma + O\left(\frac{1}{n}\right)$$

or

$$n! = (1 + o(1))\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

In such cases, the true meaning is

$$H_n = \ln(n) + \gamma + f(n)$$

for some $f(n)$ where $f(n) = O(1/n)$, and

$$n! = (1 + g(n))\sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

where $g(n) = o(1)$. These last transgressions are OK as long as you (and your reader) know what you mean.

Operator Application Blunder

It's tempting to assume that familiar operations preserve asymptotic relations, but it ain't necessarily so. For example, $f \sim g$ in general does not even imply that $3^f = \Theta(3^g)$. On the other hand, some operations preserve and even strengthen asymptotic relations, for example,

$$f = \Theta(g) \text{ IMPLIES } \ln f \sim \ln g.$$

See Problem 13.24.

Omega (Optional)

Sometimes people incorrectly use Big Oh in the context of a lower bound. For example, they might say, "The running time, $T(n)$, is at least $O(n^2)$." This is another blunder! Big Oh can only be used for *upper* bounds. The proper way to express the lower bound would be

$$n^2 = O(T(n)).$$

The lower bound can also be described with another special notation "big Omega."

Definition 13.7.15

Given functions $f, g : \mathbb{R} \to \mathbb{R}$ with f nonnegative, define

$$f = \Omega(g)$$

to mean

$$g = O(f).$$

For example, $x^2 =$ DEL $\Omega(x)$, $2^x =$ DEL $\Omega(x^2)$, and $x/100 = \Omega(100x + \sqrt{x}$.

So if the running time of your algorithm on inputs of size n is $T(n)$, and you want to say it is at least quadratic, say

$$T(n) = \Omega(n^2).$$

There is a similar "little omega" notation for lower bounds corresponding to little o:

Definition 13.7.16

Given functions $f, g : \mathbb{R} \to \mathbb{R}$ with f nonnegative, define

$$f = \omega(g)$$

to mean

$$g = o(f).$$

For example, $x^{1.5} = \omega(x)$ and $\sqrt{x} = \omega(\ln^2(x))$.

The little omega symbol is not as widely used as the other asymptotic symbols we defined.

[6]The precise definition of lim sup is

$$\limsup_{x\to\infty} h(x) ::= \lim_{x\to\infty} \text{lub}_{y\geq x} h(y),$$

where "lub" abbreviates "least upper bound."

[7]It is even conceivable that there is an $O(n^2)$ matrix multiplication procedure, but none is known.

[8]Since $\Theta(n^3)$ only implies that the running time, $T(n)$, is between cn^3 and dn^3 for constants $0 < c < d$, the time $T(2n)$ could regularly exceed $T(n)$ by a factor as large as $8d/c$. The factor is sure to be close to 8 for all large n only if $T(n) \sim n^3$.

13.7: Asymptotic Notation is shared under a not declared license and was authored, remixed, and/or curated by LibreTexts.

CHAPTER OVERVIEW

14: Cardinality Rules

14.1: Counting One Thing by Counting Another
14.2: Counting Sequences
14.3: The Generalized Product Rule
14.4: The Division Rule
14.5: Counting Subsets
14.6: Sequences with Repetitions
14.7: Counting Practice - Poker Hands
14.8: The Pigeonhole Principle
14.9: Inclusion-Exclusion
14.10: Combinatorial Proofs

This page titled 14: Cardinality Rules is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.1: Counting One Thing by Counting Another

How do you count the number of people in a crowded room? You could count heads, since for each person there is exactly one head. Alternatively, you could count ears and divide by two. Of course, you might have to adjust the calculation if someone lost an ear in a pirate raid or someone was born with three ears. The point here is that you can often *count one thing by counting another*, though some fudging may be required. This is a central theme of counting, from the easiest problems to the hardest. In fact, we've already seen this technique used in Theorem 4.5.5, where the number of subsets of an n-element set was proved to be the same as the number of length-n bit-strings, by describing a bijection between the subsets and the bit-strings.

The most direct way to count one thing by counting another is to find a bijection between them, since if there is a bijection between two sets, then the sets have the same size. This important fact is commonly known as the *Bijection Rule*. We've already seen it as the Mapping Rules bijective case (4.5.3).

The Bijection Rule

The Bijection Rule acts as a magnifier of counting ability; if you figure out the size of one set, then you can immediately determine the sizes of many other sets via bijections. For example, let's look at the two sets mentioned at the beginning of Part III:

$$\begin{aligned} A &= \text{all ways to select a dozen donuts when five varieties are available} \\ B &= \text{all 16-bit sequences with exactly 4 ones} \end{aligned} \tag{14.1.1}$$

An example of an element of set A is:

$$\underbrace{00}_{\text{chocolate}}\ \underbrace{\ }_{\text{lemon-filled}}\ \underbrace{000000}_{\text{sugar}}\ \underbrace{00}_{\text{glazed}}\ \underbrace{00}_{\text{plain}}$$

Here, we've depicted each donut with a 0 and left a gap between the different varieties. Thus, the selection above contains two chocolate donuts, no lemon-filled, six sugar, two glazed, and two plain. Now let's put a 1 into each of the four gaps:

$$\underbrace{00}_{\text{chocolate}}\ 1\ \underbrace{\ }_{\text{lemon-filled}}\ \underbrace{10000001}_{\text{sugar}}\ \underbrace{00}_{\text{glazed}}\ 1\ \underbrace{00}_{\text{plain}}$$

and close up the gaps:

$$0011000000100100.$$

We've just formed a 16-bit number with exactly 4 ones—an element of B!

This example suggests a bijection from set A to set B: map a dozen donuts consisting of:

$$c \text{ chocolate, } l \text{ lemon-filled, } s \text{ sugar, } g \text{ glazed, and } p \text{ plain}$$

to the sequence:

$$\underbrace{0\cdots0}_{c}1\underbrace{0\cdots0}_{l}1\underbrace{0\cdots0}_{s}1\underbrace{0\cdots0}_{g}1\underbrace{0\cdots0}_{p}$$

The resulting sequence always has 16 bits and exactly 4 ones, and thus is an element of B. Moreover, the mapping is a bijection: every such bit sequence comes from exactly one order of a dozen donuts. Therefore, $|A| = |B|$ by the Bijection Rule. More generally,

Lemma 14.1.1. *The number of ways to select n donuts when k flavors are available is the same as the number of binary sequences with exactly n zeroes and $k-1$ ones.*

This example demonstrates the power of the bijection rule. We managed to prove that two very different sets are actually the same size—even though we don't know exactly how big either one is. But as soon as we figure out the size of one set, we'll immediately know the size of the other.

This particular bijection might seem frighteningly ingenious if you've not seen it before. But you'll use essentially this same argument over and over, and soon you'll consider it routine.

This page titled 14.1: Counting One Thing by Counting Another is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.2: Counting Sequences

The Bijection Rule lets us count one thing by counting another. This suggests a general strategy: get really good at counting just a few things, then use bijections to count everything else! This is the strategy we'll follow. In particular, we'll get really good at counting *sequences*. When we want to determine the size of some other set T, we'll find a bijection from T to a set of sequences S. Then we'll use our super-ninja sequence-counting skills to determine $|S|$, which immediately gives us $|T|$. We'll need to hone this idea somewhat as we go along, but that's pretty much it!

The Product Rule

The *Product Rule* gives the size of a product of sets. Recall that if $P_1, P_2, \ldots P_n$ are sets, then

$$P_1 \times P_2 \times \cdots \times P_n$$

is the set of all sequences whose first term is drawn from P_1, second term is drawn from P_2 and so forth.

Rule 14.2.1 (Product Rule). *If $P_1, P_2, \ldots P_n$ are finite sets, then:*

$$|P_1 \times P_2 \times \cdots \times P_n| = |P_1| \cdot |P_2| \cdots |P_n|$$

For example, suppose a *daily diet* consists of a breakfast selected from set B, a lunch from set L, and a dinner from set D where:

$$\begin{aligned} B &= \{\text{pancakes, bacon and eggs, bagel, Doritos}\} \\ L &= \{\text{burger and fries, garden salad, Doritos}\} \\ D &= \{\text{macaroni, pizza, frozen burrito, pasta, Doritos}\} \end{aligned} \tag{14.2.1}$$

Then $B \times L \times D$ is the set of all possible daily diets. Here are some sample elements:

$$(\text{pancakes, burger and fries, pizza})$$

$$(\text{bacon and eggs, garden salad, pasta})$$

$$(\text{Doritos, Doritos, frozen burrito})$$

The Product Rule tells us how many different daily diets are possible:

$$\begin{aligned} |B \times L \times D| &= |B| \cdot |L| \cdot |D| \\ &= 4 \cdot 3 \cdot 5 \\ &= 60. \end{aligned} \tag{14.2.2}$$

Subsets of an n-element Set

The fact that there are 2^n subsets of an n-element set was proved in Theorem 4.5.5 by setting up a bijection between the subsets and the length-n bit-strings. So the original problem about subsets was tranformed into a question about sequences— *exactly according to plan*! Now we can fill in the missing explanation of why there are 2^n length-n bit-strings: we can write the set of all n-bit sequences as a product of sets:

$$\{0,1\}^n ::= \underbrace{\{0,1\} \times \{0,1\} \times \cdots \times \{0,1\}}_{n \text{ terms}}.$$

Then Product Rule gives the answer:

$$|\{0,1\}^n| = |\{0,1\}|^n = 2^n.$$

The Sum Rule

Bart allocates his little sister Lisa a quota of 20 crabby days, 40 irritable days, and 60 generally surly days. On how many days can Lisa be out-of-sorts one way or another? Let set C be her crabby days, I be her irritable days, and S be the generally surly. In these terms, the answer to the question is $|C \cup I \cup S|$. Now assuming that she is permitted at most one bad quality each day, the size of this union of sets is given by the *Sum Rule*:

Rule 14.2.2 (Sum Rule). *If $A_1, A_2, \ldots, A_n$ are* disjoint *sets, then:*

$$|A_1 \cup A_2 \cup \ldots A_n| = |A_1| + |A_2| + \cdots + |A_n|$$

$$\begin{aligned} |C \cup I \cup S| &= |C| + |I| + |S| \\ &= 20 + 40 + 60 \\ &= 120 \text{ days.} \end{aligned} \tag{14.2.3}$$

Notice that the Sum Rule holds only for a union of *disjoint* sets. Finding the size of a union of overlapping sets is a more complicated problem that we'll take up in Section 14.9.

Counting Passwords

Few counting problems can be solved with a single rule. More often, a solution is a flurry of sums, products, bijections, and other methods.

For solving problems involving passwords, telephone numbers, and license plates, the sum and product rules are useful together. For example, on a certain computer system, a valid password is a sequence of between six and eight symbols. The first symbol must be a letter (which can be lowercase or uppercase), and the remaining symbols must be either letters or digits. How many different passwords are possible?

Let's define two sets, corresponding to valid symbols in the first and subsequent positions in the password.

$$\begin{aligned} F &= \{a, b, \ldots, z, A, B, \ldots, Z\} \\ S &= \{a, b, \ldots, z, A, B, \ldots, Z, 0, 1, \ldots, 9\} \end{aligned} \tag{14.2.4}$$

In these terms, the set of all possible passwords is:[1]

$$(F \times S^5) \cup (F \times S^6) \cup (F \times S^7)$$

Thus, the length-six passwords are in the set $F \times S^5$, the length-seven passwords are in $F \times S^6$, and the length-eight passwords are in $F \times S^7$. Since these sets are disjoint, we can apply the Sum Rule and count the total number of possible passwords as follows:

$$\begin{aligned} &|(F \times S^5) \cup (F \times S^6) \cup (F \times S^7)| \\ &\quad = |F \times S^5| + |F \times S^6| + |F \times S^7| && \text{Sum Rule} \\ &\quad = |F| \times |S|^5 + |F| \times |S|^6 + |F| \times |S|^7 && \text{Product Rule} \\ &\quad = 52 \cdot 62^5 + 52 \cdot 62^6 + 52 \cdot 62^7 \\ &\quad \approx 1.8 \cdot 10^{14} \text{ different passwords.} \end{aligned} \tag{14.2.5}$$

[1]The notation S^5 means $S \times S \times S \times S \times S$.

This page titled 14.2: Counting Sequences is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.3: The Generalized Product Rule

In how many ways can, say, a Nobel prize, a Japan prize, and a Pulitzer prize be awarded to n people? This is easy to answer using our strategy of translating the problem about awards into a problem about sequences. Let P be the set of n people taking the course. Then there is a bijection from ways of awarding the three prizes to the set $P^3 ::= P \times P \times P$. In particular, the assignment:

$$\text{“Barak wins a Nobel, George wins a Japan, and Bill wins a Pulitzer prize”}$$

maps to the sequence (Barak, George, Bill). By the Product Rule, we have $|P^3| = |P|^3 = n^3$, so there are n^3 ways to award the prizes to a class of n people. Notice that P^3 includes triples like (Barak, Bill, George) where one person wins more than one prize.

But what if the three prizes must be awarded to *different* students? As before, we could map the assignment to the triple (Bill, George, Barak) $\in P^3$. But this function is *no longer a bijection*. For example, no valid assignment maps to the triple (Barak, Bill, Barak) because now we're not allowing Barak to receive two prizes. However, there *is* a bijection from prize assignments to the set:

$$S = \{(x, y, z) \in P^3 \mid x, y, \text{ and } z \text{ are different people}\}$$

This reduces the original problem to a problem of counting sequences. Unfortunately, the Product Rule does not apply directly to counting sequences of this type because the entries depend on one another; in particular, they must all be different. However, a slightly sharper tool does the trick.

Prizes for *truly exceptional* Coursework

Given everyone's hard work on this material, the instructors considered awarding some prizes for truly exceptional coursework. Here are three possible prize categories:

Best Administrative Critique We asserted that the quiz was closed-book. On the cover page, one strong candidate for this award wrote, “There is no book.”

Awkward Question Award “Okay, the left sock, right sock, and pants are in an antichain, but how—even with assistance—could I put on all three at once?”

Best Collaboration Statement Inspired by a student who wrote “I worked alone” on Quiz 1.

Rule 14.3.1 (Generalized Product Rule). *Let S be a set of length-k sequences. If there are:*

- n_1 *possible first entries,*
- n_2 *possible second entries for each first entry,*

$\vdots$

- n_k *possible kth entries for each sequence of first $k-1$ entries,*

then

$$|S| = n_1 \cdot n_2 \cdot n_3 \cdots n_k$$

In the awards example, S consists of sequences (x, y, z). There are n ways to choose x, the recipient of prize #1. For each of these, there are $n-1$ ways to choose y, the recipient of prize #2, since everyone except for person x is eligible. For each combination of x and y, there are $n-2$ ways to choose z, the recipient of prize #3, because everyone except for x and y is eligible. Thus, according to the Generalized Product Rule, there are

$$|S| = n \cdot (n-1) \cdot (n-2)$$

ways to award the 3 prizes to different people.

Defective Dollar Bills

A dollar bill is *defective* if some digit appears more than once in the 8-digit serial number. If you check your wallet, you'll be sad to discover that defective bills are all-too-common. In fact, how common are *nondefective* bills? Assuming that the digit portions of

serial numbers all occur equally often, we could answer this question by computing

Misplaced '#'

Let's first consider the denominator. Here there are no restrictions; there are 10 possible first digits, 10 possible second digits, 10 third digits, and so on. Thus, the total number of 8-digit serial numbers is 10^8 by the Product Rule.

Next, let's turn to the numerator. Now we're not permitted to use any digit twice. So there are still 10 possible first digits, but only 9 possible second digits, 8 possible third digits, and so forth. Thus, by the Generalized Product Rule, there are

$$10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 = \frac{10!}{2} = 1{,}814{,}400$$

serial numbers with all digits different. Plugging these results into Equation ???, we find:

$$\text{fraction of nondefective bills} = \frac{1{,}814{,}400}{100{,}000{,}000} = 1.8144$$

Chess Problem

In how many different ways can we place a pawn (P), a knight (N), and a bishop (B) on a chessboard so that no two pieces share a row or a column? A valid configuration is shown in Figure 14.1(a), and an invalid configuration is shown in Figure 14.1(b).

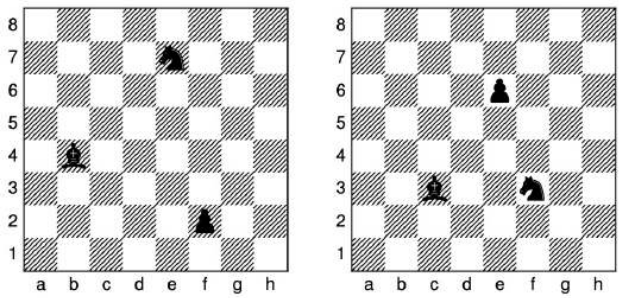


Figure 14.1 Two ways of placing a pawn, a knight, and a bishop on a chessboard. The configuration shown in (b) is invalid because the bishop and the knight are in the same row.

First, we map this problem about chess pieces to a question about sequences. There is a bijection from configurations to sequences

$$(r_P, c_P, r_N, c_N, r_B, c_B)$$

where r_P, r_N, and r_B are distinct rows and c_P, c_N, and c_B are distinct columns. In particular, r_P is the pawn's row, c_P is the pawn's column, r_N is the knight's row, etc. Now we can count the number of such sequences using the Generalized Product Rule:

- r_P is one of 8 rows
- c_P is one of 8 columns
- r_N is one of 7 rows (any one but r_P)
- c_N is one of 7 columns (any one but c_P)
- r_B is one of 6 rows (any one but r_P or r_N)
- c_B is one of 6 columns (any one but c_P or c_N)

Thus, the total number of configurations is $(8 \cdot 7 \cdot 6)^2$.

Permutations

A *permutation* of a set S is a sequence that contains every element of S exactly once. For example, here are all the permutations of the set $\{a, b, c\}$:

$$\begin{array}{lll} (a,b,c) & (a,c,b) & (b,a,c) \\ (b,c,a) & (c,a,b) & (c,b,a) \end{array} \tag{14.3.1}$$

How many permutations of an n-element set are there? Well, there are n choices for the first element. For each of these, there are $n-1$ remaining choices for the second element. For every combination of the first two elements, there are $n-2$ ways to choose the third element, and so forth. Thus, there are a total of

$$n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 = n!$$

permutations of an n-element set. In particular, this formula says that there are $3! = 6$ permutations of the 3-element set $\{a, b, c\}$, which is the number we found above.

Permutations will come up again in this course approximately 1.6 bazillion times. In fact, permutations are the reason why factorial comes up so often and why we taught you Stirling's approximation:

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

This page titled 14.3: The Generalized Product Rule is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.4: The Division Rule

Counting ears and dividing by two is a silly way to count the number of people in a room, but this approach is representative of a powerful counting principle.

A k-to-1 function maps exactly k elements of the domain to every element of the codomain. For example, the function mapping each ear to its owner is 2-to-1. Similarly, the function mapping each finger to its owner is 10-to-1, and the function mapping each finger and toe to its owner is 20-to-1. The general rule is:

Rule 14.4.1 (Division Rule). If $f : A \to B$ is k-to-1, then $|A| = k \cdot |B|$.

For example, suppose A is the set of ears in the room and B is the set of people. There is a 2-to-1 mapping from ears to people, so by the Division Rule, $|A| = 2 \cdot |B|$. Equivalently, $|B| = |A|/2$, expressing what we knew all along: the number of people is half the number of ears. Unlikely as it may seem, many counting problems are made much easier by initially counting every item multiple times and then correcting the answer using the Division Rule. Let's look at some examples.

Another Chess Problem

In how many different ways can you place two identical rooks on a chessboard so that they do not share a row or column? A valid configuration is shown in Figure 14.2(a), and an invalid configuration is shown in Figure 14.2(b).

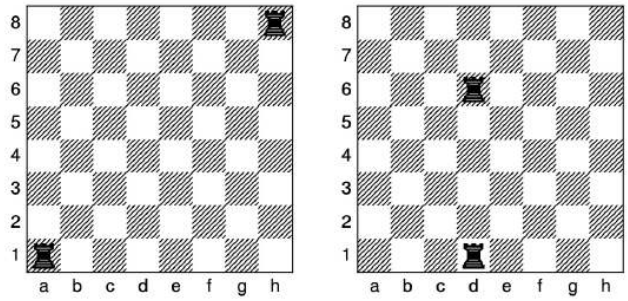


Figure 14.2 Two ways to place 2 rooks on a chessboard. The configuration in (b) is invalid because the rooks are in the same column.

Let A be the set of all sequences

$$(r_1, c_1, r_2, c_2)$$

where r_1 and r_2 are distinct rows and c_1 and c_2 are distinct columns. Let B be the set of all valid rook configurations. There is a natural function f from set A to set B; in particular, f maps the sequence (r_1, c_1, r_2, c_2) to a configuration with one rook in row r_1, column c_1 and the other rook in row r_2, column c_2.

But now there's a snag. Consider the sequences:

$$(1, a, 8, h) \quad \text{and} \quad (8, h, 1, a)$$

The first sequence maps to a configuration with a rook in the lower-left corner and a rook in the upper-right corner. The second sequence maps to a configuration with a rook in the upper-right corner and a rook in the lower-left corner. The problem is that those are two different ways of describing the *same* configuration! In fact, this arrangement is shown in Figure 14.2(a).

More generally, the function f maps exactly two sequences to *every* board configuration; f is a 2-to-1 function. Thus, by the quotient rule, $|A| = 2 \cdot |B|$. Rearranging terms gives:

$$|B| = \frac{|A|}{2} = \frac{(8 \cdot 7)^2}{2}.$$

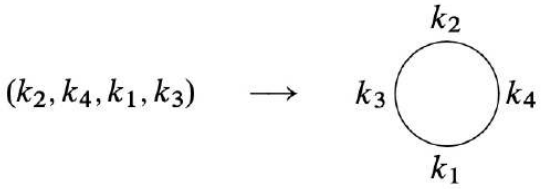
On the second line, we've computed the size of A using the General Product Rule just as in the earlier chess problem.

Knights of the Round Table

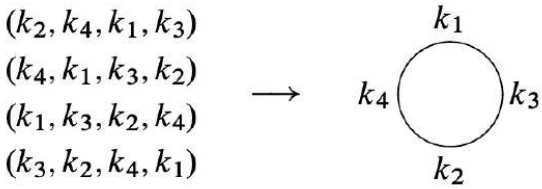
In how many ways can King Arthur arrange to seat his n different knights at his round table? A seating defines who sits where. Two seatings are considered to be the same *arrangement* if each knight sits between the same two knights in both seatings. An equivalent way to say this is that two seatings yield the same arrangement when they yield the same sequence of knights starting at knight number 1 and going clockwise around the table. For example, the following two seatings determine the same arrangement:



A seating is determined by the sequence of knights going clockwise around the table starting at the top seat. So seatings correspond to permutations of the knights, and there are $n!$ of them. For example,



Two seatings determine the same arrangement if they are the same when the table is rotated so knight 1 is at the top seat. For example with $n = 4$, there are 4 different sequences that correspond to the seating arrangement:



This mapping from seating to arrangments is actually an n-to-1 function, since all n cyclic shifts of the sequence of knights in the seating map to the same arrangement. Therefore, by the division rule, the number of circular seating arrangements is:

Misplaced '#'

This page titled 14.4: The Division Rule is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.5: Counting Subsets

How many k-element subsets of an n-element set are there? This question arises all the time in various guises:

- In how many ways can I select 5 books from my collection of 100 to bring on vacation?
- How many different 13-card bridge hands can be dealt from a 52-card deck?
- In how many ways can I select 5 toppings for my pizza if there are 14 available toppings?

This number comes up so often that there is a special notation for it:

$$\binom{n}{k} ::= \text{the number of } k\text{-element subsets of an } n\text{-element set.}$$

The expression $\binom{n}{k}$ is read "n choose k." Now we can immediately express the answers to all three questions above:

I can select 5 books from 100 in $\binom{100}{5}$ ways.

There are $\binom{52}{13}$ different bridge hands.

There are $\binom{14}{5}$ different 5-topping pizzas, if 14 toppings are available.

The Subset Rule

We can derive a simple formula for the n choose k number using the Division Rule. We do this by mapping any permutation of an n -element set $\{a_1, \ldots, a_n\}$ into a k-element subset simply by taking the first k elements of the permutation. That is, the permutation $a_1 a_2 \ldots a_n$ will map to the set $\{a_1, a_2, \ldots, a_n\}$.

Notice that any other permutation with the same first k elements $a_1, \ldots, a_n$ *in any order* and the same remaining elements $n-k$ elements *in any order* will also map to this set. What's more, a permutation can only map to $\{a_1, a_2, \ldots, a_n\}$ if its first k elements are the elements $a_1, \ldots, a_n$ in some order. Since there are $k!$ possible permutations of the first k elements and $(n-k)!$ permutations of the remaining elements, we conclude from the Product Rule that exactly $k!(n-k)!$ permutations of the n-element set map to the particular subset, S. In other words, the mapping from permutations to k-element subsets is $k!(n-k)!$-to-1.

But we know there are $n!$ permutations of an n-element set, so by the Division Rule, we conclude that

$$n! = k!(n-k)!\binom{n}{k}$$

which proves:

Rule 14.5.1 (Subset Rule). The number of k-element subsets of an n-element set is

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Notice that this works even for 0-element subsets: $n!/0!n! = 1$. Here we use the fact that $0!$ is a product of 0 terms, which by convention[2] equals 1.

Bit Sequences

How many n-bit sequences contain exactly k ones? We've already seen the straightforward bijection between subsets of an n-element set and n-bit sequences. For example, here is a 3-element subset of $\{x_1, x_2, \ldots, x_8\}$ and the associated 8-bit sequence:

$$\begin{array}{c} \{x_1, \quad x_4, x_5 \quad\} \\ (1,0,0,1,1,0,0,0) \end{array}$$

Notice that this sequence has exactly 3 ones, each corresponding to an element of the 3-element subset. More generally, the n-bit sequences corresponding to a k-element subset will have exactly k ones. So by the Bijection Rule,

Corollary 14.5.2. *The number of n-bit sequences with exactly k ones is $\binom{n}{k}$.*

Also, the bijection between selections of flavored donuts and bit sequences of Lemma 14.1.1 now implies,

Corollary 14.5.3. *The number of ways to select n donuts when k flavors are available is*

$$\binom{n+(k-1)}{n}.$$

[2]We don't use it here, but a *sum* of zero terms equals 0.

This page titled 14.5: Counting Subsets is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.6: Sequences with Repetitions

Sequences of Subsets

Choosing a k-element subset of an n-element set is the same as splitting the set into a pair of subsets: the first subset of size k and the second subset consisting of the remaining $n-k$ elements. So, the Subset Rule can be understood as a rule for counting the number of such splits into pairs of subsets.

We can generalize this to a way to count splits into more than two subsets. Let A be an n-element set and $k_1, k_2, \ldots, k_m$ be nonnegative integers whose sum is n. *$A(k_1, k_2, \ldots, k_m)$-split of A* is a sequence

$$(A_1, A_2, \ldots, A_m)$$

where the A_i are disjoint subsets of A and $|A_I| = k_i$ for $i = 1, \ldots, m$.

To count the number of splits we take the same approach as for the Subset Rule. Namely, we map any permutation $a_1 a_2 \ldots a_n$ of an n -element set A into a $(k_1, k_2, \ldots, k_m)$-split by letting the 1st subset in the split be the first k_1 elements of the permutation, the 2nd subset of the split be the next k_2 elements, . . . , and the mth subset of the split be the final k_m elements of the permutation. This map is a $k_1!k_2!\cdots k_m!$-to-1 function from the $n!$ permutations to the $(k_1, k_2, \ldots, k_m)$- splits of A, so from the Division Rule we conclude the *Subset Split Rule*:

Definition 14.6.1

For $n, k_1, \ldots, k_m \in \mathbb{N}$, such that $k_1 + k_2 + \cdots + k_m = n$, define the *multinomial coefficient*

$$nchoosek_1, k_2, \ldots, k_m ::= \frac{n!}{k_1!k_2! \ldots k_m!}.$$

Rule 14.6.2 (Subset Split Rule). *The number of $(k_1, k_2, \ldots, k_m)$-splits of an nelement set is*

$$\binom{n}{k_1, \ldots, k_m}.$$

The Bookkeeper Rule

We can also generalize our count of n-bit sequences with k ones to counting sequences of n letters over an alphabet with more than two letters. For example, how many sequences can be formed by permuting the letters in the 10-letter word BOOKKEEPER?

Notice that there are 1 B, 2 O's, 2 K's, 3 E's, 1 P, and 1 R in BOOKKEEPER. This leads to a straightforward bijection between permutations of BOOKKEEPER and (1,2,2,3,1,1)-splits of $\{1, 2, \ldots, 10\}$. Namely, map a permutation to the sequence of sets of positions where each of the different letters occur.

For example, in the permutation BOOKKEEPER itself, the B is in the 1st position, the O's occur in the 2nd and 3rd positions, K's in 4th and 5th, the E's in the 6th, 7th and 9th, P in the 8th, and R is in the 10th position. So BOOKKEEPER maps to

$$(\{1\}, \{2, 3\}, \{4, 5\}, \{6, 7, 9\}, \{8\}, \{10\}).$$

From this bijection and the Subset Split Rule, we conclude that the number of ways to rearrange the letters in the word BOOKKEEPER is:

$$\frac{\overbrace{10!}^{\text{total letters}}}{\underbrace{1!}_{\text{B's}}\ \underbrace{2!}_{\text{O's}}\ \underbrace{2!}_{\text{K's}}\ \underbrace{3!}_{\text{E's}}\ \underbrace{1!}_{\text{P's}}\ \underbrace{1!}_{\text{R's}}}$$

This example generalizes directly to an exceptionally useful counting principle which we will call the

Rule 14.6.3 (Bookkeeper Rule). Let $l_1, \ldots, l_m$ be distinct elements. The number of sequences with k_1 occurrences of l_1, and k_2 occurrences of l_2, ..., and k_m occurrences of l_m is

$$\binom{k_1 + k_2 + \cdots + k_m}{k_1, \ldots, k_m}.$$

For example, suppose you are planning a 20-mile walk, which should include 5 northward miles, 5 eastward miles, 5 southward miles, and 5 westward miles. How many different walks are possible?

There is a bijection between such walks and sequences with 5 N's, 5 E's, 5 S's, and 5 W's. By the Bookkeeper Rule, the number of such sequences is:

$$\frac{20!}{(5!)^4}.$$

A Word about Words

Someday you might refer to the Subset Split Rule or the Bookkeeper Rule in front of a roomful of colleagues and discover that they're all staring back at you blankly. This is not because they're dumb, but rather because we made up the name "Bookkeeper Rule." However, the rule is excellent and the name is apt, so we suggest that you play through: "You know? The Bookkeeper Rule? Don't you guys know *anything*?"

The Bookkeeper Rule is sometimes called the "formula for permutations with indistinguishable objects." The size k subsets of an n-element set are sometimes called *k-combinations*. Other similar-sounding descriptions are "combinations with repetition, permutations with repetition, r-permutations, permutations with indistinguishable objects," and so on. However, the counting rules we've taught you are sufficient to solve all these sorts of problems without knowing this jargon, so we won't burden you with it.

The Binomial Theorem

Counting gives insight into one of the basic theorems of algebra. A *binomial* is a sum of two terms, such as $a+b$. Now consider its 4th power, $(a+b)^4$.

By repeatedly using distributivity of products over sums to multiply out this 4th power expression completely, we get

$$\begin{aligned}(a+b)^4 = \quad & aaaa + aaab + aaba + aabb \\ & + abaa + abab + abba + abbb \\ & + baaa + baab + baba + babb \\ & + bbaa + bbab + bbba + bbbb\end{aligned} \tag{14.6.1}$$

Notice that there is one term for every sequence of a's and b's. So there are 2^4 terms, and the number of terms with k copies of b and $n-k$ copies of a is:

$$\frac{n!}{k!(n-k)!} = \binom{n}{k}$$

by the Bookkeeper Rule. Hence, the coefficient of $a^{n-k}b^k$ is $\binom{n}{k}$. So for $n=4$, this means:

$$(a+b)^4 = \binom{4}{0}\cdot a^4b^0 + \binom{4}{1}\cdot a^3b^1 + \binom{4}{2}\cdot a^2b^2 + \binom{4}{3}\cdot a^1b^3 + \binom{4}{4}\cdot a^0b^4$$

In general, this reasoning gives the Binomial Theorem:

Theorem 14.6.4

(Binomial Theorem). For all $n \in \mathbb{N}$ and $a, b \in \mathbb{R}$:

$$(a+b)^4 = \sum_{k=0}^{n} \binom{n}{k} a^{n-k}b^k$$

The Binomial Theorem explains why the n choose k number is called a *binomial coefficient*.

This reasoning about binomials extends nicely to *multinomials*, which are sums of two or more terms. For example, suppose we wanted the coefficient of

$$no^2k^2e^3pr$$

in the expansion of $(b+o+k+e+p+r)^{10}$. Each term in this expansion is a product of 10 variables where each variable is one of b, o, k, e, p, or r. Now, the coefficient of $bo^2k^2e^3pr$ is the number of those terms with exactly 1 b, 2 o's, 2 k's, 3 e's, 1 p, and 1 r. And

the number of such terms is precisely the number of rearrangements of the word BOOKKEEPER:

$$\binom{10}{1,2,2,3,1,1} = \frac{10!}{1!2!2!3!1!1!}$$

This reasoning extends to a general theorem:

Theorem 14.6.5

(Multinomial Theorem). For all $n \in \mathbb{N}$,

$$(z_1 + z_2 + \cdots + z_m)^n = \sum_{\substack{k1,\ldots,k_m \in \mathbb{N} \\ k_1+\cdots+k_m=m}} \binom{n}{k_1, k_2, \ldots, k_m} z_1^{k_1} z_2^{k_2} \cdots z_m^{k_m}.$$

This page titled 14.6: Sequences with Repetitions is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.7: Counting Practice - Poker Hands

Five-Card Draw is a card game in which each player is initially dealt a *hand* consisting of 5 cards from a deck of 52 cards.[3] The number of different hands in Five-Card Draw is the number of 5-element subsets of a 52-element set, which is

$$\binom{52}{5} = 2{,}598{,}960.$$

Let's get some counting practice by working out the number of hands with various special properties.

Hands with a Four-of-a-Kind

A *Four-of-a-Kind* is a set of four cards with the same rank. How many different hands contain a Four-of-a-Kind? Here are a couple examples:

$$\begin{array}{l}\{8\spadesuit, 8\diamondsuit, Q\heartsuit, 8\heartsuit, 8\clubsuit\} \\ \{A\clubsuit, 2\clubsuit, 2\heartsuit, 2\diamondsuit, 2\spadesuit\}\end{array} \tag{14.7.1}$$

As usual, the first step is to map this question to a sequence-counting problem. A hand with a Four-of-a-Kind is completely described by a sequence specifying:

1. The rank of the four cards.
2. The rank of the extra card.
3. The suit of the extra card.

Thus, there is a bijection between hands with a Four-of-a-Kind and sequences consisting of two distinct ranks followed by a suit. For example, the three hands above are associated with the following sequences:

$$\begin{array}{l}(8, Q, \heartsuit) \leftrightarrow \{8\spadesuit, 8\diamondsuit, 8\heartsuit, 8\clubsuit, Q\heartsuit\} \\ (2, A, \clubsuit) \leftrightarrow \{2\clubsuit, 2\heartsuit, 2\diamondsuit, 2\spadesuit, A\clubsuit\}\end{array} \tag{14.7.2}$$

Now we need only count the sequences. There are 13 ways to choose the first rank, 12 ways to choose the second rank, and 4 ways to choose the suit. Thus, by the Generalized Product Rule, there are $13 \cdot 12 \cdot 4 = 624$ hands with a Four-of-a-Kind. This means that only 1 hand in about 4165 has a Four-of-a-Kind. Not surprisingly, Four-of-a-Kind is considered to be a very good poker hand!

Hands with a Full House

A *Full House* is a hand with three cards of one rank and two cards of another rank. Here are some examples:

$$\begin{array}{l}\{2\spadesuit, 2\clubsuit, 2\diamondsuit, J\clubsuit, J\diamondsuit\} \\ \{5\diamondsuit, 5\clubsuit, 5\heartsuit, 7\heartsuit, 7\clubsuit\}\end{array} \tag{14.7.3}$$

Again, we shift to a problem about sequences. There is a bijection between Full Houses and sequences specifying:

1. The rank of the triple, which can be chosen in 13 ways.
2. The suits of the triple, which can be selected in $\binom{4}{3}$ ways.
3. The rank of the pair, which can be chosen in 12 ways.
4. The suits of the pair, which can be selected in $\binom{4}{2}$ ways.

The example hands correspond to sequences as shown below:

$$\begin{array}{l}(2, \{\spadesuit, \clubsuit, \diamondsuit\}, J, \{\clubsuit, \diamondsuit\}) \leftrightarrow \{2\spadesuit, 2\clubsuit, 2\diamondsuit, J\clubsuit, J\diamondsuit\} \\ (5, \{\diamondsuit, \clubsuit, \heartsuit\}, 7, \{\heartsuit, \clubsuit\}) \leftrightarrow \{5\diamondsuit, 5\clubsuit, 5\heartsuit, 7\heartsuit, 7\clubsuit\}\end{array} \tag{14.7.4}$$

By the Generalized Product Rule, the number of Full Houses is:

$$13 \cdot \binom{4}{3} \cdot 12 \cdot \binom{4}{2}$$

We're on a roll—but we're about to hit a speed bump.

Hands with Two Pairs

How many hands have *Two Pairs*; that is, two cards of one rank, two cards of another rank, and one card of a third rank? Here are examples:

$$\begin{array}{l}\{3\diamondsuit, 3\spadesuit, Q\diamondsuit, Q\heartsuit, A\clubsuit\}\\ \{9\heartsuit, 9\diamondsuit, 5\heartsuit, 5\clubsuit, K\spadesuit\}\end{array} \tag{14.7.5}$$

Each hand with Two Pairs is described by a sequence consisting of:

1. The rank of the first pair, which can be chosen in 13 ways.
2. The suits of the first pair, which can be selected $\binom{4}{2}$ ways.
3. The rank of the second pair, which can be chosen in 12 ways.
4. The suits of the second pair, which can be selected in $\binom{4}{2}$ ways.
5. The rank of the extra card, which can be chosen in 11 ways.
6. The suit of the extra card, which can be selected in $\binom{4}{1} = 4$ ways.

Thus, it might appear that the number of hands with Two Pairs is:

$$13 \cdot \binom{4}{2} \cdot 12 \cdot \binom{4}{2} \cdot 11 \cdot 4.$$

Wrong answer! The problem is that there is *not* a bijection from such sequences to hands with Two Pairs. This is actually a 2-to-1 mapping. For example, here are the pairs of sequences that map to the hands given above:

$$\begin{array}{ll}(3, \{\diamondsuit, \spadesuit\}, Q, \{\diamondsuit, \heartsuit\}, A, \clubsuit) & \searrow \\ & \{3\diamondsuit,\ 3\spadesuit,\ Q\diamondsuit,\ Q\heartsuit,\ A\clubsuit\} \\ (Q, \{\diamondsuit, \heartsuit\}, 3, \{\diamondsuit, \spadesuit\}, A, \clubsuit) & \nearrow \\ (9, \{\heartsuit, \diamondsuit\}, 5, \{\heartsuit, \clubsuit\}, K, \spadesuit) & \searrow \\ & \{9\heartsuit,\ 9\diamondsuit,\ 5\heartsuit,\ 5\clubsuit,\ K\spadesuit\} \\ (5, \{\heartsuit, \clubsuit\}, 9, \{\heartsuit, \diamondsuit\}, K, \spadesuit) & \nearrow\end{array}$$

The problem is that nothing distinguishes the first pair from the second. A pair of 5's and a pair of 9's is the same as a pair of 9's and a pair of 5's. We avoided this difficulty in counting Full Houses because, for example, a pair of 6's and a triple of kings is different from a pair of kings and a triple of 6's.

We ran into precisely this difficulty last time, when we went from counting arrangements of *different* pieces on a chessboard to counting arrangements of two *identical* rooks. The solution then was to apply the Division Rule, and we can do the same here. In this case, the Division rule says there are twice as many sequences as hands, so the number of hands with Two Pairs is actually:

$$\frac{13 \cdot \binom{4}{2} \cdot 12 \cdot \binom{4}{2} \cdot 11 \cdot 4}{2}.$$

Another Approach

The preceding example was disturbing! One could easily overlook the fact that the mapping was 2-to-1 on an exam, fail the course, and turn to a life of crime. You can make the world a safer place in two ways:

1. Whenever you use a mapping $f : A \to B$ to translate one counting problem to another, check that the same number of elements in A are mapped to each element in B. If k elements of A map to each of element of B, then apply the Division Rule using the constant k.
2. As an extra check, try solving the same problem in a different way. Multiple approaches are often available—and all had better give the same answer! (Sometimes different approaches give answers that *look* different, but turn out to be the same after some algebra.)

We already used the first method; let's try the second. There is a bijection between hands with two pairs and sequences that specify:

1. The ranks of the two pairs, which can be chosen in $\binom{13}{2}$ ways.
2. The suits of the lower-rank pair, which can be selected in $\binom{4}{2}$ ways.
3. The suits of the higher-rank pair, which can be selected in $\binom{4}{2}$ ways.

4. The rank of the extra card, which can be chosen in 11 ways.
5. The suit of the extra card, which can be selected in $\binom{4}{1} = 4$ ways.

For example, the following sequences and hands correspond:

$$\begin{array}{l}(\{3, Q\}, \{\diamondsuit, \spadesuit\}, \{\diamondsuit, \heartsuit\}, A, \clubsuit) \leftrightarrow \{3\diamondsuit, 3\spadesuit, Q\diamondsuit, Q\heartsuit, A\clubsuit\} \\ (\{9, 5\}, \{\heartsuit, \clubsuit\}, \{\heartsuit, \diamondsuit\}, K, \spadesuit) \leftrightarrow \{9\heartsuit, 9\diamondsuit, 5\heartsuit, 5\clubsuit, K\spadesuit\}\end{array} \tag{14.7.6}$$

Thus, the number of hands with two pairs is:

$$\binom{13}{2} \cdot \binom{4}{2} \cdot \binom{4}{2} \cdot 11 \cdot 4.$$

This is the same answer we got before, though in a slightly different form.

Hands with Every Suit

How many hands contain at least one card from every suit? Here is an example of such a hand:

$$\{7\diamondsuit, K\clubsuit, 3\diamondsuit, A\heartsuit, 2\spadesuit\}$$

Each such hand is described by a sequence that specifies:

1. The ranks of the diamond, the club, the heart, and the spade, which can be selected in $13 \cdot 13 \cdot 13 \cdot 13 = 13^4$ ways.
2. The suit of the extra card, which can be selected in 4 ways.
3. The rank of the extra card, which can be selected in 12 ways.

For example, the hand above is described by the sequence:

$$(7, K, A, 2, \diamondsuit, 3) \leftrightarrow \{7\diamondsuit, K\clubsuit, A\heartsuit, 2\spadesuit, 3\diamondsuit\}.$$

Are there other sequences that correspond to the same hand? There is one more! We could equally well regard either the $3\diamondsuit$ or the $7\diamondsuit$ as the extra card, so this is actually a 2-to-1 mapping. Here are the two sequences corresponding to the example hand:

$$\begin{array}{l}(7, K, A, 2, \diamondsuit, 3) \searrow \\ (3, K, A, 2, \diamondsuit, 7) \nearrow\end{array} \quad \{7\diamondsuit, K\clubsuit, A\heartsuit, 2\spadesuit, 3\diamondsuit\}$$

Therefore, the number of hands with every suit is:

$$\frac{13^4 \cdot 4 \cdot 12}{2}.$$

[3]There are 52 cards in a standard deck. Each card has a *suit* and a *rank*. There are four suits:

$\spadesuit$ (spades) $\heartsuit$ (hearts) $\clubsuit$ (clubs) $\diamondsuit$ (diamonds)

And there are 13 ranks, listed here from lowest to highest:

$$\overset{\text{Ace}}{A}, 2, 3, 4, 5, 6, 7, 8, 9, \overset{\text{Jack}}{J}, \overset{\text{Queen}}{Q}, \overset{\text{King}}{K}.$$

Thus, for example, $8\heartsuit$ is the 8 of hearts and $A\spadesuit$ is the ace of spades.

This page titled 14.7: Counting Practice - Poker Hands is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.8: The Pigeonhole Principle

Here is an old puzzle:

> A drawer in a dark room contains red socks, green socks, and blue socks. How many socks must you withdraw to be sure that you have a matching pair?

For example, picking out three socks is not enough; you might end up with one red, one green, and one blue. The solution relies on the

Pigeonhole Principle

If there are more pigeons than holes they occupy, then at least two pigeons must be in the same hole.

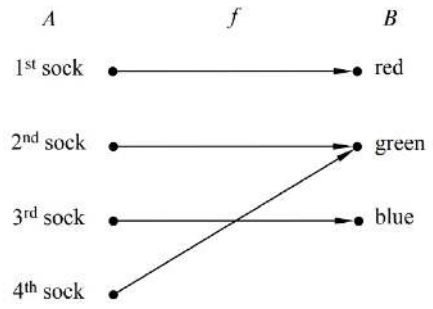


Figure 14.3 One possible mapping of four socks to three colors.

What pigeons have to do with selecting footwear under poor lighting conditions may not be immediately obvious, but if we let socks be pigeons and the colors be three pigeonholes, then as soon as you pick four socks, there are bound to be two in the same hole, that is, with the same color. So four socks are enough to ensure a matched pair. For example, one possible mapping of four socks to three colors is shown in Figure 14.3.

A rigorous statement of the Principle goes this way:

Rule 14.8.1 (Pigeonhole Principle). *If $|A| > |B|$, then for every total function $f : A \to B$, there exist two different elements of A that are mapped by f to the same element of B.*

Stating the Principle this way may be less intuitive, but it should now sound familiar: it is simply the contrapositive of the Mapping Rules injective case (4.5.2). Here, the pigeons form set A, the pigeonholes are the set B, and f describes which hole each pigeon occupies.

Mathematicians have come up with many ingenious applications for the pigeonhole principle. If there were a cookbook procedure for generating such arguments, we'd give it to you. Unfortunately, there isn't one. One helpful tip, though: when you try to solve a problem with the pigeonhole principle, the key is to clearly identify three things:

1. The set A (the pigeons).
2. The set B (the pigeonholes).
3. The function f (the rule for assigning pigeons to pigeonholes).

Hairs on Heads

There are a number of generalizations of the pigeonhole principle. For example:

Rule 14.8.2 (Generalized Pigeonhole Principle). *If $|A| > k \cdot |B|$, then every total function $f : A \to B$ maps at least $k+1$ different elements of A to the same element of B.*

For example, if you pick two people at random, surely they are extremely unlikely to have *exactly* the same number of hairs on their heads. However, in the remarkable city of Boston, Massachusetts, there is a group of *three* people who have exactly the same number of hairs! Of course, there are many completely bald people in Boston, and they all have zero hairs. But we're talking about non-bald people; say a person is non-bald if they have at least ten thousand hairs on their head.

Boston has about 500,000 non-bald people, and the number of hairs on a person's head is at most 200,000. Let A be the set of non-bald people in Boston, let $B = \{10,000, 10,001, \ldots, 200,000\}$, and let f map a person to the number of hairs on his or her head. Since $|A| > 2|B|$, the Generalized Pigeonhole Principle implies that at least three people have exactly the same number of hairs. We don't know who they are, but we know they exist!

Subsets with the Same Sum

For your reading pleasure, we have displayed ninety 25-digit numbers in Figure 14.4. Are there two different subsets of these 25-digit numbers that have the same sum? For example, maybe the sum of the last ten numbers in the first column is equal to the sum of the first eleven numbers in the second column?

0020480135385502964448038	3171004832173501394113017
5763257331083479647409398	8247331000042995311646021
0489445991866915676240992	3208234421597368647019265
5800949123548989122628663	8496243997123475922766310
1082662032430379651370981	3437254656355157864869113
6042900801199280218026001	8518399140676002660747477
1178480894769706178994993	3574883393058653923711365
6116171789137737896701405	8543691283470191452333763
1253127351683239693851327	3644909946040480189969149
6144868973001582369723512	8675309258374137092461352
1301505129234077811069011	3790044132737084094417246
6247314593851169234746152	8694321112363996867296665
1311567111143866433882194	3870332127437971355322815
6814428944266874963488274	8772321203608477245851154
1470029452721203587686214	4080505804577801451363100
6870852945543886849147881	8791422161722582546341091
1578271047286257499433886	4167283461025702348124920
6914955508120950093732397	9062628024592126283973285
1638243921852176243192354	4235996831123777788211249
6949632451365987152423541	9137845566925526349897794
1763580219131985963102365	4670939445749439042111220
7128211143613619828415650	9153762966803189291934419
1826227795601842231029694	4815379351865384279613427
7173920083651862307925394	9270880194077636406984249
1843971862675102037201420	4837052948212922604442190
7215654874211755676220587	9324301480722103490379204
2396951193722134526177237	5106389423855018550671530
7256932847164391040233050	9436090832146695147140581
2781394568268599801096354	5142368192004769218069910
7332822657075235431620317	9475308159734538249013238
2796605196713610405408019	5181234096130144084041856
7426441829541573444964139	9492376623917486974923202
2931016394761975263190347	5198267398125617994391348
7632198126531809327186321	9511972558779880288252979
2933458058294405155197296	5317592940316231219758372
7712154432211912882310511	9602413424619187112552264
3075514410490975920315348	5384358126771794128356947
7858918664240262356610010	9631217114906129219461111
8149436716871371161932035	3157693105325111284321993
3111474985252793452860017	5439211712248901995423441
7898156786763212963178679	9908189853102753335981319
3145621587936120118438701	5610379826092838192760458
8147591017037573337848616	9913237476341764299813987
3148901255628881103198549	5632317555465228677676044
5692168374637019617423712	8176063831682536571306791

Figure 14.4 Ninety 25-digit numbers. Can you find two different subsets of these numbers that have the same sum?

Finding two subsets with the same sum may seem like a silly puzzle, but solving these sorts of problems turns out to be useful in diverse applications such as finding good ways to fit packages into shipping containers and decoding secret messages.

It turns out that it is hard to find different subsets with the same sum, which is why this problem arises in cryptography. But it is easy to prove that two such subsets *exist*. That's where the Pigeonhole Principle comes in.

Let A be the collection of all subsets of the 90 numbers in the list. Now the sum of any subset of numbers is at most $90 \cdot 10^{25}$, since there are only 90 numbers and every 25-digit number is less than 10^{25}. So let B be the set of integers $\{0, 1, \ldots, 90 \cdot 10^{25}\}$, and let f map each subset of numbers (in A) to its sum (in B).

We proved that an n-element set has 2^n different subsets in Section 14.2. Therefore:

$$|A| = 2^{90} \geq 1.237 \times 10^{27}$$

On the other hand:

$$|B| = 90 \cdot 10^{25} + 1 \leq 0.901 \times 10^{27}.$$

Both quantities are enormous, but $|A|$ is a bit greater than $|B|$. This means that f maps at least two elements of A to the same element of B. In other words, by the Pigeonhole Principle, two different subsets must have the same sum!

Notice that this proof gives no indication *which* two sets of numbers have the same sum. This frustrating variety of argument is called a *nonconstructive proof.*

The $100 prize for two same-sum subsets

To see if it was possible to actually find two different subsets of the ninety 25-digit numbers with the same sum, we offered a $100 prize to the first student who did it. We didn't expect to have to pay off this bet, but we underestimated the ingenuity and initiative of the students. One computer science major wrote a program that cleverly searched only among a reasonably small set of "plausible" sets, sorted them by their sums, and actually found a couple with the same sum. He won the prize. A few days later, a math major figured out how to reformulate the sum problem as a "lattice basis reduction" problem; then he found a software package implementing an efficient basis reduction procedure, and using it, he very quickly found lots of pairs of subsets with the same sum. He didn't win the prize, but he got a standing ovation from the class—staff included.

The $500 Prize for Sets with Distinct Subset Sums

How can we construct a set of n positive integers such that all its subsets have *distinct* sums? One way is to use powers of two:

$$\{1, 2, 4, 8, 16\}$$

This approach is so natural that one suspects all other such sets must involve larger numbers. (For example, we could safely replace 16 by 17, but not by 15.) Remarkably, there are examples involving *smaller* numbers. Here is one:

$$\{6, 9, 11, 12, 13\}$$

One of the top mathematicians of the Twentieth Century, Paul Erdős, conjectured in 1931 that there are no such sets involving *significantly* smaller numbers. More precisely, he conjectured that the largest number in such a set must be greater than $c2^n$ for some constant $c > 0$. He offered $500 to anyone who could prove or disprove his conjecture, but the problem remains unsolved.

Magic Trick

A Magician sends an Assistant into the audience with a deck of 52 cards while the Magician looks away.

Five audience members each select one card from the deck. The Assistant then gathers up the five cards and holds up four of them so the Magician can see them. The Magician concentrates for a short time and then correctly names the secret, fifth card!

Since we don't really believe the Magician can read minds, we know the Assistant has somehow communicated the secret card to the Magician. Real Magicians and Assistants are not to be trusted, so we expect that the Assistant would secretly signal the Magician with coded phrases or body language, but for this trick they don't have to cheat. In fact, the Magician and Assistant could be kept out of sight of each other while some audience member holds up the 4 cards designated by the Assistant for the Magician to see.

Of course, without cheating, there is still an obvious way the Assistant can communicate to the Magician: he can choose any of the $4! = 24$ permutations of the 4 cards as the order in which to hold up the cards. However, this alone won't quite work: there are 48 cards remaining in the deck, so the Assistant doesn't have enough choices of orders to indicate exactly what the secret card is (though he could narrow it down to two cards).

The Secret

The method the Assistant can use to communicate the fifth card exactly is a nice application of what we know about counting and matching.

The Assistant has a second legitimate way to communicate: he can choose *which of the five cards to keep hidden*. Of course, it's not clear how the Magician could determine which of these five possibilities the Assistant selected by looking at the four visible cards, but there is a way, as we'll now explain.

The problem facing the Magician and Assistant is actually a bipartite matching problem. Each vertex on the left will correspond to the information available to the Assistant, namely, a *set* of 5 cards. So the set X of left hand vertices will have $\binom{52}{5}$ elements.

Each vertex on the right will correspond to the information available to the Magician, namely, a *sequence* of 4 distinct cards. So the set Y of right hand vertices will have $52 \cdot 51 \cdot 50 \cdot 49$ elements. When the audience selects a set of 5 cards, then the Assistant must reveal a sequence of 4 cards from that hand. This constraint is represented by having an edge between a set of 5 cards on the left and a sequence of 4 cards on the right precisely when every card in the sequence is also in the set. This specifies the bipartite graph. Some edges are shown in the diagram in Figure 14.5.

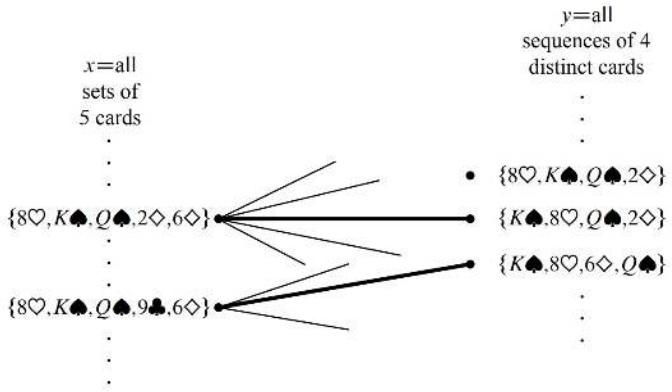


Figure 14.5 The bipartite graph where the nodes on the left correspond to sets of 5 cards and the nodes on the right correspond to *sequences* of 4 cards. There is an edge between a set and a sequence whenever all the cards in the sequence are contained in the set.

For example,

$$\{8\heartsuit, K\spadesuit, Q\spadesuit, 2\diamondsuit, 6\diamondsuit\} \tag{14.8.1}$$

is an element of X on the left. If the audience selects this set of 5 cards, then there are many different 4-card sequences on the right in set Y that the Assistant could choose to reveal, including $(8\heartsuit, K\spadesuit, Q\spadesuit, 2\diamondsuit)$, $(K\spadesuit, 8\heartsuit, Q\spadesuit, 2\diamondsuit)$, and $(K\spadesuit, 8\heartsuit, 6\diamondsuit, Q\spadesuit)$.

What the Magician and his Assistant need to perform the trick is a *matching* for the X vertices. If they agree in advance on some matching, then when the audience selects a set of 5 cards, the Assistant reveals the matching sequence of 4 cards. The Magician uses the matching to find the audience's chosen set of 5 cards, and so he can name the one not already revealed.

For example, suppose the Assistant and Magician agree on a matching containing the two bold edges in Figure 14.5. If the audience selects the set

$$\{8\heartsuit, K\spadesuit, Q\spadesuit, 9\clubsuit, 6\diamondsuit\}, \tag{14.8.2}$$

then the Assistant reveals the corresponding sequence

$$(K\spadesuit, 8\heartsuit, 6\diamondsuit, Q\spadesuit). \tag{14.8.3}$$

Using the matching, the Magician sees that the hand (14.8.2) is matched to the sequence (14.8.3), so he can name the one card in the corresponding set not already revealed, namely, the $9\clubsuit$. Notice that the fact that the sets are matched, that is, that different sets are paired with *distinct* sequences, is essential. For example, if the audience picked the previous hand (14.8.1), it would be possible for the Assistant to reveal the same sequence (14.8.3), but he better not do that; if he did, then the Magician would have no way to tell if the remaining card was the $9\clubsuit$ or the $2\diamondsuit$.

So how can we be sure the needed matching can be found? The answer is that each vertex on the left has degree $5 \cdot 4! = 120$, since there are five ways to select the card kept secret and there are $4!$ permutations of the remaining 4 cards. In addition, each vertex on the right has degree 48, since there are 48 possibilities for the fifth card. So this graph is *degree-constrained* according to Definition 11.5.5, and so has a matching by Theorem 11.5.6.

In fact, this reasoning shows that the Magician could still pull off the trick if 120 cards were left instead of 48, that is, the trick would work with a deck as large as 124 different cards—without any magic!

The Real Secret

But wait a minute! It's all very well in principle to have the Magician and his Assistant agree on a matching, but how are they supposed to remember a matching with $\binom{52}{5} = 2,598,960$ edges? For the trick to work in practice, there has to be a way to match

hands and card sequences mentally and on the fly.

We'll describe one approach. As a running example, suppose that the audience selects:

$$10\heartsuit \quad 9\diamondsuit \quad 3\heartsuit \quad Q\spadesuit \quad J\diamondsuit.$$

- The Assistant picks out two cards of the same suit. In the example, the assistant might choose the $3\heartsuit$ and $10\heartsuit$. This is always possible because of the Pigeonhole Principle—there are five cards and 4 suits so two cards must be in the same suit.
- The Assistant locates the ranks of these two cards on the cycle shown in Figure 14.6. For any two distinct ranks on this cycle, one is always between 1 and 6 hops clockwise from the other. For example, the $3\heartsuit$ is 6 hops clockwise from the $10\heartsuit$.

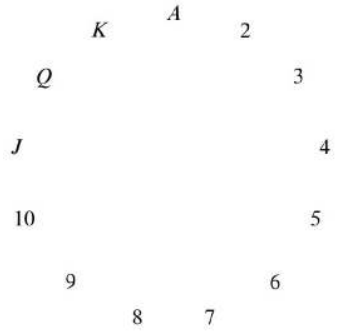


Figure 14.6 The 13 card ranks arranged in cyclic order.

- The more counterclockwise of these two cards is revealed first, and the other becomes the secret card. Thus, in our example, the $10\heartsuit$ would be revealed, and the $3\heartsuit$ would be the secret card. Therefore:
  - The suit of the secret card is the same as the suit of the first card revealed.
  - The rank of the secret card is between 1 and 6 hops clockwise from the rank of the first card revealed.
- All that remains is to communicate a number between 1 and 6. The Magician and Assistant agree beforehand on an ordering of all the cards in the deck from smallest to largest such as:

$$A\clubsuit A\diamondsuit A\heartsuit A\spadesuit 2\clubsuit 2\diamondsuit 2\heartsuit 2\spadesuit \ldots K\heartsuit K\spadesuit$$

  The order in which the last three cards are revealed communicates the number according to the following scheme:

$$\begin{array}{l}(\text{small, medium, large}) = 1\\ (\text{small, large, medium}) = 2\\ (\text{medium, small, large}) = 3\\ (\text{medium, large, small}) = 4\\ (\text{large, small, medium}) = 5\\ (\text{large, medium, small}) = 6\end{array} \tag{14.8.4}$$

  In the example, the Assistant wants to send 6 and so reveals the remaining three cards in large, medium, small order. Here is the complete sequence that the Magician sees:

$$10\heartsuit \quad Q\spadesuit \quad J\diamondsuit \quad 9\diamondsuit$$

- The Magician starts with the first card, $10\heartsuit$, and hops 6 ranks clockwise to reach $3\heartsuit$, which is the secret card!

So that's how the trick can work with a standard deck of 52 cards. On the other hand, Hall's Theorem implies that the Magician and Assistant can *in principle* perform the trick with a deck of up to 124 cards. It turns out that there is a method which they could actually learn to use with a reasonable amount of practice for a 124-card deck, but we won't explain it here.

The Same Trick with Four Cards?

Suppose that the audience selects only *four* cards and the Assistant reveals a sequence of *three* to the Magician. Can the Magician determine the fourth card?

Let X be all the sets of four cards that the audience might select, and let Y be all the sequences of three cards that the Assistant might reveal. Now, on one hand, we have

$$|X| = \binom{52}{4} = 270,725$$

by the Subset Rule. On the other hand, we have

$$|Y| = 52 \cdot 51 \cdot 50 = 132,600$$

by the Generalized Product Rule. Thus, by the Pigeonhole Principle, the Assistant must reveal the *same* sequence of three cards for at least

$$\left\lceil \frac{270,725}{132,600} \right\rceil = 3$$

different four-card hands. This is bad news for the Magician: if he sees that sequence of three, then there are at least three possibilities for the fourth card which he cannot distinguish. So there is no legitimate way for the Assistant to communicate exactly what the fourth card is!

This page titled 14.8: The Pigeonhole Principle is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.9: Inclusion-Exclusion

How big is a union of sets? For example, suppose there are 60 math majors, 200 EECS majors, and 40 physics majors. How many students are there in these three departments? Let M be the set of math majors, E be the set of EECS majors, and P be the set of physics majors. In these terms, we're asking for $|M \cup E \cup P|$.

The Sum Rule says that if M, E, and P are disjoint, then the sum of their sizes is

$$|M \cup E \cup P| = |M| + |E| + |P|.$$

However, the sets M, E, and P might *not* be disjoint. For example, there might be a student majoring in both math and physics. Such a student would be counted twice on the right side of this equation, once as an element of M and once as an element of P. Worse, there might be a triple-major[5] counted *three* times on the right side!

Our most-complicated counting rule determines the size of a union of sets that are not necessarily disjoint. Before we state the rule, let's build some intuition by considering some easier special cases: unions of just two or three sets.

Union of Two Sets

For two sets, S_1 and S_2, the *Inclusion-Exclusion Rule* is that the size of their union is:

Intuitively, each element of S_1 accounted for in the first term, and each element of S_2 is accounted for in the second term. Elements in *both* S_1 and S_2 are counted *twice*—once in the first term and once in the second. This double-counting is corrected by the final term.

Union of Three Sets

So how many students are there in the math, EECS, and physics departments? In other words, what is $|M \cup E \cup P|$ if:

$$\begin{aligned} |M| &= 60 \\ |E| &= 200 \\ |P| &= 40. \end{aligned} \tag{14.9.1}$$

The size of a union of three sets is given by a more complicated Inclusion-Exclusion formula:

$$\begin{aligned} |S_1 \cup S_2 \cup S_3| = {} & |S_1| + |S_2| + |S_3| \\ & - |S_1 \cap S_2| - |S_1 \cap S_3| - |S_2 \cap S_3| \\ & + |S_1 \cap S_2 \cap S_3| \end{aligned} \tag{14.9.2}$$

Remarkably, the expression on the right accounts for each element in the union of S_1, S_2, and S_3 exactly once. For example, suppose that x is an element of all three sets. Then x is counted three times (by the $|S_1|, |S_2|$, and $|S_3|$ terms), subtracted off three times (by the $|S_1 \cap S_2|, |S_1 \cap S_3|$, and $|S_2 \cap S_3|$ terms), and then counted once more (by the $|S_1 \cap S_2 \cap S_3|$ term). The net effect is that x is counted just once.

If x is in two sets (say, S_1 and S_2), then x is counted twice (by the $|S_1|$ and $|S_2|$ terms) and subtracted once (by the $|S_1 \cap S_2|$ term). In this case, x does not contribute to any of the other terms, since $x \notin S_3$.

So we can't answer the original question without knowing the sizes of the various intersections. Let's suppose that there are:

$$\begin{aligned} 4 &\quad \text{math - EECS double majors} \\ 3 &\quad \text{math - physics double majors} \\ 11 &\quad \text{EECS - physics double majors} \\ 2 &\quad \text{triple majors} \end{aligned} \tag{14.9.3}$$

Then $|M \cap E| = 4 + 2, |M \cap P| = 3 + 2, |E \cap P| = 11 + 2$, and $|M \cap E \cap P| = 2$. Plugging all this into the formula gives:

$$\begin{aligned} |M \cup E \cup P| &= |M| + |E| + |P| - |M \cap E| - |M \cap P| - |E \cap P| \\ &\quad + |M \cap E \cap P| \\ &= 60 + 200 + 40 - 6 - 5 - 13 + 2 \\ &= 278 \end{aligned} \tag{14.9.4}$$

Sequences with 42, 04, or 60

In how many permutations of the set $\{0, 1, 2, \ldots, 9\}$ do either 4 and 2, 0 and 4, or 6 and 0 appear consecutively? For example, none of these pairs appears in:

$$(7, 2, 9, 5, 4, 1, 3, 8, 0, 6).$$

The 06 at the end doesn't count; we need 60. On the other hand, both 04 and 60 appear consecutively in this permutation:

$$(7, 2, 5, \underline{6}, \underline{0}, \underline{4}, 3, 8, 1, 9).$$

Let P_{42} be the set of all permutations in which 42 appears. Define P_{60} and P_{04} similarly. Thus, for example, the permutation above is contained in both P_{60} and P_{04}, but not P_{42}. In these terms, we're looking for the size of the set $P_{42} \cup P_{04} \cup P_{60}$.

First, we must determine the sizes of the individual sets, such as P_{60}. We can use a trick: group the 6 and 0 together as a single symbol. Then there is an immediate bijection between permutations of $\{0, 1, 2, \ldots, 9\}$ containing 6 and 0 consecutively and permutations of:

$$\{60, 1, 2, 3, 4, 5, 7, 8, 9\}.$$

For example, the following two sequences correspond:

$$(7, 2, 5, \underline{6}, \underline{0}, 4, 3, 8, 1, 9) \longleftrightarrow (7, 2, 5, \underline{60}, 4, 3, 8, 1, 9)$$

There are $9!$ permutations of the set containing 60, so $|P_{60}| = 9!$ by the Bijection Rule. Similarly, $|P_{04}| = |P_{42}| = 9!$ as well.

Next, we must determine the sizes of the two-way intersections, such as $P_{42} \cap P_{60}$. Using the grouping trick again, there is a bijection with permutations of the set:

$$\{42, 60, 1, 3, 5, 7, 8, 9\}.$$

Thus, $|P_{42} \cap P_{60}| = 8!$. Similarly, $|P_{60} \cap P_{04}| = 8!$ by a bijection with the set:

$$\{604, 1, 2, 3, 5, 7, 8, 9\}.$$

And $|P_{42} \cap P_{04}| = 8!$ as well by a similar argument. Finally, note that $|P_{60} \cap P_{04} \cap P_{42}| = 7!$ by a bijection with the set:

$$\{6042, 1, 3, 5, 7, 8, 9\}.$$

Plugging all this into the formula gives:

$$|P_{42} \cap P_{04} \cap P_{60}| = 9! + 9! + 9! - 8! - 8! - 8! + 7!.$$

Union of n Sets

The size of a union of n sets is given by the following rule.

Rule 14.9.1 (Inclusion-Exclusion).

$$\begin{aligned} & |S_1 \cup S_2 \cup \cdots \cup S_n| = \\ & \textit{the sum of the sizes of the individual sets} \\ \textbf{minus}\quad & \textit{the sizes of all two-way intersections} \\ \textbf{plus}\quad & \textit{the sizes of all three-way intersections} \\ \textbf{minus}\quad & \textit{the sizes of all four-way intersections} \\ \textbf{plus}\quad & \textit{the sizes of all five-way intersections, etc.} \end{aligned} \tag{14.9.5}$$

The formulas for unions of two and three sets are special cases of this general rule.

This way of expressing Inclusion-Exclusion is easy to understand and nearly as precise as expressing it in mathematical symbols, but we'll need the symbolic version below, so let's work on deciphering it now.

We already have a concise notation for the sum of sizes of the individual sets, namely,

$$\sum_{i=1}^{n} |S_i|.$$

A "two-way intersection" is a set of the form $S_i \cap S_j$ for $i \neq j$. We regard $S_j \cap S_i$ as the same two-way intersection as $S_i \cap S_j$, so we can assume that $i < j$. Now we can express the sum of the sizes of the two-way intersections as

$$\sum_{1 \leq i < j \leq n} |S_i \cap S_j|.$$

Similarly, the sum of the sizes of the three-way intersections is

$$\sum_{1 \leq i < j < k \leq n} |S_i \cap S_j \cap S_k|.$$

These sums have alternating signs in the Inclusion-Exclusion formula, with the sum of the k-way intersections getting the sign $(-1)^{k-1}$. This finally leads to a symbolic version of the rule:

Rule (Inclusion-Exclusion).

$$\begin{aligned} \left|\bigcup_{i=1}^{n} S_i\right| = &\sum_{i=1}^{n} |S_i| \\ &- \sum_{1 \leq i < j \leq n} |S_i \cap S_j| \\ &+ \sum_{1 \leq i < j < k \leq n} |S_i \cap S_j \cap S_k| + \cdots \\ &+ (-1)^{n-1} \left|\bigcap_{i=1}^{n} S_i\right|. \end{aligned} \tag{14.9.6}$$

While it's often handy express the rule in this way as a sum of sums, it is not necessary to group the terms by how many sets are in the intersections. So another way to state the rule is:

Rule (Inclusion-Exclusion-II).

$$\left|\bigcup_{i=1}^{n} S_i\right| = \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} \left|\bigcap_{i \in I} S_i\right| \tag{14.9.7}$$

A proof of these rules using just highschool algebra is given in Problem 14.52.

Computing Euler's Function

We can also use Inclusion-Exclusion to derive the explicit formula for Euler's function claimed in Corollary 8.10.11: if the prime factorization of n is $p_1^{e_1} \cdots p_m^{e_m}$ for distinct primes p_i, then

$$\phi(n) = n \prod_{i=1}^{m} \left(1 - \frac{1}{p_i}\right). \tag{14.9.8}$$

To begin, let S be the set of integers in $[0..n)$ that are *not* relatively prime to n. So $\phi(n) = n - |S|$. Next, let C_a be the set of integers in $[0..n)$ that are divisible by a:

$$C_a ::= \{k \in [0..n) \mid a | k\}.$$

So the integers in S are precisely the integers in $[0..n)$ that are divisible by at least one of the p_i's. Namely,

$$S = \bigcup_{i=1}^{m} C_{p_i}. \tag{14.9.9}$$

We'll be able to find the size of this union using Inclusion-Exclusion because the intersections of the C_{p_i}'s are easy to count. For example, $C_p \cap C_q \cap C_r$ is the set of integers in $[0..n)$ that are divisible by each of p, q and r. But since the p, q, r are distinct primes, being divisible by each of them is the same as being divisible by their product. Now if k is a positive divisor of n, then there are exactly n/k multiples of k in $[0..n)$. So exactly n/pqr of the integers in $[0..n)$ are divisible by all three primes p, q, r. In other words,

$$|C_p \cap C_q \cap C_r| = \frac{n}{pqr}.$$

This reasoning extends to arbitrary intersections of C_p's, namely,

$$\left|\bigcap_{j\in I} C_{p_j}\right| = \frac{n}{\prod_{j\in I} p_j}. \tag{14.9.10}$$

for any nonempty set $I \in [1..m]$. This lets us calculate:

$$\begin{aligned}
|S| &= \left|\bigcup_{i=1}^{m} C_{p_i}\right| && \text{(by 14.9.3)}\\
&= \sum_{\emptyset\neq I\subseteq[1..m]} (-1)^{|I|+1}\left|\bigcap_{i\in I} C_{p_i}\right| && \text{(by Inclusion-Exclusion (14.9.1))}\\
&= \sum_{\emptyset\neq I\subseteq[1..m]} (-1)^{|I|+1}\frac{n}{\prod_{j\in I} p_j} && \text{(by 14.9.4)}\\
&= -n\sum_{\emptyset\neq I\subseteq[1..m]} \frac{1}{\prod_{j\in I}(-p_j)}\\
&= -n\left(\prod_{i=1}^{m}\left(1-\frac{1}{p_i}\right)\right)+n,
\end{aligned} \tag{14.9.11}$$

so

$$\phi(n) = n - |S| = n\prod_{i=1}^{m}\left(1-\frac{1}{p_i}\right),$$

which proves (14.9.8).

Yikes! That was pretty hairy. Are you getting tired of all that nasty algebra? If so, then good news is on the way. In the next section, we will show you how to prove some heavy-duty formulas without using any algebra at all. Just a few words and you are done. No kidding.

[5] ...though not at MIT anymore.

This page titled 14.9: Inclusion-Exclusion is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

14.10: Combinatorial Proofs

Suppose you have n different T-shirts, but only want to keep k. You could equally well select the k shirts you want to keep or select the complementary set of $n-k$ shirts you want to throw out. Thus, the number of ways to select k shirts from among n must be equal to the number of ways to select $n-k$ shirts from among n. Therefore:

$$\binom{n}{k} = \binom{n}{n-k}.$$

This is easy to prove algebraically, since both sides are equal to:

$$\frac{n!}{k!(n-k)!}.$$

But we didn't really have to resort to algebra; we just used counting principles.

Hmmm....

Pascal's Triangle Identity

Bob, famed Math for Computer Science Teaching Assistant, has decided to try out for the US Olympic boxing team. After all, he's watched all of the *Rocky* movies and spent hours in front of a mirror sneering, "Yo, you wanna piece a' *me*?!" Bob figures that n people (including himself) are competing for spots on the team and only k will be selected. As part of maneuvering for a spot on the team, he needs to work out how many different teams are possible. There are two cases to consider:

- Bob *is* selected for the team, and his $k-1$ teammates are selected from among the other $n-1$ competitors. The number of different teams that can be formed in this way is:

$$\binom{n-1}{k-1}.$$

- Bob is *not* selected for the team, and all k team members are selected from among the other $n-1$ competitors. The number of teams that can be formed this way is:

$$\binom{n-1}{k}.$$

All teams of the first type contain Bob, and no team of the second type does; therefore, the two sets of teams are disjoint. Thus, by the Sum Rule, the total number of possible Olympic boxing teams is:

$$\binom{n-1}{k-1} + \binom{n-1}{k}.$$

Ted, equally-famed Teaching Assistant, thinks Bob isn't so tough and so he might as well also try out. He reasons that n people (including himself) are trying out for k spots. Thus, the number of ways to select the team is simply:

$$\binom{n}{k}.$$

Ted and Bob each correctly counted the number of possible boxing teams. Thus, their answers must be equal. So we know:

Lemma 14.10.1 *(Pascal's Triangle Identity).*

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \tag{14.10.1}$$

We proved Pascal's Triangle Identity without any algebra! Instead, we relied purely on counting techniques.

Giving a Combinatorial Proof

A *combinatorial proof* is an argument that establishes an algebraic fact by relying on counting principles. Many such proofs follow the same basic outline:

1. Define a set S.

2. Show that $|S| = n$ by counting one way.
3. Show that $|S| = m$ by counting another way.
4. Conclude that $n = m$.

In the preceding example, S was the set of all possible Olympic boxing teams. Bob computed

$$|S| = \binom{n-1}{k-1} + \binom{n-1}{k}$$

by counting one way, and Ted computed

$$|S| = \binom{n}{k}$$

by counting another way. Equating these two expressions gave Pascal's Identity.

Checking a Combinatorial Proof

Combinatorial proofs are based on counting the same thing in different ways. This is fine when you've become practiced at different counting methods, but when in doubt, you can fall back on bijections and sequence counting to check such proofs.

For example, let's take a closer look at the combinatorial proof of Pascal's Identity (14.10.1). In this case, the set S of things to be counted is the collection of all size-k subsets of integers in the interval $[1..n]$.

Now we' ve already counted S one way, via the Bookkeeper Rule, and found $|S| = \binom{n}{k}$. The other "way" corresponds to defining a bijection between S and the disjoint union of two sets A and B where,

$$\begin{aligned} A &::= \{(1, X) \mid X \subseteq [2, n] \text{ AND } |X| = k-1\} \\ B &::= \{(0, Y) \mid Y \subseteq [2, n] \text{ AND } |Y| = k\} \end{aligned} \tag{14.10.2}$$

Clearly A and B are disjoint since the pairs in the two sets have different first coordinates, so $|A \cup B| = |A| + |B|$. Also,

Misplaced '#'

Misplaced '#'

Now finding a bijection $f : (A \cup B) \to S$ will prove the identity (14.10.1). In particular, we can define

$$f(c) ::= \begin{cases} X \cup \{1\} & \text{if } c = (1, X), \\ Y & \text{if } c = (0, Y). \end{cases}$$

It should be obvious that f is a bijection.

Colorful Combinatorial Proof

The set that gets counted in a combinatorial proof in different ways is usually defined in terms of simple sequences or sets rather than an elaborate story about Teaching Assistants. Here is another colorful example of a combinatorial argument.

Theorem 14.10.2

$$\sum_{r=0}^{n} \binom{n}{r}\binom{2n}{n-r} = \binom{3n}{n}$$

Proof

We give a combinatorial proof. Let S be all n-card hands that can be dealt from a deck containing n different red cards and $2n$ different black cards. First, note that every $3n$-element set has

$$|S| = \binom{3n}{n}$$

n-element subsets.

From another perspective, the number of hands with exactly r red cards is

$$\binom{n}{r}\binom{2n}{n-r}$$

since there are $\binom{n}{r}$ ways to choose the r red cards and $\binom{2n}{n-r}$ ways to choose the $n-r$ black cards. Since the number of red cards can be anywhere from 0 to n, the total number of n-card hands is:

$$|S| = \sum_{r=0}^{n} \binom{n}{r}\binom{2n}{n-r}.$$

Equating these two expressions for $|S|$ proves the theorem. ■

Finding a Combinatorial Proof

Combinatorial proofs are almost magical. Theorem 14.10.2 looks pretty scary, but we proved it without any algebraic manipulations at all. The key to constructing a combinatorial proof is choosing the set S properly, which can be tricky. Generally, the simpler side of the equation should provide some guidance. For example, the right side of Theorem 14.10.2 is $\binom{3n}{n}$, which suggests that it will be helpful to choose S to be all n-element subsets of some $3n$-element set.

This page titled 14.10: Combinatorial Proofs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

15: Generating Functions

Generating Functions are one of the most surprising and useful inventions in Discrete Mathematics. Roughly speaking, generating functions transform problems about *sequences* into problems about *algebra*. This is great because we've got piles of algebraic rules. Thanks to generating functions, we can reduce problems about sequences to checking properties of algebraic expressions. This will allow us to use generating functions to solve all sorts of counting problems.

Several flavors of generating functions such as *ordinary*, *exponential*, and *Dirichlet* come up regularly in combinatorial mathematics. In addition, *Z-transforms*, which are closely related to ordinary generating functions, are important in control theory and signal processing. But ordinary generating functions are enough to illustrate the power of the idea, so we'll stick to them. So from now on *generating function* will mean the ordinary kind, and we will offer a taste of this large subject by showing how generating functions can be used to solve certain kinds of counting problems and how they can be used to find simple formulas for *linear-recursive* functions.

15.1: Infinite Series

15.2: Counting with Generating Functions

15.3: Partial Fractions

15.4: Solving Linear Recurrences

15.5: Formal Power Series

This page titled 15: Generating Functions is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

15.1: Infinite Series

Informally, a generating function, $F(x)$, is an infinite series

$$F(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3 + \cdots. \tag{15.1.1}$$

We use the notation $[x^n]F(x)$ for the coefficient of xn in the generating function $F(x)$. That is, $[x^n]F(x) ::= f_n$.

We can analyze the behavior of any sequence of numbers $f_0, f_1, \ldots f_n \ldots$ by regarding the elements of the sequence as successive coefficients of a generating function. It turns out that properties of complicated sequences that arise from counting, recursive definition, and programming problems are easy to explain by treating them as generating functions.

Generating functions can produce noteworthy insights even when the sequence of coefficients is trivial. For example, let $G(x)$ be the generating function for the infinite sequence of ones $1, 1, \ldots$, namely, the geometric series.

$$G(x) ::= 1 + x + x^2 + \cdots + x^n + \cdots. \tag{15.1.2}$$

We'll use typical generating function reasoning to derive a simple formula for $G(x)$. The approach is actually an easy version of the perturbation method of Section 13.1.2. Specifically,

$$\begin{array}{rl} G(x) = & 1 + x + x^2 + x^3 + \cdots + x^n + \cdots \\ -xG(x) = & \quad -x - x^2 - x^3 - \cdots - x^n - \cdots \\ \hline G(x) - xG(x) = & 1. \end{array} \tag{15.1.3}$$

Solving for $G(x)$ gives

$$\frac{1}{1-x} = G(x) ::= \sum_{n=0}^{\infty} x^n. \tag{15.1.4}$$

In other words,

$$[x^n]\left(\frac{1}{1-x}\right) = 1$$

Continuing with this approach yields a nice formula for

$$N(x) ::= 1 + 2x + 3x^2 + \cdots + (n+1)x^n + \cdots. \tag{15.1.5}$$

Specifically,

$$\begin{array}{rl} N(x) = & 1 + 2x + 3x^2 + 4x^3 + \cdots + (n+1)x^n + \cdots \\ -xN(x) = & \quad -x - 2x^2 - 3x^3 - \cdots - nx^n - \cdots \\ \hline N(x) - xN(x) = & 1 + x + x^2 + x^3 + \cdots + x^n + \cdots \\ = & G(x). \end{array} \tag{15.1.6}$$

Solving for $N(x)$ gives

$$\frac{1}{(1-x)^2} = \frac{G(x)}{1-x} = N(x) ::= \sum_{n=0}^{\infty} (n+1)x^n. \tag{15.1.7}$$

On other words,

$$[x^n]\left(\frac{1}{(1-x)^2}\right) = n + 1.$$

Never Mind Convergence

Equations (15.1.4) and (15.1.7) hold numerically only when $|x| < 1$, because both generating function series diverge when $|x| \geq 1$. But in the context of generating functions, we regard infinite series as formal algebraic objects. Equations such as (15.1.4) and (15.1.7) define symbolic identities that hold for purely algebraic reasons. In fact, good use can be made of generating functions determined by infinite series that don't converge *anywhere* (besides $x = 0$). We'll explain this further in Section 15.5 at the end of this chapter, but for now, take it on faith that you don't need to worry about convergence.

This page titled 15.1: Infinite Series is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

15.2: Counting with Generating Functions

Generating functions are particularly useful for representing and counting the number of ways to select n things. For example, suppose there are two flavors of donuts, chocolate and plain. Let dn be the number of ways to select n chocolate or plain flavored donuts. $d_n = n+1$, because there are $n+1$ such donut selections—all chocolate, 1 plain and $n-1$ chocolate, 2 plain and $n-2$ chocolate,... , all plain. We define a generating function, $D(x)$, for counting these donut selections by letting the coefficient of x^n be d_n. This gives us equation (15.1.5)

$$D(x) = \frac{1}{(1-x)^2}. \tag{15.2.1}$$

Apples and Bananas too

More generally, suppose we have two kinds of things—say, apples and bananas— and some constraints on how many of each may be selected. Say there are a_n ways to select n apples and b_n ways to select n bananas. The generating function for counting apples would be

$$A(x) ::= \sum_{n=0}^{\infty} a_n x^n,$$

and for bananas would be

$$B(x) ::= \sum_{n=0}^{\infty} b_n x^n,$$

Now suppose apples come in baskets of 6, so there is no way to select 1 to 5 apples, one way to select 6 apples, no way to select 7, etc. In other words,

$$a_n ::= \begin{cases} 1 & \text{if } n \text{ is a multiple of 6,} \\ 0 & \text{otherwise.} \end{cases}$$

In this case we would have

$$\begin{aligned} A(x) &= 1 + x^6 + x^{12} + \cdots + x^{6n} + \cdots \\ &= 1 + y + y^2 + \cdots + y^n + \cdots \quad \text{where } y = x^6. \\ &= \frac{1}{1-y} = \frac{1}{1-x^6}. \end{aligned} \tag{15.2.2}$$

Let's also suppose there are two kinds of bananas—red and yellow. Now, $b_n = n+1$ by the same reasoning used to count selections of n chocolate and plain donuts, and by (15.2.1) we have

$$B(x) = \frac{1}{(1-x)^2}.$$

So how many ways are there to select a mix of n apples and bananas? First, we decide how many apples to select. This can be any number k from 0 to n. We can then select these apples in a_k ways, by definition. This leaves $n-k$ bananas to be selected, which by definition can be done in b_{n-k} ways. So the total number of ways to select k apples and $n-k$ bananas is $a_k b_{n-k}$. This means that the total number of ways to select some size n mix of apples and bananas is

$$a_0 b_n + a_1 b_{n-1} + a_2 b_{n-2} + \cdots + a_n b_0. \tag{15.2.3}$$

Products of Generating Functions

Now here's the cool connection between counting and generating functions: expression (15.2.3) is equal to the coefficient of xn in the product $A(x)B(x)$.

In other words, we're claiming that

$$[x^n](A(x) \cdot B(x)) = a_0 b_n + a_1 b_{n-1} + a_2 b_{n-2} + \cdots + a_n b_0. \tag{15.2.4}$$

Rule (Product).

To explain the generating function Product Rule, we can think about evaluating the product $A(x) \cdot B(x)$ by using a table to identify all the cross-terms from the product of the sums:

	b_0x^0	b_1x^1	b_2x^2	b_3x^3	...
a_0x^0	$a_0b_0x^0$	$a_0b_1x^1$	$a_0b_2x^2$	$a_0b_3x^3$	...
a_1x^1	$a_1b_0x^1$	$a_1b_1x^2$	$a_1b_2x^3$	...	
a_2x^2	$a_2b_0x^2$	$a_2b_1x^3$	...		
a_3x^3	$a_3b_0x^3$	...			
$\vdots$	...				

In this layout, all the terms involving the same power of x lie on a 45-degree sloped diagonal. So, the index-n diagonal contains all the x^n-terms, and the coefficient of x^n in the product $A(x) \cdot B(x)$ is the sum of all the coefficients of the terms on this diagonal, namely, (15.2.3). The sequence of coefficients of the product $A(x) \cdot B(x)$ is called the convolution of the sequences $(a_0, a_1, a_2, ...)$ and $(b_0, b_1, b_2, ...)$. In addition to their algebraic role, convolutions of sequences play a prominent role in signal processing and control theory.

This Product Rule provides the algebraic justification for the fact that a geometric series equals $1/(1-x)$ regardless of convergence. Specifically, the constant 1 describes the generating function

$$1 = 1 + 0x + 0x^2 + \cdots + 0x^n + \cdots.$$

Likewise, the expression $1 - x$ describes the generating function

$$1 - x = 1 + (-1)x + 0x^2 + \cdots + 0x^n + \cdots.$$

So for the series $G(x)$ whose coefficients are all equal to 1, the Product Rule implies in a purely formal way that

$$(1-x) \cdot G(x) = 1 + 0x + 0x^2 + \cdots + 0x^n + \cdots = 1.$$

In other words, under the Product Rule, the geometric series $G(x)$ is the multiplicative inverse, $1/(1-x)$, of $1 - x$.

Similar reasoning justifies multiplying a generating function by a constant term by term. That is, a special case of the Product Rule is the

Rule (Constant Factor). *For any constant, c, and generating function, $F(x)$,*

$$[x^n](c \cdot F(x)) = c \cdot [x^n]F(x). \tag{15.2.5}$$

The Convolution Rule

We can summarize the discussion above with the

Rule (Convolution). *Let $A(x)$ be the generating function for selecting items from a set A, and let $B(x)$ be the generating function for selecting items from a set B disjoint from A. The generating function for selecting items from the union $A \cup B$ is the product $A(x) \cdot B(x)$.*

The Rule depends on a precise definition of what "selecting items from the union $A \cup B$" means. Informally, the idea is that the restrictions on the selection of items from sets A and B carry over to selecting items from $A \cup B$[1].

Counting Donuts with the Convolution Rule

We can use the Convolution Rule to derive in another way the generating function $D(x)$ for the number of ways to select chocolate and plain donuts given in (15.6). To begin, there is only one way to select exactly n chocolate donuts. That means every coefficient of the generating function for selecting n chocolate donuts equals one. So the generating function for chocolate donut selections is $1/(1-x)$; likewise for the generating function for selecting only plain donuts. Now by the Convolution Rule, the generating function for the number of ways to select n donuts when both chocolate and plain flavors are available is

$$D(x) = \frac{1}{1-x} \cdot \frac{1}{1-x} = \frac{1}{(1-x)^2}.$$

So we have derived (15.2.1) without appeal to (15.1.5).

Our application of the Convolution Rule for two flavors carries right over to the general case of k flavors; the generating function for selections of donuts when k flavors are available is $1/(1-x)^k$. We already derived the formula for the number of ways to select a n donuts when k flavors are available, namely, $\binom{n+(k-1)}{n}$ from Corollary 14.5.3. So we have

$$[x^n]\left(\frac{1}{(1-x)^k}\right) = \binom{n+(k-1)}{n}. \tag{15.2.6}$$

Extracting Coefficients from Maclaurin's Theorem

We've used a donut-counting argument to derive the coefficients of $1/(1-x)^k$, but it's instructive to derive this coefficient algebraically, which we can do using Maclaurin's Theorem:

Theorem 15.2.1

(Maclaurin's Theorem).

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \cdots + \frac{f^{(n)}(0)}{n!}x^n + \cdots.$$

This theorem says that the nth coefficient of $1/(1-x)^k$ is equal to its nth derivative evaluated at 0 and divided by $n!$. Computing the nth derivative turns out not to be very difficult

$$\frac{d^n}{d^n x}\frac{1}{(1-x)^k} = k(k+1)\cdot(k+n-1)(1-x)^{-(k+n)}$$

(see Problem 15.5), so

$$\begin{aligned}[x^n]\left(\frac{1}{(1-x)^k}\right) &= \left(\frac{d^n}{d^n x}\frac{1}{(1-x)^k}\right)(0)\frac{1}{n!}\\ &= \frac{k(k+1)\cdots(k+n-1)(1-0)^{-(k+n)}}{n!}\\ &= \binom{n+(k-1)}{n}.\end{aligned} \tag{15.2.7}$$

In other words, instead of using the donut-counting formula (15.2.6) to find the coefficients of x^n, we could have used this algebraic argument and the Convolution Rule to derive the donut-counting formula.

The Binomial Theorem from the Convolution Rule

The Convolution Rule also provides a new perspective on the Binomial Theorem 14.6.4. Here's how: first, work with the single-element set $\{a_1\}$. The generating function for the number of ways to select n different elements from this set is simply $1+x$: we have 1 way to select zero elements, 1 way to select the one element, and 0 ways to select more than one element. Similarly, the number of ways to select n elements from any single-element set $\{a_i\}$ has the same generating function $1+x$. Now by the Convolution Rule, the generating function for choosing a subset of n elements from the set $\{a_1, a_2, \ldots, a_m\}$ is the product, $(1+x)^m$, of the generating functions for selecting from each of the m one-element sets. Since we know that the number of ways to select n elements from a set of size m is $\binom{m}{n}$, we conclude that that

$$[x^n](1+x)^m = \binom{m}{n},$$

which is a restatement of the Binomial Theorem 14.6.4. Thus, we have proved the Binomial Theorem without having to analyze the expansion of the expression $(1+x)^m$ into a sum of products.

These examples of counting donuts and deriving the binomial coefficients illustrate where generating functions get their power:

> Generating functions can allow counting problems to be solved by algebraic manipulation, and conversely, they can allow algebraic identities to be derived by counting techniques.

Absurd Counting Problem

So far everything we've done with generating functions we could have done another way. But here is an absurd counting problem —really over the top! In how many ways can we fill a bag with n fruits subject to the following constraints?

- The number of apples must be even.
- The number of bananas must be a multiple of 5.
- There can be at most four oranges.
- There can be at most one pear.

For example, there are 7 ways to form a bag with 6 fruits:

Apples	6	4	4	2	2	0	0
Bananas	0	0	0	0	0	5	5
Oranges	0	2	1	4	3	1	0
Pears	0	0	1	0	1	0	1

These constraints are so complicated that getting a nice answer may seem impossible. But let's see what generating functions reveal.

First, we'll construct a generating function for choosing apples. We can choose a set of 0 apples in one way, a set of 1 apple in zero ways (since the number of apples must be even), a set of 2 apples in one way, a set of 3 apples in zero ways, and so forth. So, we have:

$$A(x) = 1 + x^2 + x^4 + x^6 + \cdots = \frac{1}{1-x^2}$$

Similarly, the generating function for choosing bananas is:

$$B(x) = 1 + x^5 + x^{10} + x^{15} + \cdots = \frac{1}{1-x^5}$$

Now, we can choose a set of 0 oranges in one way, a set of 1 orange in one way, and so on. However, we cannot choose more than four oranges, so we have the generating function:

$$O(x) = 1 + x + x^2 + x^3 + x^4 = \frac{1-x^5}{1-x}.$$

Here the right hand expression is simply the formula (13.2) for a finite geometric sum. Finally, we can choose only zero or one pear, so we have:

$$P(x) = 1 + x$$

The Convolution Rule says that the generating function for choosing from among all four kinds of fruit is:

$$\begin{aligned} A(x)B(x)O(x)P(x) &= \frac{1}{1-x^2}\frac{1}{1-x^5}\frac{1-x^5}{1-x}(1+x) \\ &= \frac{1}{(1-x)^2} \\ &= 1 + 2x + 3x^2 + 4x^3 + \cdots \end{aligned}$$

Almost everything cancels! We're left with $1/(1-x)^2$, which we found a power series for earlier: the coefficient of x^n is simply $n+1$. Thus, the number of ways to form a bag of n fruits is just $n+1$. This is consistent with the example we worked out, since there were 7 different fruit bags containing 6 fruits. *Amazing*!

[1]Formally, the Convolution Rule applies when there is a bijection between n-element selections from $A \cup B$ and ordered pairs of selections from the sets A and B containing a total of n elements. We think the informal statement is clear enough.

This page titled 15.2: Counting with Generating Functions is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

15.3: Partial Fractions

We got a simple solution to the seemingly impossible counting problem of Section 15.2.6 because its generating function simplified to the expression $1/(1-x)^2$, whose power series coefficients we already knew. You've probably guessed that this problem was contrived so the answer would work out neatly. But other problems may not be so neat. To solve more general problems using generating functions, we need ways to find power series coefficients for generating functions given as formulas. Maclaurin's Theorem 15.2.1 is a very general method for finding coefficients, but it only applies when formulas for repeated derivatives can be found, which isn't often. However, there is an automatic way to find the power series coefficients for any formula that is a quotient of polynomials, namely, the method of partial fractions from elementary calculus.

The partial fraction method is based on the fact that quotients of polynomials can be expressed as sums of terms whose power series coefficients have nice formulas. For example when the denominator polynomial has distinct nonzero roots, the method rests on

Lemma 15.3.1. *Let $p(x)$ be a polynomial of degree less than n and let $\alpha_1, \ldots, \alpha_n$ be distinct, nonzero numbers. Then there are constants $c_1, \ldots, c_n$ such that*

$$\frac{p(x)}{(1-\alpha_1 x)(1-\alpha_2 x)\cdots(1-\alpha_n x)} = \frac{c_1}{1-\alpha_1 x} + \frac{c_2}{1-\alpha_2 x} + \cdots + \frac{c_n}{1-\alpha_n x}.$$

Let's illustrate the use of Lemma 15.3.1 by finding the power series coefficients for the function

$$R(x) ::= \frac{x}{1-x-x^2}.$$

We can use the quadratic formula to find the roots r_1, r_2 of the denominator, $1-x-x^2$.

$$r_1 = \frac{-1-\sqrt{5}}{2}, r_2 = \frac{-1+\sqrt{5}}{2}.$$

So

$$1-x-x^2 = (x-r_1)(x-r_2) = r_1 r_2 (1-x/r_1)(1-x/r_2).$$

With a little algebra, we find that

$$R(x) = \frac{x}{(1-\alpha_1 x)(1-\alpha_2 x)}$$

where

$$\begin{aligned} \alpha_1 &= \frac{1+\sqrt{5}}{2} \\ \alpha_2 &= \frac{1-\sqrt{5}}{2}. \end{aligned} \tag{15.3.1}$$

Next we find c_1 and c_2 which satisfy:

$$\frac{x}{(1-\alpha_1 x)(1-\alpha_2 x)} = \frac{c_1}{1-\alpha_1 x} + \frac{c_2}{1-\alpha_2 x} \tag{15.3.2}$$

In general, we can do this by plugging in a couple of values for x to generate two linear equations in c_1 and c_2 and then solve the equations for c_1 and c_2. A simpler approach in this case comes from multiplying both sides of (15.3.2) by the left hand denominator to get

$$x = c_1(1-\alpha_2 x) + c_2(1-\alpha_1 x).$$

Now letting $x = 1/\alpha_2$ we obtain

$$c_2 = \frac{1/\alpha_2}{1-\alpha_1/\alpha_2} = \frac{1}{\alpha_2-\alpha_1} = -\frac{1}{\sqrt{5}}.$$

and similarly, letting $x = 1/\alpha_1$ we obtain

$$c_1 = \frac{1}{\sqrt{5}}.$$

Plugging these values for c_1, c_2 into equation (15.3.2) finally gives the partial fraction expansion

$$R(x) = \frac{x}{1-x-x^2} = \frac{1}{\sqrt{5}}\left(\frac{1}{1-\alpha_1 x} - \frac{1}{1-\alpha_2 x}\right)$$

Each term in the partial fractions expansion has a simple power series given by the geometric sum formula:

$$\begin{aligned} \frac{1}{1-\alpha_1 x} &= 1 + \alpha_1 x + \alpha_1^2 x^2 + \cdots \\ \frac{1}{1-\alpha_2 x} &= 1 + \alpha_2 x + \alpha_2^2 x^2 + \cdots \end{aligned} \tag{15.3.3}$$

Substituting in these series gives a power series for the generating function:

$$R(x) = \frac{1}{\sqrt{5}}((1 + \alpha_1 x + \alpha_1^2 x^2 + \cdots) - (1 + \alpha_2 x + \alpha_2^2 x^2 + \cdots)),$$

so

$$\begin{aligned} [x^n]R(x) &= \frac{\alpha_1^n - \alpha_2^n}{\sqrt{5}} \\ &= \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right) \end{aligned} \tag{15.3.4}$$

Partial Fractions with Repeated Roots

Lemma 15.3.1 generalizes to the case when the denominator polynomial has a repeated nonzero root with multiplicity m by expanding the quotient into a sum a terms of the form

$$\frac{c}{(1-\alpha x)^k}$$

where α is the reciprocal of the root and $k \leq m$. A formula for the coefficients of such a term follows from the donut formula (15.2.5).

$$[x^n]\left(\frac{c}{(1-\alpha x)^k}\right) = c\alpha^n \binom{n+(k-1)}{n}. \tag{15.3.5}$$

When $\alpha = 1$, this follows from the donut formula (15.2.5) and termwise multiplication by the constant c. The case for arbitrary α follows by substituting αx for x in the power series; this changes x^n into $(\alpha x)^n$ and so has the effect of multiplying the coefficient of x^n by α^n.[2]

[2]In other words,

$$[x^n]F(\alpha x) = \alpha^n \cdot [x^n]F(x).$$

This page titled 15.3: Partial Fractions is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

15.4: Solving Linear Recurrences

Generating Function for the Fibonacci Numbers

The Fibonacci numbers $f_0, f_1, \ldots, f_n, \ldots$ are defined recursively as follows:

$$\begin{aligned} f_0 &::= 0 \\ f_1 &::= 1 \\ f_n &=::= f_{n-1} + f_{n-2} \qquad (\text{for } n \geq 2). \end{aligned} \tag{15.4.1}$$

Generating functions will now allow us to derive an astonishing closed formula for f_n.

Let $F(x)$ be the generating function for the sequence of Fibonacci numbers, that is,

$$F(x) ::= f_0 + f_1 x + f_2 x^2 + \cdots + f_n x^n + \cdots.$$

Reasoning as we did at the start of this chapter to derive the formula for a geometric series, we have

Illegal pream-token (=)

so

$$F(x) = \frac{x}{1 - x - x^2}.$$

But $F(x)$ is the same as the function we used to illustrate the partial fraction method for finding coefficients in Section 15.3. So by equation (15.3.2), we arrive at what is called *Binet's formula*:

$$f_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right) \tag{15.4.2}$$

Binet's formula for Fibonacci numbers is astonishing and maybe scary. It's not even obvious that the expression on the right hand side (15.4.2) is an integer. But the formula is very useful. For example, it provides—via the repeated squaring method—a much more efficient way to compute Fibonacci numbers than crunching through the recurrence. It also make explicit the exponential growth of these numbers.

The Towers of Hanoi

According to legend, there is a temple in Hanoi with three posts and 64 gold disks of different sizes. Each disk has a hole through the center so that it fits on a post. In the misty past, all the disks were on the first post, with the largest on the bottom and the smallest on top, as shown in Figure 15.1.

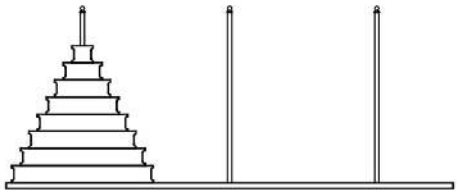


Figure 15.1 The initial configuration of the disks in the Towers of Hanoi problem.

Monks in the temple have labored through the years since to move all the disks to one of the other two posts according to the following rules:

- The only permitted action is removing the top disk from one post and dropping it onto another post.
- A larger disk can never lie above a smaller disk on any post.

So, for example, picking up the whole stack of disks at once and dropping them on another post is illegal. That's good, because the legend says that when the monks complete the puzzle, the world will end!

To clarify the problem, suppose there were only 3 gold disks instead of 64. Then the puzzle could be solved in 7 steps as shown in Figure 15.2.

The questions we must answer are, “Given sufficient time, can the monks succeed?” If so, “How long until the world ends?” And, most importantly, “Will this happen before the final exam?”

A Recursive Solution

The Towers of Hanoi problem can be solved recursively. As we describe the procedure, we’ll also analyze the minimum number, t_n, of steps required to solve the n-disk problem. For example, some experimentation shows that $t_1 = 1$ and $t_2 = 3$. The procedure illustrated above uses 7 steps, which shows that t_3 is at most 7.

The recursive solution has three stages, which are described below and illustrated in Figure 15.3. For clarity, the largest disk is shaded in the figures.

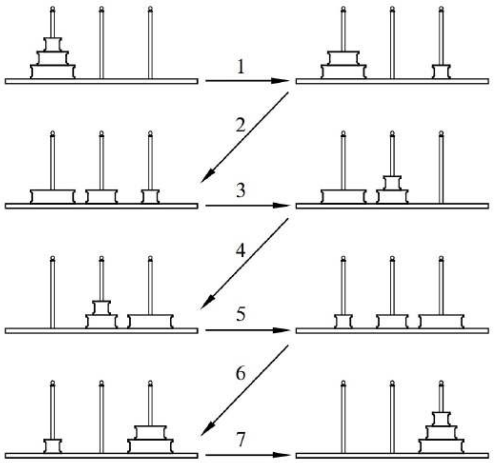


Figure 15.2 The 7-step solution to the Towers of Hanoi problem when there are $n = 3$ disks.

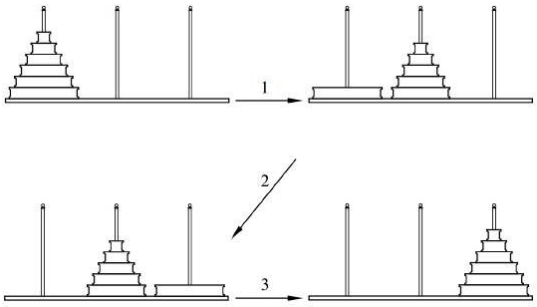


Figure 15.3 A recursive solution to the Towers of Hanoi problem.

Stage 1. Move the top $n-1$ disks from the first post to the second using the solution for $n-1$ disks. This can be done in t_{n-1} steps.

Stage 2. Move the largest disk from the first post to the third post. This takes just 1 step.

Stage 3. Move the $n-1$ disks from the second post to the third post, again using the solution for $n-1$ disks. This can also be done in t_{n-1} steps.

This algorithm shows that t_n, the minimum number of steps required to move n disks to a different post, is at most $t_{n-1} + 1 + t_{n-1} = 2t_{n-1} + 1$. We can use this fact to upper bound the number of operations required to move towers of various heights:

$$\begin{aligned} t_3 &\leq 2 \cdot t_2 + 1 = 7 \\ t_4 &\leq 2 \cdot t_3 + 1 \leq 15 \end{aligned} \tag{15.4.3}$$

Continuing in this way, we could eventually compute an upper bound on t_{64}, the number of steps required to move 64 disks. So this algorithm answers our first question: given sufficient time, the monks can finish their task and end the world. This is a shame. After all that effort, they’d probably want to smack a few high-fives and go out for burgers and ice cream, but nope—world’s over.

Finding a Recurrence

We cannot yet compute the exact number of steps that the monks need to move the 64 disks, only an upper bound. Perhaps, having pondered the problem since the beginning of time, the monks have devised a better algorithm

Lucky for us, there is no better algorithm. Here's why: at some step, the monks must move the largest disk from the first post to a different post. For this to happen, the $n-1$ smaller disks must all be stacked out of the way on the only remaining post. Arranging the $n-1$ smaller disks this way requires at least t_{n-1} moves. After the largest disk is moved, at least another t_{n-1} moves are required to pile the $n-1$ smaller disks on top

This argument shows that the number of steps required is at least $2t_{n-1}+1$. Since we gave an algorithm using exactly that number of steps, we can now write an expression for t_n, the number of moves required to complete the Towers of Hanoi problem with n disks:

$$\begin{aligned} t_0 &= 0 \\ t_n &= 2t_{n-1}+1 \qquad (\text{for } n \geq 1). \end{aligned} \tag{15.4.4}$$

Solving the Recurrence

We can now find a formula for t_n using generating functions. Let $T(x)$ be the generating function for the t_n's, that is,

$$T(x) ::= t_0 + t_1 x + t_2 x^2 + \cdots + t_n x^n + \cdots.$$

Reasoning as we did for the Fibonacci recurrence, we have

Illegal pream-token (=)

so

$$T(x)(1-2x) = \frac{1}{1-x} - 1 = \frac{x}{1-x},$$

and

$$T(x) = \frac{x}{(1-2x)(1-x)}.$$

Using partial fractions,

$$\frac{x}{(1-2x)(1-x)} = \frac{c_1}{1-2x} + \frac{c_2}{1-x}$$

for some constant c_1, c_2. Now multiplying both sides by the left hand denominator gives

$$x = c_1(1-x) + c_2(1-2x).$$

Substituting $1/2$ for x yields $c_1 = 1$ and substituting 1 for x yields $c_2 = -1$, which gives

$$T(x) = \frac{1}{1-2x} - \frac{1}{1-x}.$$

Finally we can read off the simple formula for the numbers of steps needed to move a stack of n disks:

$$t_n = [x^n]T(x) = [x^n]\left(\frac{1}{1-2x}\right) - [x^n]\left(\frac{1}{1-x}\right) = 2^n - 1.$$

Solving General Linear Recurrences

An equation of the form

$$f(n) = c_1 f(n-1) + c_2 f(n-2) + \cdots + c_d f(n-d) + h(n) \tag{15.4.5}$$

for constants $c_i \in \mathbb{C}$ is called a *degree d linear recurrence* with inhomogeneous term $h(n)$.

The methods above can be used to solve linear recurrences with a large class of inhomogeneous terms. In particular, when the inhomogeneous term itself has a generating function that can be expressed as a quotient of polynomials, the approach used above to derive generating functions for the Fibonacci and Tower of Hanoi examples carries over to yield a quotient of polynomials that

defines the generating function $f(0) + f(1)x + f(2)x^2 + \cdots$. Then partial fractions can be used to find a formula for $f(n)$ that is a linear combination of terms of the form $n^k \alpha^n$ where k is a nonnegative integer $\leq d$ and α is the reciprocal of a root of the denominator polynomial. For example, see Problems 15.15, 15.16, 15.20, and 15.21.

This page titled 15.4: Solving Linear Recurrences is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

15.5: Formal Power Series

Divergent Generating Functions

Let $F(x)$ be the generating function for $n!$, that is,

$$F(x) ::= 1 + 1x + 2x^2 + \cdots + n!x^n + \cdots.$$

Because $x^n = o(n!)$ for all $x \neq 0$, this generating function converges only at $x = 0$.[3]

Next, let $H(x)$ be the generating function for $n \cdot n!$, that is,

$$H(x) ::= 0 + 1x + 4x^2 + \cdots + n \cdot n!x^n \ + \cdots.$$

Again, $H(x)$ converges only for $x = 0$, so $H(x)$ and $F(x)$ describe the same, trivial, partial function on the reals.

On the other hand, $F(x)$ and $H(x)$/ have different coefficients for all powers of x greater than 1, and we can usefully distinguish them as formal, symbolic objects.

To illustrate this, note than by subtracting 1 from $F(x)$ and then dividing each of the remaining terms by x, we get a series where the coefficient if x^n is $(n+1)!$. That is

$$[x^n]\left(\frac{F(x)-1}{x}\right) = (n+1)!. \tag{15.5.1}$$

Now a little further formal reasoning about $F(x)$ and $H(x)$ will allow us to deduce the following identity for $n!$: [4]

$$n! = 1 + \sum_{i=1}^{n}(i-1)\cdot(i-1)! \tag{15.5.2}$$

To prove this identity, note that from (15.5.1), we have

$$[x^n]H(x) ::= n \cdot n! = (n+1)! - n! = [x^n]\left(\frac{F(x)-1}{x}\right) - [x^n]F(x).$$

In other words,

$$H(x) = \frac{F(x)-1}{x} - F(x), \tag{15.5.3}$$

Solving (15.5.3) for $F(x)$, we get

$$F(x) = \frac{xH(x)+1}{1-x}. \tag{15.5.4}$$

But $[x^n](xH(x)+1)$ is $(n-1)\cdot(n-1)!$ for $n \geq 1$ and is 1 for $n = 0$, so by the convolution formula,

$$[x^n]\left(\frac{xH(x)+1}{1-x}\right) = 1 + \sum_{i=1}^{n}(i-1)\cdot(i-1)!.$$

The identity (15.5.2) now follows immediately from (15.5.4).

The Ring of Power Series

So why don't we have to worry about series whose radius of convergence is zero, and how do we justify the kind of manipulation in the previous section to derive the formula (15.5.4)? The answer comes from thinking abstractly about infinite sequences of numbers and operations that can be performed on them.

For example, one basic operation combining two infinite sequences is adding them coordinatewise. That is, if we let

$$\begin{aligned} G &::= (g_0, g_1, g_2, \ldots), \\ H &::= (h_0, h_1, h_2, \ldots), \end{aligned} \tag{15.5.5}$$

then we can define the sequence sum, $\oplus$, by the rule:

$$G \oplus H ::= (g_0 + h_0, g_1 + h_1, \ldots, g_n + h_n, \ldots).$$

Another basic operation is sequence multiplication, $\otimes$, defined by the convolution rule (*not* coordinatewise):

$$G \otimes H ::= \left(g_0 + h_0, g_0 h_1 + g_1 h_0, \ldots, \sum_{i=0}^{n} g_i h_{n-i}, \ldots\right).$$

These operations on infinite sequences have lots of nice properties. For example, it's easy to check that sequence addition and multiplication are commutative:

$$\begin{aligned} G \oplus H &= H \oplus G, \\ G \otimes H &= H \otimes G. \end{aligned} \tag{15.5.6}$$

If we let

$$\begin{aligned} Z &::= (0, 0, 0, \ldots), \\ I &::= (1, 0, 0, 0, \ldots, 0, \ldots), \end{aligned} \tag{15.5.7}$$

then it's equally easy to check that Z acts like a zero for sequences and I acts like the number one:

$$\begin{aligned} Z \oplus G &= G, \\ Z \otimes G &= Z, \\ I \otimes G &= G. \end{aligned} \tag{15.5.8}$$

Now if we define

$$-G ::= (-g_0, -g_1, -g_2, \ldots)$$

then

$$G \oplus (-G) = Z.$$

In fact, the operations $\oplus$ and $\otimes$ satisfy all the *commutative ring* axioms described in Section 8.7.1. The set of infinite sequences of numbers together with these operations is called the *ring of formal power series* over these numbers.[5]

A sequence H is the *reciprocal* of a sequence G when

$$G \otimes H = I.$$

A reciprocal of G is also called a *multiplicative inverse* or simply an "inverse" of G. The ring axioms imply that if there is a reciprocal, it is unique (see Problem 8.32), so the suggestive notation $1/G$ can be used unambiguously to denote this reciprocal, if it exists. For example, letting

$$\begin{aligned} J &::= (1, -1, 0, 0, \ldots 0, \ldots) \\ K &::= (1, 1, 1, 1, \ldots, 1, \ldots), \end{aligned} \tag{15.5.9}$$

the definition of $\otimes$ implies that $J \otimes K = I$, and so $K = 1/J$ and $J = 1/Ke$.

In the ring of formal power series, equation (15.5.8) implies that the zero sequence Z has no inverse, so $1/Z$ is undefined—just as the expression 1/0 is undefined over the real numbers or the ring $\mathbb{Z}_n$ of Section 8.7.1. It's not hard to verify that a series has an inverse iff its initial element is nonzero (see Problem 15.26).

Now we can explain the proper way to understand a generating function definition

$$G(x) ::= \sum_{n=0}^{\infty} g_n x^n.$$

It simply means that $G(x)$ really refers to its infinite sequence of coefficients $(g_0, g_1, \ldots)$ in the ring of formal power series. The simple expression, x, can be understood as referring to the sequence

$$X ::= (0, 1, 0, 0, \ldots, 0, \ldots).$$

Likewise, $1 - x$ abbreviates the sequence J above, and the familiar equation

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \cdots \tag{15.5.10}$$

can be understood as a way of restating the assertion that K is $1/J$. In other words, the powers of the variable x just serve as a place holders—and as reminders of the definition of convolution. The equation (15.5.10) has nothing to do with the values of x or the convergence of the series. Rather, it is stating a property that holds in the ring of formal power series. The reasoning about the divergent series in the previous section is completely justified as properties of formal power series.

[3]This section is based on an example from "Use of everywhere divergent generating function," math**overflow**, response 8,147 by Aaron Meyerowitz, Nov. 12, 2010.

[4]A combinatorial proof of (15.5.2) is given in Problem 14.64.

[5]The elements in the sequences may be the real numbers, complex numbers, or, more generally, may be the elements from any given commutative ring.

This page titled 15.5: Formal Power Series is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

SECTION OVERVIEW

4: Probability

16: Events and Probability Spaces

16.1: Let's Make a Deal
16.2: The Four Step Method
16.3: Strange Dice
16.4: The Birthday Principle
16.5: Set Theory and Probability

17: Conditional Probability

17.1: Monty Hall Confusion
17.2: Definition and Notation
17.3: The Four-Step Method for Conditional Probability
17.4: Why Tree Diagrams Work
17.5: The Law of Total Probability
17.6: Simpson’s Paradox
17.7: Independence
17.8: Mutual Independence

18: Random Variables

18.1: Random Variable Examples
18.2: Independence
18.3: Distribution Functions
18.4: Great Expectations
18.5: Linearity of Expectation

19: Deviation from the Mean

19.1: Markov’s Theorem
19.2: Chebyshev’s Theorem
19.3: Properties of Variance
19.4: Estimation by Random Sampling
19.5: Confidence versus Probability
19.6: Sums of Random Variables
19.7: Really Great Expectations

20: Random Walks

20.1: Gambler’s Ruin
20.2: Random Walks on Graphs

This page titled 4: Probability is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

16: Events and Probability Spaces

16.1: Let's Make a Deal

16.2: The Four Step Method

16.3: Strange Dice

16.4: The Birthday Principle

16.5: Set Theory and Probability

This page titled 16: Events and Probability Spaces is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

16.1: Let's Make a Deal

In the September 9, 1990 issue of *Parade* magazine, columnist Marilyn vos Savant responded to this letter:

> *Suppose you're on a game show, and you're given the choice of three doors. Behind one door is a car, behind the others, goats. You pick a door, say number 1, and the host, who knows what's behind the doors, opens another door, say number 3, which has a goat. He says to you, "Do you want to pick door number 2?" Is it to your advantage to switch your choice of doors?*
>
> **Craig. F. Whitaker**
> **Columbia, MD**

The letter describes a situation like one faced by contestants in the 1970's game show *Let's Make a Deal*, hosted by Monty Hall and Carol Merrill. Marilyn replied that the contestant should indeed switch. She explained that if the car was behind either of the two unpicked doors—which is twice as likely as the the car being behind the picked door—the contestant wins by switching. But she soon received a torrent of letters, many from mathematicians, telling her that she was wrong. The problem became known as the *Monty Hall Problem* and it generated thousands of hours of heated debate.

This incident highlights a fact about probability: the subject uncovers lots of examples where ordinary intuition leads to completely wrong conclusions. So until you've studied probabilities enough to have refined your intuition, a way to avoid errors is to fall back on a rigorous, systematic approach such as the Four Step Method that we will describe shortly. First, let's make sure we really understand the setup for this problem. This is always a good thing to do when you are dealing with probability.

Clarifying the Problem

Craig's original letter to Marilyn vos Savant is a bit vague, so we must make some assumptions in order to have any hope of modeling the game formally. For example, we will assume that:

1. The car is equally likely to be hidden behind each of the three doors.
2. The player is equally likely to pick each of the three doors, regardless of the car's location.
3. After the player picks a door, the host *must* open a different door with a goat behind it and offer the player the choice of staying with the original door or switching.
4. If the host has a choice of which door to open, then he is equally likely to select each of them.

In making these assumptions, we're reading a lot into Craig Whitaker's letter. There are other plausible interpretations that lead to different answers. But let's accept these assumptions for now and address the question, "What is the probability that a player who switches wins the car?"

This page titled 16.1: Let's Make a Deal is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

16.2: The Four Step Method

Every probability problem involves some sort of randomized experiment, process, or game. And each such problem involves two distinct challenges:

1. How do we model the situation mathematically?
2. How do we solve the resulting mathematical problem?

In this section, we introduce a four step approach to questions of the form, "What is the probability that. . . ?" In this approach, we build a probabilistic model step by step, formalizing the original question in terms of that model. Remarkably, this structured approach provides simple solutions to many famously confusing problems. For example, as you'll see, the four step method cuts through the confusion surrounding the Monty Hall problem like a Ginsu knife.

Step 1: Find the Sample Space

Our first objective is to identify all the possible outcomes of the experiment. A typical experiment involves several randomly-determined quantities. For example, the Monty Hall game involves three such quantities:

1. The door concealing the car.
2. The door initially chosen by the player.
3. The door that the host opens to reveal a goat.

Every possible combination of these randomly-determined quantities is called an *outcome*. The set of all possible outcomes is called the *sample space* for the experiment.

A *tree diagram* is a graphical tool that can help us work through the four step approach when the number of outcomes is not too large or the problem is nicely structured. In particular, we can use a tree diagram to help understand the sample space of an experiment. The first randomly-determined quantity in our experiment is the door concealing the prize. We represent this as a tree with three branches, as shown in Figure 16.1. In this diagram, the doors are called A, B, and C instead of $\mathbf{1}, \mathbf{2}$, and $\mathbf{3}$, because we'll be adding a lot of other numbers to the picture later.

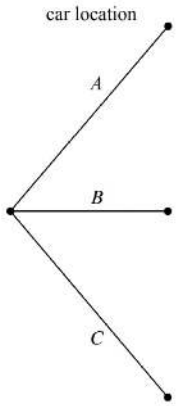


Figure 16.1 The first level in a tree diagram for the Monty Hall Problem. The branches correspond to the door behind which the car is located.

For each possible location of the prize, the player could initially choose any of the three doors. We represent this in a second layer added to the tree. Then a third layer represents the possibilities of the final step when the host opens a door to reveal a goat, as shown in Figure 16.2

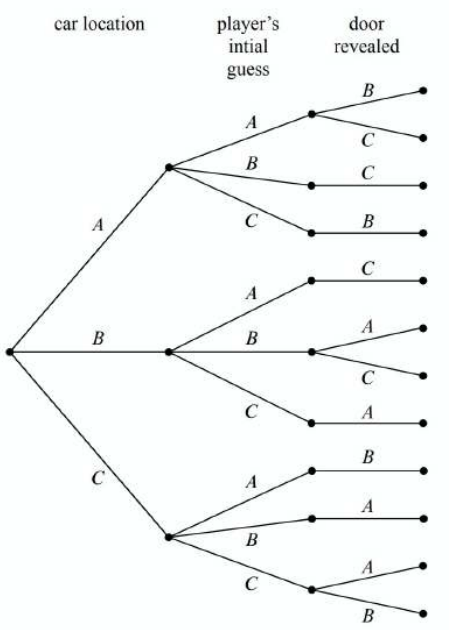


Figure 16.2 The full tree diagram for the Monty Hall Problem. The second level indicates the door initially chosen by the player. The third level indicates the door revealed by Monty Hall.

Notice that the third layer reflects the fact that the host has either one choice or two, depending on the position of the car and the door initially selected by the player. For example, if the prize is behind door A and the player picks door B, then the host must open door C. However, if the prize is behind door A and the player picks door A, then the host could open either door B or door C.

Now let's relate this picture to the terms we introduced earlier: the leaves of the tree represent *outcomes* of the experiment, and the set of all leaves represents the *sample space*. Thus, for this experiment, the sample space consists of 12 outcomes. For reference, we've labeled each outcome in Figure 16.3 with a triple of doors indicating:

(door concealing prize, door initially chosen, door opened to reveal a goat).

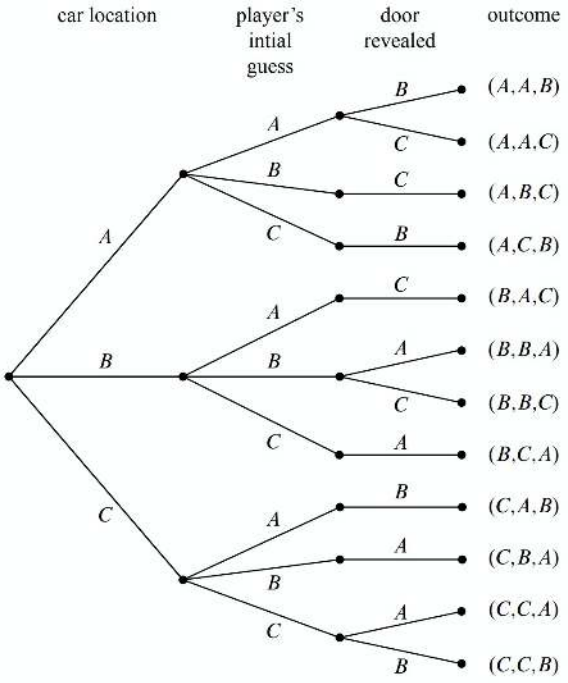


Figure 16.3 The tree diagram for the Monty Hall Problem with the outcomes labeled for each path from root to leaf. For example, outcome (A, A, B) corresponds to the car being behind door A, the player initially choosing door A, and Monty Hall revealing the goat behind door B.

In these terms, the sample space is the set

$$S = \left\{ \begin{array}{l} (A,A,B),(A,A,C),(A,B,C),(A,C,B),(B,A,C),(B,B,A), \\ (B,B,C),(B,C,A),(C,A,B),(C,B,A),(C,C,A),(C,C,B) \end{array} \right\}$$

The tree diagram has a broader interpretation as well: we can regard the whole experiment as following a path from the root to a leaf, where the branch taken at each stage is "randomly" determined. Keep this interpretation in mind; we'll use it again later.

Step 2: Define Events of Interest

Our objective is to answer questions of the form "What is the probability that . . . ?", where, for example, the missing phrase might be "the player wins by switching," "the player initially picked the door concealing the prize," or "the prize is behind door C."

A set of outcomes is called an *event*. Each of the preceding phrases characterizes an event. For example, the event [prize is behind door C] refers to the set:

$$\{(C,A,B),(C,B,A),(C,C,A),(C,C,B)\},$$

and the event [prize is behind the door first picked by the player] is:

$$\{(A,A,B),(A,A,C),(B,B,A),(B,B,C),(C,C,A),(C,C,B)\}.$$

Here we're using square brackets around a property of outcomes as a notation for the event whose outcomes are the ones that satisfy the property.

What we're really after is the event [player wins by switching]:

$$\{(A,B,C),(A,C,B),(B,A,C),(B,C,A),(C,A,B),(C,B,A)\}. \tag{16.2.1}$$

The outcomes in this event are marked with checks in Figure 16.4.

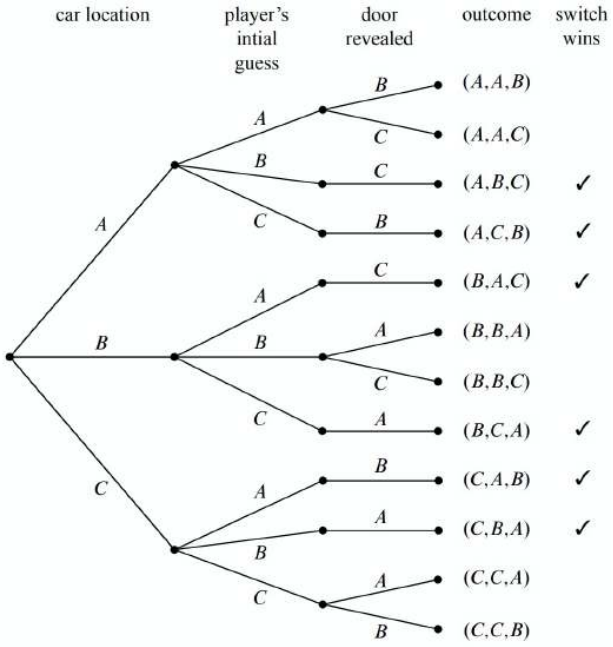


Figure 16.4 The tree diagram for the Monty Hall Problem, where the outcomes where the player wins by switching are denoted with a check mark.

Notice that exactly half of the outcomes are checked, meaning that the player wins by switching in half of all outcomes. You might be tempted to conclude that a player who switches wins with probability $1/2$. *This is wrong*. The reason is that these outcomes are not all equally likely, as we'll see shortly.

Step 3: Determine Outcome Probabilities

So far we've enumerated all the possible outcomes of the experiment. Now we must start assessing the likelihood of those outcomes. In particular, the goal of this step is to assign each outcome a probability, indicating the fraction of the time this outcome is expected to occur. The sum of all the outcome probabilities must equal one, reflecting the fact that there always must be an outcome.

Ultimately, outcome probabilities are determined by the phenomenon we're modeling and thus are not quantities that we can derive mathematically. However, mathematics can help us compute the probability of every outcome *based on fewer and more elementary modeling decisions*. In particular, we'll break the task of determining outcome probabilities into two stages.

Step 3a: Assign Edge Probabilities

First, we record a probability on each *edge* of the tree diagram. These edgeprobabilities are determined by the assumptions we made at the outset: that the prize is equally likely to be behind each door, that the player is equally likely to pick each door, and that the host is equally likely to reveal each goat, if he has a choice. Notice that when the host has no choice regarding which door to open, the single branch is assigned probability 1. For example, see Figure 16.5.

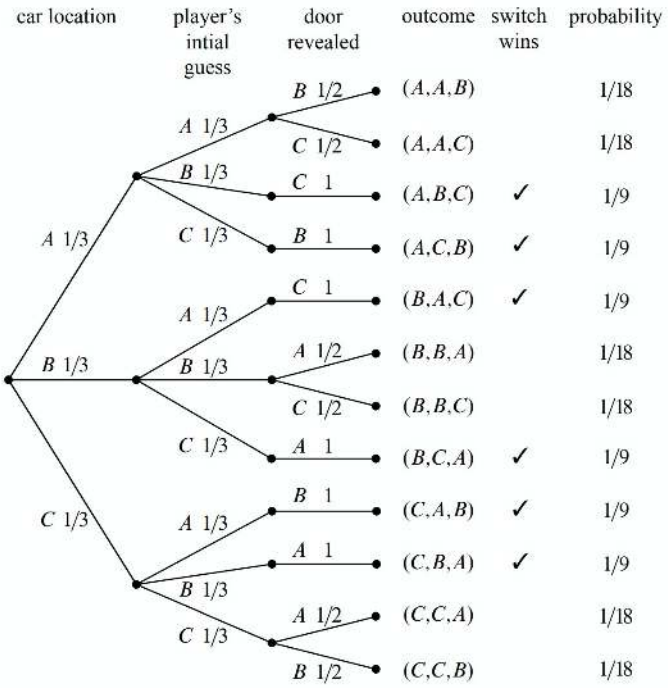


Figure 16.5 The tree diagram for the Monty Hall Problem where edge weights denote the probability of that branch being taken given that we are at the parent of that branch. For example, if the car is behind door A, then there is a 1/3 chance that the player's initial selection is door B. The rightmost column shows the outcome probabilities for the Monty Hall Problem. Each outcome probability is simply the product of the probabilities on the path from the root to the outcome leaf.

Step 3b: Compute Outcome Probabilities

Our next job is to convert edge probabilities into outcome probabilities. This is a purely mechanical process:

> calculate the probability of an outcome by multiplying the edge-probabilities on the path from the root to that outcome.

For example, the probability of the topmost outcome in Figure 16.5, (A, A, B), is

$$\frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{2} = \frac{1}{18}. \tag{16.2.2}$$

We'll examine the official justification for this rule in Section 17.4, but here's an easy, intuitive justification: as the steps in an experiment progress randomly along a path from the root of the tree to a leaf, the probabilities on the edges indicate how likely the path is to proceed along each branch. For example, a path starting at the root in our example is equally likely to go down each of the three top-level branches.

How likely is such a path to arrive at the topmost outcome, (A, A, B)? Well, there is a 1-in-3 chance that a path would follow the A-branch at the top level, a 1-in-3 chance it would continue along the A-branch at the second level, and 1-in-2 chance it would follow the B-branch at the third level. Thus, there is half of a one third of a one third chance, of arriving at the (A, A, B) leaf. That is, the chance is $1/3 \cdot 1/3 \cdot 1/2 = 1/18$—the same product (in reverse order) we arrived at in (16.2.2).

We have illustrated all of the outcome probabilities in Figure 16.5.

Specifying the probability of each outcome amounts to defining a function that maps each outcome to a probability. This function is usually called $\Pr[\text{—}]$. In these terms, we've just determined that:

$$\begin{aligned} \Pr[(A, A, B)] &= \frac{1}{18}, \\ \Pr[(A, A, C)] &= \frac{1}{18}, \\ \Pr[(A, B, C)] &= \frac{1}{9}, \\ &\text{etc.} \end{aligned} \tag{16.2.3}$$

Step 4: Compute Event Probabilities

We now have a probability for each *outcome*, but we want to determine the probability of an *event*. The probability of an event E is denoted by $\Pr[E]$, and it is the sum of the probabilities of the outcomes in E. For example, the probability of the [switching wins] event (16.2.1) is

$$\begin{aligned} &\Pr[\text{switching wins}] \\ &= \Pr[(A, B, C)] + \Pr[(A, C, B)] + \Pr[(B, A, C)] + \Pr[(B, C, A)] + \Pr[(C, A, B)] + \Pr[(C, B, A)] \\ &= \frac{1}{9} + \frac{1}{9} + \frac{1}{9} + \frac{1}{9} + \frac{1}{9} + \frac{1}{9} \\ &= \frac{2}{3}. \end{aligned} \tag{16.2.4}$$

It seems Marilyn's answer is correct! A player who switches doors wins the car with probability $2/3$. In contrast, a player who stays with his or her original door wins with probability $1/3$, since staying wins if and only if switching loses.

We're done with the problem! We didn't need any appeals to intuition or ingenious analogies. In fact, no mathematics more difficult than adding and multiplying fractions was required. The only hard part was resisting the temptation to leap to an "intuitively obvious" answer.

Alternative Interpretation of the Monty Hall Problem

Was Marilyn really right? Our analysis indicates that she was. But a more accurate conclusion is that her answer is correct *provided we accept her interpretation of the question*. There is an equally plausible interpretation in which Marilyn's answer is wrong. Notice that Craig Whitaker's original letter does not say that the host is *required* to reveal a goat and offer the player the option to switch, merely that he *did* these things. In fact, on the *Let's Make a Deal* show, Monty Hall sometimes simply opened the door that the contestant picked initially. Therefore, if he wanted to, Monty could give the option of switching only to contestants who picked the correct door initially. In this case, switching never works!

This page titled 16.2: The Four Step Method is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

16.3: Strange Dice

The four-step method is surprisingly powerful. Let's get some more practice with it. Imagine, if you will, the following scenario.

It's a typical Saturday night. You're at your favorite pub, contemplating the true meaning of infinite cardinalities, when a burly-looking biker plops down on the stool next to you. Just as you are about to get your mind around $\text{pow}(\text{pow}(\mathbb{R}))$, biker dude slaps three strange-looking dice on the bar and challenges you to a $100 wager. His rules are simple. Each player selects one die and rolls it once. The player with the lower value pays the other player $100.

Naturally, you are skeptical, especially after you see that these are not ordinary dice. Each die has the usual six sides, but opposite sides have the same number on them, and the numbers on the dice are different, as shown in Figure 16.6.

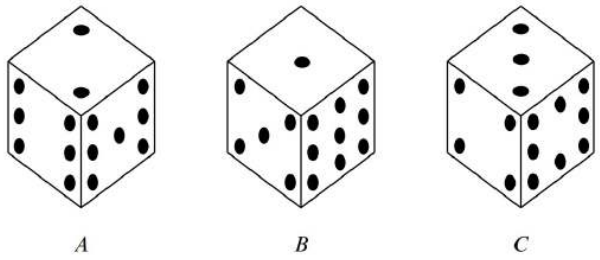


Figure 16.6 The strange dice. The number of pips on each concealed face is the same as the number on the opposite face. For example, when you roll die A, the probabilities of getting a 2, 6, or 7 are each $1/3$.

Biker dude notices your hesitation, so he sweetens his offer: he will pay you $105 if you roll the higher number, but you only need pay him $100 if he rolls higher, *and* he will let you pick a die first, after which he will pick one of the other two. The sweetened deal sounds persuasive since it gives you a chance to pick what you think is the best die, so you decide you will play. But which of the dice should you choose? Die B is appealing because it has a 9, which is a sure winner if it comes up. Then again, die A has two fairly large numbers, and die C has an 8 and no really small values.

In the end, you choose die B because it has a 9, and then biker dude selects die A. Let's see what the probability is that you will win. (Of course, you probably should have done this before picking die B in the first place.) Not surprisingly, we will use the four-step method to compute this probability.

Die A versus Die B

Step 1: Find the sample space.

The tree diagram for this scenario is shown in Figure 16.7. In particular, the sample space for this experiment are the nine pairs of values that might be rolled with Die A and Die B:

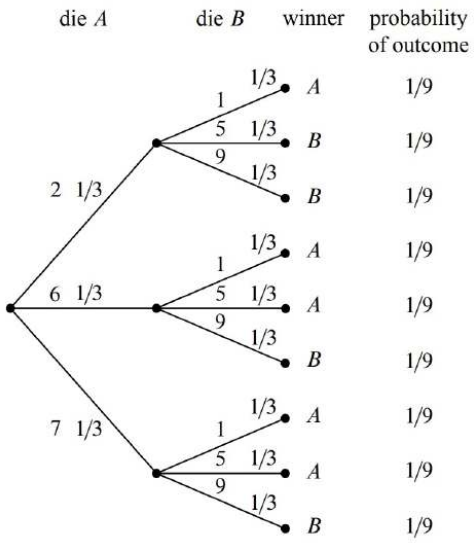


Figure 16.7 The tree diagram for one roll of die A versus die B. Die A wins with probability $5/9$.

For this experiment, the sample space is a set of nine outcomes:

$$S = \{(2,1),(2,5),(2,9),(6,1),(6,5),(6,9),(7,1),(7,5),(7,9)\}.$$

Step 2: Define events of interest.

We are interested in the event that the number on die A is greater than the number on die B. This event is a set of five outcomes:

$$\{(2,1),(6,1),(6,5),(7,1),(7,5)\}.$$

These outcomes are marked A in the tree diagram in Figure 16.7.

Step 3: Determine outcome probabilities.

To find outcome probabilities, we first assign probabilities to edges in the tree diagram. Each number on each die comes up with probability $1/3$, regardless of the value of the other die. Therefore, we assign all edges probability $1/3$. The probability of an outcome is the product of the probabilities on the corresponding root-to-leaf path, which means that every outcome has probability $1/9$. These probabilities are recorded on the right side of the tree diagram in Figure 16.7.

Step 4: Compute event probabilities.

The probability of an event is the sum of the probabilities of the outcomes in that event. In this case, all the outcome probabilities are the same, so we say that the sample space is *uniform*. Computing event probabilities for uniform sample spaces is particularly easy since you just have to compute the number of outcomes in the event. In particular, for any event E in a uniform sample space S,

$$\Pr[E] = \frac{|E|}{|S|}. \tag{16.3.1}$$

In this case, E is the event that die A beats die B, so $|E| = 5$, $|S| = 9$, and

$$\Pr[E] = 5/9.$$

This is bad news for you. Die A beats die B more than half the time and, not surprisingly, you just lost $100.

Biker dude consoles you on your "bad luck" and, given that he's a sensitive guy beneath all that leather, he offers to go double or nothing.[1] Given that your wallet only has $25 in it, this sounds like a good plan. Plus, you figure that choosing die A will give *you* the advantage.

So you choose A, and then biker dude chooses C. Can you guess who is more likely to win? (Hint: it is generally not a good idea to gamble with someone you don't know in a bar, especially when you are gambling with strange dice.)

Die A versus Die C

We can construct the tree diagram and outcome probabilities as before. The result is shown in Figure 16.8, and there is bad news again. Die C will beat die A with probability $5/9$, and you lose once again.

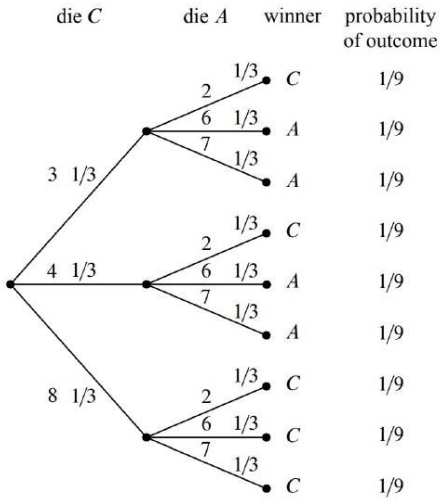


Figure 16.8 The tree diagram for one roll of die C versus die A. Die C wins with probability $5/9$.

You now owe the biker dude $200 and he asks for his money. You reply that you need to go to the bathroom.

Die B versus Die C

Being a sensitive guy, biker dude nods understandingly and offers yet another wager. This time, he'll let you have die C. He'll even let you raise the wager to $200 so you can win your money back.

This is too good a deal to pass up. You know that die C is likely to beat die A and that die A is likely to beat die B, and so die C is *surely* the best. Whether biker dude picks A or B, the odds would be in your favor this time. Biker dude must really be a nice guy.

So you pick C, and then biker dude picks B. Wait—how come you haven't caught on yet and worked out the tree diagram before you took this bet? If you do it now, you'll see by the same reasoning as before that B beats C with probability $5/9$. But surely there is a mistake! How is it possible that

$$\begin{aligned} &C \text{ beats } A \text{ with probability } 5/9,\\ &A \text{ beats } B \text{ with probability } 5/9,\\ &B \text{ beats } C \text{ with probability } 5/9? \end{aligned} \tag{16.3.2}$$

The problem is not with the math, but with your intuition. Since A will beat B more often than not, and B will beat C more often than not, it *seems* like A ought to beat C more often than not, that is, the "beats more often" relation ought to be *transitive*. But this intuitive idea is simply false: whatever die you pick, biker dude can pick one of the others and be likely to win. So picking first is actually a disadvantage, and as a result, you now owe biker dude $400.

Just when you think matters can't get worse, biker dude offers you one final wager for $1,000. This time, instead of rolling each die once, you will each roll your die twice, and your score is the sum of your rolls, and he will even let you pick your die second, that is, after he picks his. Biker dude chooses die B. Now you know that die A will beat die B with probability $5/9$ on one roll, so, jumping at this chance to get ahead, you agree to play, and you pick die A. After all, you figure that since a roll of die A beats a roll of die B more often that not, two rolls of die A are even more likely to beat two rolls of die B, right?

Wrong! (Did we mention that playing strange gambling games with strangers in a bar is a bad idea?)

Rolling Twice

If each player rolls twice, the tree diagram will have four levels and $3^4 = 81$ outcomes. This means that it will take a while to write down the entire tree diagram. But it's easy to write down the first two levels as in Figure 16.9(a) and then notice that the remaining two levels consist of nine identical copies of the tree in Figure 16.9(b).

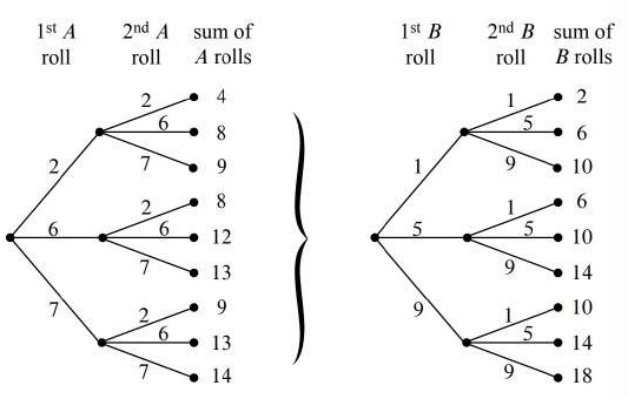


Figure 16.9 Parts of the tree diagram for die B versus die A where each die is rolled twice. The first two levels are shown in (a). The last two levels consist of nine copies of the tree in (b).

The probability of each outcome is $(1/3)^4 = 1/81$ and so, once again, we have a uniform probability space. By equation (16.3.1), this means that the probability that A wins is the number of outcomes where A beats B divided by 81.

To compute the number of outcomes where A beats B, we observe that the two rolls of die A result in nine equally likely outcomes in a sample space S_A in which the two-roll sums take the values

$$(4, 8, 8, 9, 9, 12, 13, 13, 14).$$

Likewise, two rolls of die B result in nine equally likely outcomes in a sample space S_B in which the two-roll sums take the values

$$(2, 6, 6, 10, 10, 10, 14, 14, 18).$$

We can treat the outcome of rolling both dice twice as a pair $(x, y) \in S_A \times S_B$, where A wins iff the sum of the two A-rolls of outcome x is larger the sum of the two B-rolls of outcome y. If the A-sum is 4, there is only one y with a smaller B-sum, namely, when the B-sum is 2. If the A-sum is 8, there are three y's with a smaller B-sum, namely, when the B-sum is 2 or 6. Continuing the count in this way, the number of pairs (x, y) for which the A-sum is larger than the B-sum is

$$1 + 3 + 3 + 3 + 3 + 6 + 6 + 6 + 6 = 37.$$

A similar count shows that there are 42 pairs for which B-sum is larger than the A-sum, and there are two pairs where the sums are equal, namely, when they both equal 14. This means that A *loses* to B with probability $42/81 > 1/2$ and ties with probability $2/81$. Die A wins with probability only $37/81$.

How can it be that A is more likely than B to win with one roll, but B is more likely to win with two rolls? Well, why not? The only reason we'd think otherwise is our unreliable, untrained intuition. (Even the authors were surprised when they first learned about this, but at least they didn't lose $1400 to biker dude.) In fact, the die strength reverses no matter which two die we picked. So for one roll,

$$A \succ B \succ C \succ A, \tag{16.3.3}$$

but for two rolls,

$$A \prec B \prec C \prec A, \tag{16.3.4}$$

where we have used the symbols $\succ$ and $\prec$ to denote which die is more likely to result in the larger value.

The weird behavior of the three strange dice above generalizes in a remarkable way: there are arbitrarily large sets of dice which will beat each other in any desired pattern according to how many times the dice are rolled.[2]

[2] **TBA - Reference Ron Graham paper.**

This page titled 16.3: Strange Dice is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

16.4: The Birthday Principle

There are 95 students in a class. What is the probability that some birthday is shared by two people? Comparing 95 students to the 365 possible birthdays, you might guess the probability lies somewhere around $1/4$—but you'd be wrong: the probability that there will be two people in the class with matching birthdays is actually more than 0.9999.

To work this out, we'll assume that the probability that a randomly chosen student has a given birthday is $1/d$. We'll also assume that a class is composed of n randomly and independently selected students. Of course $d = 365$ and $n = 95$ in this case, but we're interested in working things out in general. These randomness assumptions are not really true, since more babies are born at certain times of year, and students' class selections are typically not independent of each other, but simplifying in this way gives us a start on analyzing the problem. More importantly, these assumptions are justifiable in important computer science applications of birthday matching. For example, birthday matching is a good model for collisions between items randomly inserted into a hash table. So we won't worry about things like spring procreation preferences that make January birthdays more common, or about twins' preferences to take classes together (or not).

Exact Formula for Match Probability

There are d^n sequences of n birthdays, and under our assumptions, these are equally likely. There are $d(d-1)(d-2)\cdots(d-(n-1))$ length n sequences of distinct birthdays. That means the probability that everyone has a different birthday is:[3]

$$\begin{aligned}
&\frac{d(d-1)(d-2)\cdots(d-(n-1))}{d^n} \\
&= \frac{d}{d}\cdot\frac{d-1}{d}\cdot\frac{d-2}{d}\cdots\frac{d-(n-1)}{d} \qquad (16.4.1)\\
&= \left(1-\frac{0}{d}\right)\left(1-\frac{1}{d}\right)\left(1-\frac{2}{d}\right)\cdots\left(1-\frac{n-1}{d}\right) \\
&< e^0\cdot e^{-1/d}\cdot e^{-2/d}\cdots e^{-(n-1)/d} \quad (\text{since } 1+x<e^x) \\
&= e^{-\left(\sum_{i=1}^{n-1} i/d\right)} \\
&= e^{-(n(n-1)/2d)}. \qquad (16.4.2)
\end{aligned}$$

For $n = 95$ and $d = 365$, the value of (16.4.2) is less than $1/200,000$, which means the probability of having some pair of matching birthdays actually is more than $1 - 1/200,000 > 0.99999$. So it would be pretty astonishing if there were no pair of students in the class with matching birthdays.

For $d \le n^2/2$, the probability of no match turns out to be asymptotically equal to the upper bound (16.4.2). For $d \le n^2/2$ in particular, the probability of no match is asymptotically equal to $1/e$. This leads to a rule of thumb which is useful in many contexts in computer science:

The Birthday Principle

If there are d days in a year and $\sqrt{2d}$ people in a room, then the probability that two share a birthday is about $1 - 1/e \approx 0.632$.

For example, the Birthday Principle says that if you have $\sqrt{2\cdot 365} \approx 27$ people in a room, then the probability that two share a birthday is about 0.632. The actual probability is about 0.626, so the approximation is quite good.

Among other applications, it implies that to use a hash function that maps n items into a hash table of size d, you can expect many collisions if n^2 is more than a small fraction of d. The Birthday Principle also famously comes into play as the basis of "birthday attacks" that crack certain cryptographic systems.

[3]The fact that $1 - x < e^{-x}$ for all $x > 0$ follows by truncating the Taylor series $e^{-x} = 1 - x + x^2/2! - x^3/3! + \cdots$. The approximation $e^{-x} \approx 1 - x$ is pretty accurate when x is small.

This page titled 16.4: The Birthday Principle is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

16.5: Set Theory and Probability

Let's abstract what we've just done into a general mathematical definition of sample spaces and probability.

Probability Spaces

Definition 16.5.1

A countable *sample space* S is a nonempty countable set.[4] An element $\omega \in S$ is called an *outcome*. A subset of S is called an *event*.

Definition 16.5.2

A *probability function* on a sample space S is a total function $\Pr : S \to \mathbb{R}$ such that

- $\Pr[\omega] \geq 0$ for all $\omega \in S$, and
- $\sum_{\omega \in S} \Pr[\omega] = 1$.

A sample space together with a probability function is called a *probability space*. For any event $E \subseteq S$, the *probability* of E is defined to be the sum of the probabilities of the outcomes in E:

$$\Pr[E] ::= \sum_{\omega \in E} \Pr[\omega].$$

In the previous examples there were only finitely many possible outcomes, but we'll quickly come to examples that have a countably infinite number of outcomes.

The study of probability is closely tied to set theory because any set can be a sample space and any subset can be an event. General probability theory deals with uncountable sets like the set of real numbers, but we won't need these, and sticking to countable sets lets us define the probability of events using sums instead of integrals. It also lets us avoid some distracting technical problems in set theory like the Banach-Tarski "paradox" mentioned in Chapter 7.

Probability Rules from Set Theory

Most of the rules and identities that we have developed for finite sets extend very naturally to probability.

An immediate consequence of the definition of event probability is that for *disjoint* events E and F,

$$\Pr[E \cup F] = \Pr[E] + \Pr[F].$$

This generalizes to a countable number of events:

Rule 16.5.3 (Sum Rule). *If $E_0, E_1, \ldots, E_n, \ldots$ are pairwise disjoint events, then*

$$\Pr\left[\bigcup_{n \in \mathbb{N}} E_n\right] = \sum_{n \in \mathbb{N}} \Pr[E_n].$$

The Sum Rule lets us analyze a complicated event by breaking it down into simpler cases. For example, if the probability that a randomly chosen MIT student is native to the United States is 60%, to Canada is 5%, and to Mexico is 5%, then the probability that a random MIT student is native to one of these three countries is 70%.

Another consequence of the Sum Rule is that $\Pr[A] + \Pr[\overline{A}] = 1$, which follows because $\Pr[S] = 1$ and S is the union of the disjoint sets A and $\overline{A}$. This equation often comes up in the form:

$$\Pr[\overline{A}] = 1 - \Pr[A]. \qquad \text{(Complement Rule)}$$

Sometimes the easiest way to compute the probability of an event is to compute the probability of its complement and then apply this formula.

Some further basic facts about probability parallel facts about cardinalities of finite sets. In particular:

$$\begin{array}{ll} \Pr[B-A] = \Pr[B] - \Pr[A \cap B], & \text{(Difference Rule)} \\ \Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B], & \text{(Inclusion-Exclusion)} \\ \Pr[A \cup B] \leq \Pr[A] + \Pr[B], & \text{(Boole's Inequality)} \\ \text{If } A \subseteq B, \text{then } \Pr[A] \leq \Pr[B]. & \text{(Monotonicity Rule)} \end{array} \tag{16.5.1}$$

The Difference Rule follows from the Sum Rule because B is the union of the disjoint sets $B-A$ and $A \cap B$. Inclusion-Exclusion then follows from the Sum and Difference Rules, because $A \cup B$ is the union of the disjoint sets A and $B-A$. Boole's inequality is an immediate consequence of Inclusion-Exclusion since probabilities are nonnegative. Monotonicity follows from the definition of event probability and the fact that outcome probabilities are nonnegative.

The two-event Inclusion-Exclusion equation above generalizes to any finite set of events in the same way as the corresponding Inclusion-Exclusion rule for n sets. Boole's inequality also generalizes to both finite and countably infinite sets of events:

Rule 16.5.4 (Union Bound).

$$\Pr[E_1 \cup \cdots \cup E_n \cup \cdots] \leq \Pr[E_1] + \cdots + \Pr[E_n] + \cdots. \tag{16.5.2}$$

The Union Bound is useful in many calculations. For example, suppose that E_i is the event that the i-th critical component among n components in a spacecraft fails. Then $E_1 \cup \cdots \cup E_n$ is the event that *some* critical component fails. If $\sum_{i=1}^{n} \Pr[E_i]$ is small, then the Union Bound can provide a reassuringly small upper bound on this overall probability of critical failure.

Uniform Probability Spaces

Definition 16.5.5

A finite probability space, S, is said to be *uniform* if $\Pr[\omega]$ is the same for every outcome $\omega \in S$.

As we saw in the strange dice problem, uniform sample spaces are particularly easy to work with. That's because for any event $E \subseteq S$,

$$\Pr[E] = \frac{|E|}{|S|}. \tag{16.5.3}$$

This means that once we know the cardinality of E and S, we can immediately obtain $\Pr[E]$. That's great news because we developed lots of tools for computing the cardinality of a set in Part III.

For example, suppose that you select five cards at random from a standard deck of 52 cards. What is the probability of having a full house? Normally, this question would take some effort to answer. But from the analysis in Section 14.7.2, we know that

$$|S| = \binom{52}{5}$$

and

$$|E| = 13 \cdot \binom{4}{3} \cdot 12 \cdot \binom{4}{2}$$

where E is the event that we have a full house. Since every five-card hand is equally likely, we can apply equation (16.5.3) to find that

$$\begin{aligned} \Pr[E] &= \frac{13 \cdot 12 \cdot \binom{4}{3} \cdot \binom{4}{2}}{\binom{52}{5}} \\ &= \frac{13 \cdot 12 \cdot 4 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2}{52 \cdot 51 \cdot 50 \cdot 49 \cdot 48} = \frac{18}{12495} \\ &\approx \frac{1}{694}. \end{aligned} \tag{16.5.4}$$

Infinite Probability Spaces

Infinite probability spaces are fairly common. For example, two players take turns flipping a fair coin. Whoever flips heads first is declared the winner. What is the probability that the first player wins? A tree diagram for this problem is shown in Figure 16.10.

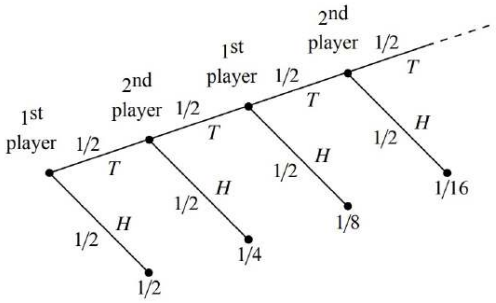


Figure 16.10 The tree diagram for the game where players take turns flipping a fair coin. The first player to flip heads wins.

The event that the first player wins contains an infinite number of outcomes, but we can still sum their probabilities:

$$\begin{aligned}\Pr[\text{first player wins}] &= \frac{1}{2}+\frac{1}{8}+\frac{1}{32}+\frac{1}{128}+\cdots \\ &= \frac{1}{2}\sum_{n=0}^{\infty}\left(\frac{1}{4}\right)^n \\ &= \frac{1}{2}\left(\frac{1}{1-1/4}\right)=\frac{2}{3}\end{aligned} \tag{16.5.5}$$

Similarly, we can compute the probability that the second player wins:

$$\Pr[\text{second player wins}] = \frac{1}{4}+\frac{1}{16}+\frac{1}{64}+\frac{1}{256}+\cdots=\frac{1}{3}.$$

In this case, the sample space is the infinite set

$$S ::= \{T^n H \mid n \in \mathbb{N}\},$$

where T^n stands for a length n string of T's. The probability function is

$$\Pr[T^n H] ::= \frac{1}{2^{n+1}}.$$

To verify that this is a probability space, we just have to check that all the probabilities are nonnegative and that they sum to 1. The given probabilities are all nonnegative, and applying the formula for the sum of a geometric series, we find that

$$\sum_{n\in\mathbb{N}}\Pr[T^n H] ::= \sum_{n\in\mathbb{N}}\frac{1}{2^{n+1}} = 1.$$

Notice that this model does not have an outcome corresponding to the possibility that both players keep flipping tails forever. (In the diagram, flipping forever corresponds to following the infinite path in the tree without ever reaching a leaf/outcome.) If leaving this possibility out of the model bothers you, you're welcome to fix it by adding another outcome, $\omega_{forever}$, to indicate that that's what happened. Of course since the probabilities of the other outcomes already sum to 1, you have to define the probability of $\omega_{forever}$ to be 0. Now outcomes with probability zero will have no impact on our calculations, so there's no harm in adding it in if it makes you happier. On the other hand, in countable probability spaces it isn't necessary to have outcomes with probability zero, and we will generally ignore them.

[4]Yes, sample spaces can be infinite. If you did not read Chapter 7, don't worry—*countable* just means that you can list the elements of the sample space as $\omega_0, \omega_1, \omega_2, \ldots$

This page titled 16.5: Set Theory and Probability is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

17: Conditional Probability

17.1: Monty Hall Confusion
17.2: Definition and Notation
17.3: The Four-Step Method for Conditional Probability
17.4: Why Tree Diagrams Work
17.5: The Law of Total Probability
17.6: Simpson’s Paradox
17.7: Independence
17.8: Mutual Independence

This page titled 17: Conditional Probability is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

17.1: Monty Hall Confusion

Remember how we said that the Monty Hall problem confused even professional mathematicians? Based on the work we did with tree diagrams, this may seem surprising—the conclusion we reached followed routinely and logically. How could this problem be so confusing to so many people?

Well, one flawed argument goes as follows: let's say the contestant picks door A. And suppose that Carol, Monty's assistant, opens door B and shows us a goat. Let's use the tree diagram 16.3 from Chapter 16 to capture this situation. There are exactly three outcomes where contestant chooses door A, and there is a goat behind door B:

$$(A, A, B), (A, A, C), (C, A, B). \tag{17.1.1}$$

These outcomes have respective probabilities 1/18, 1/18, 1/9.

Among those outcomes, switching doors wins only on the last outcome, (C, A, B). The other two outcomes *together* have the *same* 1/9 probability as the last one So in this situation, the probability that we win by switching is the *same* as the probability that we lose. In other words, in this situation, switching isn't any better than sticking!

Something has gone wrong here, since we know that the actual probability of winning by switching in 2/3. The mistaken conclusion that sticking or switching are equally good strategies comes from a common blunder in reasoning about how probabilities change given some information about what happened. We have asked for the probability that one event, [win by switching], happens, *given* that another event, [pick A \text{AND} goat at B], happens. We use the notation

$$\mathbf{Pr}[[\text{win by switching}] \mid [\text{pick A AND goat at B}]]$$

for this probability which, by the reasoning above, equals 1/2.

Behind the Curtain

A "given" condition is essentially an instruction to focus on only some of the possible outcomes. Formally, we're defining a new sample space consisting only of some of the outcomes. In this particular example, we're given that the player chooses door A and that there is a goat behind B. Our new sample space therefore consists solely of the three outcomes listed in (17.1.1). In the opening of Section 17.1, we calculated the conditional probability of winning by switching given that one of these outcome happened, by weighing the 1/9 probability of the win-by-switching outcome, (C, A, B), against the $1/18 + 1/18 + 1/9$ probability of the three outcomes in the new sample space.

$$\begin{aligned}
&\mathbf{Pr}[[\text{win by switching}] \mid [\text{pick A AND goat at B}]] \\
&= \mathbf{Pr}[(C, A, B) \mid \{(C, A, B), (A, A, B), (A, A, C)\}] + \frac{\mathbf{Pr}[(C, A, B)]}{\mathbf{Pr}[\{(C, A, B), (A, A, B), (A, A, C)\}]} \\
&= \frac{1/9}{1/18 + 1/18 + 1/9} = \frac{1}{2}.
\end{aligned}$$

There is nothing wrong with this calculation. So how come it leads to an incorrect conclusion about whether to stick or switch? The answer is that this was the wrong thing to calculate, as we'll explain in the next section.

This page titled 17.1: Monty Hall Confusion is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

17.2: Definition and Notation

The expression $\Pr[X \mid Y]$ denotes the probability of event X, given that event Y happens. In the example above, event X is the event of winning on a switch, and event Y is the event that a goat is behind door B and the contestant chose door A. We calculated $\Pr[X \mid Y]$ using a formula which serves as the definition of conditional probability:

Definition 17.2.1

Let X and Y be events where Y has nonzero probability. Then

$$\Pr[X \mid Y] ::= \frac{\Pr[X \cap Y]}{\Pr[Y]}. \tag{17.2.1}$$

The conditional probability $\Pr[X \mid Y]$ is undefined when the probability of event Y is zero. To avoid cluttering up statements with uninteresting hypotheses that conditioning events like Y have nonzero probability, we will make an implicit assumption from now on that all such events have nonzero probability.

Pure probability is often counterintuitive, but conditional probability can be even worse. Conditioning can subtly alter probabilities and produce unexpected results in randomized algorithms and computer systems as well as in betting games. But Definition 17.2.1 is very simple and causes no trouble—provided it is properly applied.

What went wrong

So if everything in the opening Section 17.1 is mathematically sound, why does it seem to contradict the results that we established in Chapter 16? The problem is a common one: *we chose the wrong condition*. In our initial description of the scenario, we learned the location of the goat when Carol opened door B. But when we defined our condition as "the contestant opens A and the goat is behind B," we included the outcome (A, A, C) in which Carol opens door C! The correct conditional probability should have been "what are the odds of winning by switching given the contestant chooses door A and Carol opens door B." By choosing a condition that did not reflect everything known. we inadvertently included an extraneous outcome in our calculation. With the correct conditioning, we still win by switching 1/9 of the time, but the smaller set of known outcomes has smaller total probability:

$$\Pr[\{(A, A, B), (C, A, B)\}] = \frac{1}{18} + \frac{1}{9} = \frac{3}{18}.$$

The conditional probability would then be:

$$\begin{aligned} &\Pr[[\text{[win by switching]} \mid [\text{[pick A AND Carol opens B]}] \\ &= \Pr[(C, A, B) \mid \{(C, A, B), (A, A, B)\}] + \frac{\Pr[(C, A, B)]}{\Pr[\{(C, A, B), (A, A, B)\}]} = \frac{1/9}{1/9 + 1/18} = \frac{1}{2}. \end{aligned}$$

which is exactly what we already deduced from the tree diagram 16.2 in the previous chapter.

The O. J. Simpson Trial

In an opinion article in the *New York Times*, Steven Strogatz points to the O. J. Simpson trial as an example of poor choice of conditions. O. J. Simpson was a retired football player who was accused, and later acquitted, of the murder of his wife, Nicole Brown Simpson. The trial was widely publicized and called the "trial of the century." Racial tensions, allegations of police misconduct, and new-at-the-time DNA evidence captured the public's attention. But Strogatz, citing mathematician and author I.J. Good, focuses on a less well-known aspect of the case: whether O. J.'s history of abuse towards his wife was admissible into evidence.

The prosecution argued that abuse is often a precursor to murder, pointing to statistics indicating that an abuser was as much as ten times more likely to commit murder than was a random indidual. The defense, however, countered with statistics indicating that the odds of an abusive husband murdering his wife were "infinitesimal," roughly 1 in 2500. Based on those numbers, the actual relevance of a history of abuse to a murder case would appear limited at best. According to the defense, introducing that history would make the jury hate Simpson but would lack any probitive value. Its discussion should be barred as prejudicial.

In other words, both the defense and the prosecution were arguing conditional probability, specifically the likelihood that a woman will be murdered by her husband, given that her husband abuses her. But both defense and prosecution omitted a vital piece of data from their calculations: Nicole Brown Simpson *was* murdered. Strogatz points out that based on the defense's numbers and the crime statistics of the time, the probability that a woman was murdered by her abuser, given that she was abused *and* murdered, is around 80%.

Strogatz's article goes into more detail about the calculations behind that 80% figure. But the real point we wanted to make is that conditional probability is used and misused all the time, and even experts under public scrutiny make mistakes.

This page titled 17.2: Definition and Notation is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

17.3: The Four-Step Method for Conditional Probability

In a best-of-three tournament, the local C-league hockey team wins the first game with probability $1/2$. In subsequent games, their probability of winning is determined by the outcome of the previous game. If the local team won the previous game, then they are invigorated by victory and win the current game with probability $2/3$. If they lost the previous game, then they are demoralized by defeat and win the current game with probability only $1/3$. What is the probability that the local team wins the tournament, given that they win the first game?

This is a question about a conditional probability. Let A be the event that the local team wins the tournament, and let B be the event that they win the first game. Our goal is then to determine the conditional probability $\Pr[A \mid B]$.

We can tackle conditional probability questions just like ordinary probability problems: using a tree diagram and the four step method. A complete tree diagram is shown in Figure 17.1.

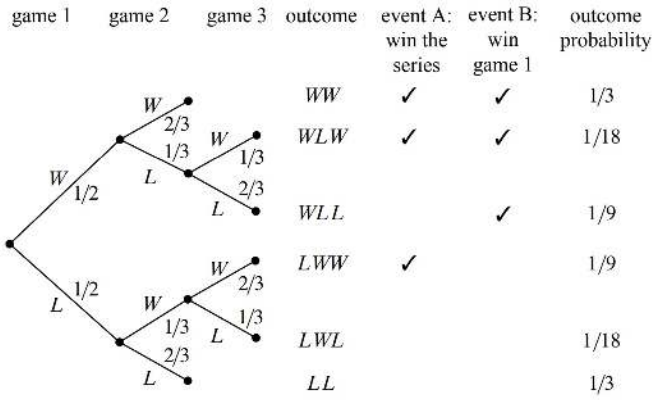


Figure 17.1 The tree diagram for computing the probability that the local team wins two out of three games given that they won the first game.

Step 1: Find the Sample Space

Each internal vertex in the tree diagram has two children, one corresponding to a win for the local team (labeled W) and one corresponding to a loss (labeled L). The complete sample space is:

$$S = \{WW, WLW, WLL, LWW, LWL, LL\}.$$

Step 2: Define Events of Interest

The event that the local team wins the whole tournament is:

$$T = \{WW, WLW, LWW\}.$$

And the event that the local team wins the first game is:

$$F = \{WW, WLW, WLL\}.$$

The outcomes in these events are indicated with check marks in the tree diagram in Figure 17.1.

Step 3: Determine Outcome Probabilities

Next, we must assign a probability to each outcome. We begin by labeling edges as specified in the problem statement. Specifically, the local team has a $1/2$ chance of winning the first game, so the two edges leaving the root are each assigned probability $1/2$. Other edges are labeled $1/3$ or $2/3$ based on the outcome of the preceding game. We then find the probability of each outcome by multiplying all probabilities along the corresponding root-to-leaf path. For example, the probability of outcome WLL is:

$$\frac{1}{2} \cdot \frac{1}{3} \cdot \frac{2}{3} = \frac{1}{9}.$$

Step 4: Compute Event Probabilities

We can now compute the probability that the local team wins the tournament, given that they win the first game:

$$\begin{aligned}\Pr[A \mid B] &= \frac{\Pr[A \cap B]}{\Pr[B]} \\ &= \frac{\Pr[\{WW, WLW\}]}{\Pr[\{WW, WLW, WLL\}]} \\ &= \frac{1/3 + 1/18}{1/3 + 1/18 + 1/9} \\ &= \frac{7}{9}.\end{aligned} \tag{17.3.1}$$

We're done! If the local team wins the first game, then they win the whole tournament with probability $7/9$.

This page titled 17.3: The Four-Step Method for Conditional Probability is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

17.4: Why Tree Diagrams Work

We've now settled into a routine of solving probability problems using tree diagrams. But we've left a big question unaddressed: mathematical justification behind those funny little pictures. Why do they work?

The answer involves conditional probabilities. In fact, the probabilities that we've been recording on the edges of tree diagrams *are* conditional probabilities. For example, consider the uppermost path in the tree diagram for the hockey team problem, which corresponds to the outcome WW. The first edge is labeled $1/2$, which is the probability that the local team wins the first game. The second edge is labeled $2/3$, which is the probability that the local team wins the second game, given that they won the first—a conditional probability! More generally, on each edge of a tree diagram, we record the probability that the experiment proceeds along that path, given that it reaches the parent vertex.

So we've been using conditional probabilities all along. For example, we concluded that:

$$\Pr[WW] = \frac{1}{2} \cdot \frac{2}{3} = \frac{1}{3}.$$

Why is this correct?

The answer goes back to Definition 17.2.1 of conditional probability which could be written in a form called the Product Rule for conditional probabilities:

Rule: Conditional Probability Product Rule - 2 Events

$$\Pr[E_1 \cap E_2] = \Pr[E_1] \cdot \Pr[E_2 \mid E_1].$$

Multiplying edge probabilities in a tree diagram amounts to evaluating the right side of this equation. For example:

$$\begin{aligned} &\Pr[\text{win first game} \cap \text{win second game}] \\ &\quad = \Pr[\text{win first game}] \cdot \Pr[\text{win second game} \mid \text{win first game}] \\ &\quad = \frac{1}{2} \cdot \frac{2}{3}. \end{aligned} \tag{17.4.1}$$

So the Conditional Probability Product Rule is the formal justification for multiplying edge probabilities to get outcome probabilities.

To justify multiplying edge probabilities along a path of length three, we need a rule for three events:

$$\Pr[E_1 \cap E_2 \cap E_3] = \Pr[E_1] \cdot \Pr[E_2 \mid E_1] \cdot \Pr[E_3 \mid E_1 \cap E_2].$$

An n-event version of the Rule is given in Problem 17.1, but its form should be clear from the three event version.

Probability of Size-k Subsets

As a simple application of the product rule for conditional probabilities, we can use the rule to calculate the number of size-k subsets of the integers $[1..n]$. Of course we already know this number is $\binom{n}{k}$, but now the rule will give us a new derivation of the formula for $\binom{n}{k}$.

Let's pick some size-k subset, $S \subseteq [1..n]$, as a target. Suppose we choose a size-k subset at random, with all subsets of $[1..n]$ equally likely to be chosen, and let p be the probability that our randomly chosen equals this target. That is, the probability of picking S is p, and since all sets are equally likely to be chosen, the number of size-k subsets equals $1/p$.

So what's p? Well, the probability that the *smallest* number in the random set is one of the k numbers in S is k/n. Then, *given* that the smallest number in the random set is in S, the probability that the second smallest number in the random set is one of the remaining $k-1$ elements in S is $(k-1)/(n-1)$. So by the product rule, the probability that the *two* smallest numbers in the random set are both in S is

$$\frac{k}{n} \cdot \frac{k-1}{n-1}.$$

Next, given that the two smallest numbers in the random set are in S, the probability that the third smallest number is one of the $k-2$ remaining elements in S is $(k-2)/(n-2)$. So by the product rule, the probability that the *three* smallest numbers in the random set are all in S is

$$\frac{k}{n} \cdot \frac{k-1}{n-1} \cdot \frac{k-2}{n-2}.$$

Continuing in this way, it follows that the probability that *all* k elements in the randomly chosen set are in S, that is, the probabilty that the randomly chosen set equals the target, is

$$\begin{aligned} p &= \frac{k}{n} \cdot \frac{k-1}{n-1} \cdot \frac{k-2}{n-2} \cdots \frac{k-(k-1)}{n-(k-1)} \\ &= \frac{k \cdot (k-1) \cdot (k-1) \cdots 1}{n \cdot (n-1) \cdot (n-2) \cdots (n-(k-1))} \\ &= \frac{k!}{n!/(n-k)!} \\ &= \frac{k!(n-k)!}{n!}. \end{aligned} \tag{17.4.2}$$

So we have again shown the number of size-k subsets of $[1..n]$, namely $1/p$, is

$$\frac{n!}{k!/(n-k)!}.$$

Medical Testing

Breast cancer is a deadly disease that claims thousands of lives every year. Early detection and accurate diagnosis are high priorities, and routine mammograms are one of the first lines of defense. They're not very accurate as far as medical tests go, but they are correct between 90% and 95% of the time, which seems pretty good for a relatively inexpensive non-invasive test.[1] However, mammogram results are also an example of conditional probabilities having counterintuitive consequences. If the test was positive for breast cancer in you or a loved one, and the test is better than 90% accurate, you'd naturally expect that to mean there is better than 90% chance that the disease was present. But a mathematical analysis belies that gut instinct. Let's start by precisely defining how accurate a mammogram is:

1. If you have the condition, there is a 10% chance that the test will say you do not have it. This is called a "false negative."
2. If you do not have the condition, there is a 5% chance that the test will say you do. This is a "false positive."

Four Steps Again

Now suppose that we are testing middle-aged women with no family history of cancer. Among this cohort, incidence of breast cancer rounds up to about 1%.

Step 1: Find the Sample Space

The sample space is found with the tree diagram in Figure 17.2.

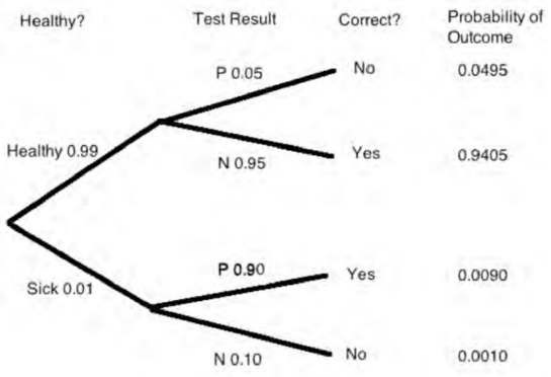


Figure 17.2 The tree diagram for a breast cancer test.

Step 2: Define Events of Interest

Let A be the event that the person has breast cancer. Let B be the event that the test was positive. The outcomes in each event are marked in the tree diagram. We want to find $\Pr[A \mid B]$, the probability that a person has breast cancer, given that the test was positive.

Step 3: Find Outcome Probabilities

First, we assign probabilities to edges. These probabilities are drawn directly from the problem statement. By the Product Rule, the probability of an outcome is the product of the probabilities on the corresponding root-to-leaf path. All probabilities are shown in Figure 17.2.

Step 4: Compute Event Probabilities

From Definition 17.2.1, we have

$$\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]} = \frac{0.009}{0.009 + 0.0495} \approx 15.4\%$$

So, if the test is positive, then there is an 84.6% chance that the result is incorrect, even though the test is nearly 95% accurate! So this seemingly pretty accurate test doesn't tell us much. To see why percent accuracy is no guarantee of value, notice that there is a simple way to make a test that is 99% accurate: always return a negative result! This test gives the right answer for all healthy people and the wrong answer only for the 1% that actually have cancer. This 99% accurate test tells us nothing; the "less accurate" mammogram is still a lot more useful.

Natural Frequencies

That there is only about a 15% chance that the patient actually has the condition when the test say so may seem surprising at first, but it makes sense with a little thought. There are two ways the patient could test positive: first, the patient could have the condition and the test could be correct; second, the patient could be healthy and the test incorrect. But almost everyone is healthy! The number of healthy individuals is so large that even the mere 5% with false positive results overwhelm the number of genuinely positive results from the truly ill.

Thinking like this in terms of these "natural frequencies" can be a useful tool for interpreting some of the strange seeming results coming from those formulas. For example, let's take a closer look at the mammogram example.

Imagine 10,000 women in our demographic. Based on the frequency of the disease, we'd expect 100 of them to have breast cancer. Of those, 90 would have a positve result. The remaining 9,900 woman are healthy, but 5% of them—500, give or take—will show a false positive on the mammogram. That gives us 90 real positives out of a little fewer than 600 positives. An 85% error rate isn't so surprising after all.

Posteriori Probabilities

If you think about it much, the medical testing problem we just considered could start to trouble you. You may wonder if a statement like "If someone tested positive, then that person has the condition with probability 18%" makes sense, since a given person being tested either has the disease or they don't.

One way to understand such a statement is that it just means that 15% of the people who test positive will actually have the condition. Any particular person has it or they don't, but a *randomly selected* person among those who test positive will have the condition with probability 15%.

But what does this 15% probability tell you if you *personally* got a positive result? Should you be relieved that there is less than one chance in five that you have the disease? Should you worry that there is nearly one chance in five that you do have the disease? Should you start treatment just in case? Should you get more tests?

These are crucial practical questions, but it is important to understand that they are not *mathematical* questions. Rather, these are questions about statistical judgements and the philosophical meaning of probability. We'll say a bit more about this after looking at one more example of after-the-fact probabilities.

The Hockey Team in Reverse

Suppose that we turn the hockey question around: what is the probability that the local C-league hockey team won their first game, given that they won the series?

As we discussed earlier, some people find this question absurd. If the team has already won the tournament, then the first game is long since over. Who won the first game is a question of fact, not of probability. However, our mathematical theory of probability contains no notion of one event preceding another. There is no notion of time at all. Therefore, from a mathematical perspective, this is a perfectly valid question. And this is also a meaningful question from a practical perspective. Suppose that you're told that the local team won the series, but not told the results of individual games. Then, from your perspective, it makes perfect sense to wonder how likely it is that local team won the first game.

A conditional probability $\Pr[B \mid A]$ is called *a posteriori* if event B precedes event A in time. Here are some other examples of a posteriori probabilities:

- The probability it was cloudy this morning, given that it rained in the afternoon.
- The probability that I was initially dealt two queens in Texas No Limit Hold 'Em poker, given that I eventually got four-of-a-kind.

from ordinary probabilities; the distinction comes from our view of causality, which is a philosophical question rather than a mathematical one.

Let's return to the original problem. The probability that the local team won their first game, given that they won the series is $\Pr[B \mid A]$. We can compute this using the definition of conditional probability and the tree diagram in Figure 17.1:

$$\Pr[B \mid A] = \frac{\Pr[B \cap A]}{\Pr[A]} = \frac{1/3 + 1/18}{1/3 + 1/18 + 1/9} = \frac{7}{9}.$$

In general, such pairs of probabilities are related by Bayes' Rule:

Theorem 17.4.1: Bayes' Rule

$$\Pr[B \mid A] = \frac{\Pr[A \mid B] \cdot \Pr[B]}{\Pr[A]} \tag{17.4.3}$$

Proof

We have

$$\Pr[B \mid A] \cdot \Pr[A] = \Pr[A \cap B] = \Pr[A \mid B] \cdot \Pr[B]$$

definition of conditional probability. Dividing by $\Pr[A]$ gives (17.4.3). ■

Philosphy of Probability

Let's try to assign a probability to the event

$$[2^{6972607} - 1 \text{ is a prime number}]$$

It's not obvious how to check whether such a large number is prime, so you might try an estimation based on the density of primes. The Prime Number Theorem implies that only about 1 in 5 million numbers in this range are prime, so you might say that the probability is about $2 \cdot 10^{-8}$. On the other hand, given that we chose this example to make some philosophical point, you might guess that we probably purposely chose an obscure looking prime number, and you might be willing to make an even money bet that the number is prime. In other words, you might think the probability is $1/2$. Finally, we can take the position that assigning a probability to this statement is nonsense because there is no randomness involved; the number is either prime or it isn't. This is the view we take in this text.

An alternate view is the *Bayesian* approach, in which a probability is interpreted as a *degree of belief* in a proposition. A Bayesian would agree that the number above is either prime or composite, but they would be perfectly willing to assign a probability to each possibility. The Bayesian approach is very broad in its willingness to assign probabilities to any event, but the problem is that there is no single "right" probability for an event, since the probability depends on one's initial beliefs. On the other hand, if you have

confidence in some set of initial beliefs, then Bayesianism provides a convincing framework for updating your beliefs as further information emerges.

As an aside, it is not clear whether Bayes himself was Bayesian in this sense. However, a Bayesian would be willing to talk about the probability that Bayes was Bayesian.

Another school of thought says that probabilities can only be meaningfully applied to *repeatable processes* like rolling dice or flipping coins. In this *frequentist* view, the probability of an event represents the fraction of trials in which the event occurred. So we can make sense of the *a posteriori* probabilities of the Cleague hockey example of Section 17.4.5 by imagining that many hockey series were played, and the probability that the local team won their first game, given that they won the series, is simply the fraction of series where they won the first game among all the series they won.

Getting back to prime numbers, we mentioned in Section 8.5.1 that there is a probabilistic primality test. If a number N is composite, there is at least a $3/4$ chance that the test will discover this. In the remaining $1/4$ of the time, the test is inconclusive. But as long as the result is inconclusive, the test can be run independently again and again up to, say, 1000 times. So if N actually is composite, then the probability that 1000 repetitions of the probabilistic test do not discover this is at most:

$$\left(\frac{1}{4}\right)^{1000}$$

If the test remained inconclusive after 1000 repetitions, it is still logically possible that N is composite, but betting that N is prime would be the best bet you'll ever get to make! If you're comfortable using probability to describe your personal belief about primality after such an experiment, you are being a Bayesian. A frequentist would not assign a probability to N's primality, but they would also be happy to bet on primality with tremendous confidence. We'll examine this issue again when we discuss polling and confidence levels in Section 19.5.

Despite the philosophical divide, the real world conclusions Bayesians and Frequentists reach from probabilities are pretty much the same, and even where their interpretations differ, they use the same theory of probability.

[1]The statistics in this example are roughly based on actual medical data, but have been rounded or simplified for illustrative purposes.

This page titled 17.4: Why Tree Diagrams Work is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

17.5: The Law of Total Probability

Breaking a probability calculation into cases simplifies many problems. The idea is to calculate the probability of an event A by splitting into two cases based on whether or not another event E occurs. That is, calculate the probability of $A \cap E$ and $A \cap \overline{E}$. By the Sum Rule, the sum of these probabilities equals $\Pr[A]$. Expressing the intersection probabilities as conditional probabilities yields:

Rule 17.5.1 (Law of Total Probability: single event).

$$\Pr[A] = \Pr[A \mid E] \cdot \Pr[E] + \Pr[A \mid \overline{E}] \cdot \Pr[\overline{E}].$$

For example, suppose we conduct the following experiment. First, we flip a fair coin. If heads comes up, then we roll one die and take the result. If tails comes up, then we roll two dice and take the sum of the two results. What is the probability that this process yields a 2? Let E be the event that the coin comes up heads, and let A be the event that we get a 2 overall. Assuming that the coin is fair, $\Pr[E] = \Pr[\overline{E}] = 1/2$. There are now two cases. If we flip heads, then we roll a 2 on a single die with probability $\Pr[A \mid E] = 1/6$. On the other hand, if we flip tails, then we get a sum of 2 on two dice with probability $\Pr[A \mid \overline{E}] = 1/36$. Therefore, the probability that the whole process yields a 2 is

$$\Pr[A] = \frac{1}{2} \cdot \frac{1}{6} + \frac{1}{2} \cdot \frac{1}{36} = \frac{7}{72}.$$

This rule extends to any set of disjoint events that make up the entire sample space. For example,

Rule (Law of Total Probability: 3-events). *If E_1, E_2, and E_3 are disjoint and $\Pr[E_1 \cup E_2 \cup E_3] = 1$, then*

$$\Pr[A] = \Pr[A \mid E_1] \cdot \Pr[E_1] + \Pr[A \mid E_2] \cdot \Pr[E_2] + \Pr[A \mid E_3] \cdot \Pr[E_3].$$

This in turn leads to a three-event version of Bayes' Rule in which the probability of event E_1 given A is calculated from the "inverse" conditional probabilities of A given E_1, E_2, and E_3:

Rule (Bayes' Rule: 3-events).

$$\Pr(E_1 \mid A) = \frac{\Pr[A \mid E_1] \cdot \Pr[E_1]}{\Pr[A \mid E_1] \cdot \Pr[E_1] + \Pr[A \mid E_2] \cdot \Pr[E_2] + \Pr[A \mid E_3] \cdot \Pr[E_3]}.$$

The generalization of these rules to n disjoint events is a routine exercise (Problems 17.3 and 17.4).

Conditioning on a Single Event

The probability rules that we derived in Section 16.5.2 extend to probabilities conditioned on the same event. For example, the Inclusion-Exclusion formula for two sets holds when all probabilities are conditioned on an event C:

$$\Pr[A \cup B \mid C] = \Pr[A \mid C] + \Pr[B \mid C] - \Pr[A \cap B \mid C].$$

This is easy to verify by plugging in the Definition 17.2.1 of conditional probability.[2]

It is important not to mix up events before and after the conditioning bar. For example, the following is *not* a valid identity:

False Claim.

$$\Pr[A \mid B \cup C] = \Pr[A \mid B] + \Pr[A \mid C] - \Pr[A \mid B \cap C]. \tag{17.5.1}$$

A simple counter-example is to let B and C be events over a uniform space with most of their outcomes in A, but not overlapping. This ensures that $\Pr[A \mid B]$ and $\Pr[A \mid C]$ are both close to 1. For example,

$$\begin{aligned} B &::= [0..9], \\ C &::= [10..18] \cup \{0\}, \\ A &::= [1..18], \end{aligned} \tag{17.5.2}$$

so

$$\Pr[A \mid B] = \frac{9}{10} = \Pr[A \mid C].$$

Also, since 0 is the only outcome in $B \cap C$ and $0 \notin A$, we have

$$\Pr[A \mid B \cap C] = 0$$

So the right hand side of (17.5.1) is 1.8, while the left hand side is a probability which can be at most 1—actually, it is $18/19$.

[2]Problem 17.14 explains why this and similar conditional identities follow on general principles from the corresponding unconditional identities.

This page titled 17.5: The Law of Total Probability is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

17.6: Simpson's Paradox

In 1973, a famous university was investigated for gender discrimination[5]. The investigation was prompted by evidence that, at first glance, appeared definitive: in 1973, 44% of male applicants to the school's graduate programs were accepted, but only 35% of female applicants were admitted.

However, this data turned out to be completely misleading. Analysis of the individual departments, showed not only that few showed significant evidence of bias, but also that among the few departments that *did* show statistical irregularities, most were slanted *in favor of women*. This suggests that if there was any sex discrimination, then it was against men!

Given the discrepancy in these findings, it feels like someone must be doing bad math—intentionally or otherwise. But the numbers are not actually inconsistent. In fact, this statistical hiccup is common enough to merit its own name: *Simpson's Paradox* occurs when multiple small groups of data all exhibit a similar trend, but that trend reverses when those groups are aggregated. To explain how this is possible, let's first clarify the problem by expressing both arguments in terms of conditional probabilities. For simplicity, suppose that there are only two departments, EE and CS. Consider the experiment where we pick a random candidate. Define the following events:

- A ::= the candidate is admitted to his or her program of choice,
- F_{EE} ::= the candidate is a woman applying to the EE department,
- F_{CS} ::= the candidate is a woman applying to the CS department,
- M_{EE} ::= the candidate is a man applying to the EE department,
- M_{CS} ::= the candidate is a man applying to the CS department.

Illegal pream-token (=)

Table 17.1 A scenario in which men are overall more likely than women to be admitted to a school, despite being less likely to be admitted into any given program.

Assume that all candidates are either men or women, and that no candidate belongs to both departments. That is, the events F_{EE}, F_{CS}, M_{EE}, and M_{CS} are all disjoint.

In these terms, the plaintiff is making the following argument:

$$\Pr[A \mid M_{EE} \cup M_{CS}] > \Pr[A \mid F_{EE} \cup F_{CS}].$$

In plain English, across the university, the total probability that a woman candidate is admitted is less than the probability for a man.

The university retorts that *in any given department*, a woman candidate has chances *equal to or greater* than those of a male candidate; more formally, that

$$\begin{aligned} \Pr[A \mid M_{EE}] &\leq \Pr[A \mid F_{EE}] \quad \text{and} \\ \Pr[A \mid M_{CS}] &\leq \Pr[A \mid F_{CS}]. \end{aligned} \tag{17.6.1}$$

It is easy to believe that these two positions are contradictory . But Table 17.1 shows a set of admission statistics for which the assertions of both the plaintiff and the university hold. In this case, a higher percentage of female applicants were admitted to each department, but overall a higher percentage of males were accepted! So the apparently contradictory claims can in fact both be true. How can we make sense of this seemingly paradoxical situation?

Initially, we and the plaintiffs both assumed that the overall admissions statistics for the university could only be explained by discrimination. However, the department-by-department breakdown shows that the source of the discrepancy is that the CS department lets in about 20% fewer candidates overall, but attracts a far larger number of woman applicants than the more permissive EE department[3]. This leads us to the conclusion that the admissions gap in not due to any systematic bias on the school's part.

But suppose we replaced "the candidate is a man/woman applying to the EE department," by "the candidate is a man/woman for whom an admissions decision was made during an odd-numbered day of the month," and likewise with CS and an even-numbered day of the month. Since we don't think the parity of a date is a cause for the outcome of an admission decision, we would most

likely dismiss the “coincidence” that on both odd and even dates, women are more frequently admitted. Instead we would judge, based on the overall data showing women less likely to be admitted, that gender bias against women *was* an issue in the university.

Bear in mind that it would be the *same numerical data* that we would be using to justify our different conclusions in the department-by-department case and the even-day-odd-day case. We interpreted the same numbers differently based on our implicit causal beliefs, specifically that departments matter and date parity does not. It is circular to claim that the data corroborated our beliefs that there is or is not discrimination. Rather, our interpretation of the data correlation depended on our beliefs about the causes of admission in the first place.[4] This example highlights a basic principle in statistics which people constantly ignore: *never assume that correlation implies causation*.

[3]At the actual university in the lawsuit, the “exclusive” departments more popular among women were those that did not require a mathematical foundation, such as English and education. Women’s disproportionate choice of these careers reflects gender bias, but one which predates the university’s involvement.

[4]These issues are thoughtfully examined in *Causality: Models, Reasoning and Inference*, Judea Pearl, Cambridge U. Press, 2001.

This page titled 17.6: Simpson’s Paradox is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

17.7: Independence

Suppose that we flip two fair coins simultaneously on opposite sides of a room. Intuitively, the way one coin lands does not affect the way the other coin lands. The mathematical concept that captures this intuition is called *independence*.

Definition 17.7.1

An event with probability 0 is defined to be independent of every event (including itself). If $\Pr[B] \neq 0$, then event A is independent of event B iff

$$\Pr[A \mid B] = \Pr[A]. \tag{17.7.1}$$

In other words, A and B are independent if knowing that B happens does not alter the probability that A happens, as is the case with flipping two coins on opposite sides of a room.

Potential Pitfall

Students sometimes get the idea that disjoint events are independent. The *opposite* is true: if $A \cap B = \emptyset$, then knowing that A happens means you know that B does not happen. Disjoint events are *never* independent—unless one of them has probability zero.

Alternative Formulation

Sometimes it is useful to express independence in an alternate form which follows immediately from Definition 17.7.1:

Theorem 17.7.2

A is independent of B if and only if

$$\Pr[A \cap B] = \Pr[A] \cdot \Pr[B]. \tag{17.7.2}$$

Notice that Theorem 17.7.2 makes apparent the symmetry between A being independent of B and B being independent of A:

Corollary 17.7.3. A is independent of B iff B is independent of A.

Independence Is an Assumption

Generally, independence is something that you *assume* in modeling a phenomenon. For example, consider the experiment of flipping two fair coins. Let A be the event that the first coin comes up heads, and let B be the event that the second coin is heads. If we assume that A and B are independent, then the probability that both coins come up heads is:

$$\Pr[A \cap B] = \Pr[A] \cdot \Pr[B] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}.$$

In this example, the assumption of independence is reasonable. The result of one coin toss should have negligible impact on the outcome of the other coin toss. And if we were to repeat the experiment many times, we would be likely to have $A \cap B$ about $1/4$ of the time.

On the other hand, there are many examples of events where assuming independence isn't justified. For example, an hourly weather forecast for a clear day might list a 10% chance of rain every hour from noon to midnight, meaning each hour has a 90% chance of being dry. But that does *not* imply that the odds of a rainless day are a mere $0.9^{12} \approx 0.28$. In reality, if it doesn't rain as of 5pm, the odds are higher than 90% that it will stay dry at 6pm as well—and if it starts pouring at 5pm, the chances are much higher than 10% that it will still be rainy an hour later.

Deciding when to *assume* that events are independent is a tricky business. In practice, there are strong motivations to assume independence since many useful formulas (such as equation (17.7.2)) only hold if the events are independent. But you need to be careful: we'll describe several famous examples where (false) assumptions of independence led to trouble. This problem gets even trickier when there are more than two events in play.

This page titled 17.7: Independence is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

https://eng.libretexts.org/@go/page/48428

17.8: Mutual Independence

We have defined what it means for two events to be independent. What if there are more than two events? For example, how can we say that the flips of n coins are all independent of one another? A set of events is said to be *mutually independent* if the probability of each event in the set is the same no matter which of the other events has occurred. This is equivalent to saying that for any selection of two or more of the events, the probability that all the selected events occur equals the product of the probabilities of the selected events.

For example, four events E_1, E_2, E_3, E_4 are mutually independent if and only if all of the following equations hold:

$$\begin{aligned}
\Pr[E_1 \cap E_2] &= \Pr[E_1] \cdot \Pr[E_2] \\
\Pr[E_1 \cap E_3] &= \Pr[E_1] \cdot \Pr[E_3] \\
\Pr[E_1 \cap E_4] &= \Pr[E_1] \cdot \Pr[E_4] \\
\Pr[E_2 \cap E_3] &= \Pr[E_2] \cdot \Pr[E_3] \\
\Pr[E_2 \cap E_4] &= \Pr[E_2] \cdot \Pr[E_4] \\
\Pr[E_3 \cap E_4] &= \Pr[E_3] \cdot \Pr[E_4] \\
\Pr[E_1 \cap E_2 \cap E_3] &= \Pr[E_1] \cdot \Pr[E_2] \cdot \Pr[E_3] \\
\Pr[E_1 \cap E_2 \cap E_4] &= \Pr[E_1] \cdot \Pr[E_2] \cdot \Pr[E_4] \\
\Pr[E_1 \cap E_3 \cap E_4] &= \Pr[E_1] \cdot \Pr[E_3] \cdot \Pr[E_4] \\
\Pr[E_2 \cap E_3 \cap E_4] &= \Pr[E_2] \cdot \Pr[E_3] \cdot \Pr[E_4] \\
\Pr[E_1 \cap E_2 \cap E_3 \cap E_4] &= \Pr[E_1] \cdot \Pr[E_2] \cdot \Pr[E_3] \cdot \Pr[E_4]
\end{aligned} \tag{17.8.1}$$

The generalization to mutual independence of n events should now be clear.

DNA Testing

Assumptions about independence are routinely made in practice. Frequently, such assumptions are quite reasonable. Sometimes, however, the reasonableness of an independence assumption is not so clear, and the consequences of a faulty assumption can be severe.

Let's return to the O. J. Simpson murder trial. The following expert testimony was given on May 15, 1995:

Mr. Clarke: When you make these estimations of frequency—and I believe you touched a little bit on a concept called independence?

Dr. Cotton: Yes, I did.

Mr. Clarke: And what is that again?

Dr. Cotton: It means whether or not you inherit one allele that you have is not— does not affect the second allele that you might get. That is, if you inherit a band at 5,000 base pairs, that doesn't mean you'll automatically or with some probability inherit one at 6,000. What you inherit from one parent is what you inherit from the other.

Mr. Clarke: Why is that important?

Dr. Cotton: Mathematically that's important because if that were not the case, it would be improper to multiply the frequencies between the different genetic locations.

Mr. Clarke: How do you—well, first of all, are these markers independent that you've described in your testing in this case?

Presumably, this dialogue was as confusing to you as it was for the jury. Essentially, the jury was told that genetic markers in blood found at the crime scene matched Simpson's. Furthermore, they were told that the probability that the markers would be found in a randomly-selected person was at most 1 in 170 million. This astronomical figure was derived from statistics such as:

- 1 person in 100 has marker A.
- 1 person in 50 marker B.
- 1 person in 40 has marker C.
- 1 person in 5 has marker D.
- 1 person in 170 has marker E.

Then these numbers were multiplied to give the probability that a randomly-selected person would have all five markers:

$$\begin{aligned}\Pr[A \cap B \cap C \cap D \cap E] &= \Pr[A] \cdot \Pr[B] \cdot \Pr[C] \cdot \Pr[D] \cdot \Pr[E] \\ &= \frac{1}{100} \cdot \frac{1}{50} \cdot \frac{1}{40} \cdot \frac{1}{5} \cdot \frac{1}{170} = \frac{1}{170,000,000}.\end{aligned} \tag{17.8.2}$$

The defense pointed out that this assumes that the markers appear mutually independently. Furthermore, all the statistics were based on just a few hundred blood samples.

After the trial, the jury was widely mocked for failing to "understand" the DNA evidence. If you were a juror, would *you* accept the 1 in 170 million calculation?

Pairwise Independence

The definition of mutual independence seems awfully complicated—there are so many selections of events to consider! Here's an example that illustrates the subtlety of independence when more than two events are involved. Suppose that we flip three fair, mutually-independent coins. Define the following events:

- A_1 is the event that coin 1 matches coin 2.
- A_2 is the event that coin 2 matches coin 3.
- A_3 is the event that coin 3 matches coin 1.

Are A_1, A_2, A_3 mutually independent?

The sample space for this experiment is:

$$\{HHH, HHT, HTH, HTT, THH, THT, TTH, TTT\}.$$

Every outcome has probability $(1/2)^3 = 1/8$ by our assumption that the coins are mutually independent.

To see if events A_1, A_2, and A_3 are mutually independent, we must check a sequence of equalities. It will be helpful first to compute the probability of each event A_i:

$$\begin{aligned}\Pr[A_1] &= \Pr[HHH] + \Pr[HHT] + \Pr[TTH] + \Pr[TTT] \\ &= \frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = \frac{1}{2}\end{aligned} \tag{17.8.3}$$

By symmetry, $\Pr[A_2] = \Pr[A_3] = 1/2$ as well. Now we can begin checking all the equalities required for mutual independence:

$$\begin{aligned}\Pr[A_1 \cap A_2] &= \Pr[HHH] + \Pr[TTT] = \frac{1}{8} + \frac{1}{8} = \frac{1}{4} = \frac{1}{2} \cdot \frac{1}{2} \\ &= \Pr[A_1]\Pr[A_2].\end{aligned} \tag{17.8.4}$$

By symmetry, $\Pr[A_1 \cap A_3] = \Pr[A_1] \cdot \Pr[A_3]$ and $\Pr[A_2 \cap A_3] = \Pr[A_2] \cdot \Pr[A_3]$ must hold also. Finally, we must check one last condition:

$$\begin{aligned}\Pr[A_1 \cap A_2 \cap A_3] &= \Pr[HHH] + \Pr[TTT] = \frac{1}{8} + \frac{1}{8} = \frac{1}{4} \\ &\neq \frac{1}{8} = \Pr[A_1]\Pr[A_2]\Pr[A_3].\end{aligned} \tag{17.8.5}$$

The three events A_1, A_2, and A_3 are not mutually independent even though any two of them are independent! This not-quite mutual independence seems weird at first, but it happens. It even generalizes:

Definition 17.8.1

A set $A_1, A_2, \ldots$, of events is *k-way independent* iff every set of k of these events is mutually independent. The set is *pairwise independent* iff it is 2-way independent.

So the events A_1, A_2, A_3 above are pairwise independent, but not mutually independent. Pairwise independence is a much weaker property than mutual independence.

For example, suppose that the prosecutors in the O. J. Simpson trial were wrong and markers A, B, C, D, and E appear only *pairwise* independently. Then the probability that a randomly-selected person has all five markers is no more than:

$$\begin{aligned}\Pr[A \cap B \cap C \cap D \cap E] &\leq \Pr[A \cap E] = \Pr[A]\Pr[E] \\ &= \frac{1}{100} \cdot \frac{1}{170} = \frac{1}{17,000}.\end{aligned} \tag{17.8.6}$$

The first line uses the fact that $\Pr[A \cap B \cap C \cap D \cap E]$ is a subset of $\Pr[A \cap E]$ (We picked out the A and E markers because they're the rarest.) We use pairwise independence on the second line. Now the probability of a random match is 1 in 17,000—a far cry from 1 in 170 million! And this is the strongest conclusion we can reach assuming only pairwise independence.

On the other hand, the 1 in 17,000 bound that we get by assuming pairwise independence is a lot better than the bound that we would have if there were no independence at all. For example, if the markers are dependent, then it is possible that

everyone with marker E has marker A,

everyone with marker A has marker B,

everyone with marker B has marker C, and

everyone with marker C has marker D.

In such a scenario, the probability of a match is

$$\Pr[E] = \frac{1}{170}.$$

So a stronger independence assumption leads to a smaller bound on the probability of a match. The trick is to figure out what independence assumption is reasonable. Assuming that the markers are *mutually* independent may well *not* be reasonable unless you have examined hundreds of millions of blood samples. Otherwise, how would you know that marker D does not show up more frequently whenever the other four markers are simultaneously present?

This page titled 17.8: Mutual Independence is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

18: Random Variables

Thus far, we have focused on probabilities of events. For example, we computed the probability that you win the Monty Hall game or that you have a rare medical condition given that you tested positive. But, in many cases we would like to know more. For example, *how many* contestants must play the Monty Hall game until one of them finally wins? *How long* will this condition last? *How much* will I lose gambling with strange dice all night? To answer such questions, we need to work with random variables.

18.1: Random Variable Examples

18.2: Independence

18.3: Distribution Functions

18.4: Great Expectations

18.5: Linearity of Expectation

This page titled 18: Random Variables is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

18.1: Random Variable Examples

Definition 18.1.1

A *random variable* R on a probability space is a total function whose domain is the sample space.

The codomain of R can be anything, but will usually be a subset of the real numbers. Notice that the name "random variable" is a misnomer; random variables are actually functions.

For example, suppose we toss three independent, unbiased coins. Let C be the number of heads that appear. Let $M = 1$ if the three coins come up all heads or all tails, and let $M = 0$ otherwise. Now every outcome of the three coin flips uniquely determines the values of C and M. For example, if we flip heads, tails, heads, then $C = 2$ and $M = 0$. If we flip tails, tails, tails, then $C = 0$ and $M = 1$. In effect, C counts the number of heads, and M indicates whether all the coins match.

Since each outcome uniquely determines C and M, we can regard them as functions mapping outcomes to numbers. For this experiment, the sample space is:

$$S = \{HHH, HHT, HTH, HTT, THH, THT, TTH, TTT\}.$$

Now C is a function that maps each outcome in the sample space to a number as follows:

$$\begin{array}{ll} C(HHH) = 3 & C(THH) = 2 \\ C(HHT) = 2 & C(THT) = 1 \\ C(HTH) = 2 & C(TTH) = 1 \\ C(HTT) = 1 & C(TTT) = 0. \end{array} \tag{18.1.1}$$

Similarly, M is a function mapping each outcome another way:

$$\begin{array}{ll} M(HHH) = 1 & M(THH) = 0 \\ M(HHT) = 0 & M(THT) = 0 \\ M(HTH) = 0 & M(TTH) = 0 \\ M(HTT) = 0 & M(TTT) = 1. \end{array} \tag{18.1.2}$$

So C and M are random variables.

Indicator Random Variables

An *indicator random variable* is a random variable that maps every outcome to either 0 or 1. Indicator random variables are also called *Bernoulli variables*. The random variable M is an example. If all three coins match, then $M = 1$; otherwise, $M = 0$.

Indicator random variables are closely related to events. In particular, an indicator random variable partitions the sample space into those outcomes mapped to 1 and those outcomes mapped to 0. For example, the indicator M partitions the sample space into two blocks as follows:

$$\underbrace{HHH \quad TTT}_{M=1} \quad \underbrace{HHT \quad HTH \quad HTT \quad THH \quad THT \quad TTH}_{M=0}.$$

In the same way, an event E partitions the sample space into those outcomes in E and those not in E. So E is naturally associated with an indicator random variable, I_E, where $I_E(\omega) = 1$ for outcomes $\omega \in E$ and $I_E(\omega) = 0$ for outcomes $\omega \notin E$. Thus, $M = I_E$ where E is the event that all three coins match.

Random Variables and Events

There is a strong relationship between events and more general random variables as well. A random variable that takes on several values partitions the sample space into several blocks. For example, C partitions the sample space as follows:

$$\underbrace{TTT}_{C=0} \quad \underbrace{TTH \quad THT \quad HTT}_{C=1} \quad \underbrace{THH \quad HTH \quad HHT}_{C=2} \quad \underbrace{HHH}_{C=3}.$$

Each block is a subset of the sample space and is therefore an event. So the assertion that $C = 2$ defines the event

$$[C = 2] = \{THH, HTH, HHT\},$$

and this event has probability

$$\Pr[C = 2] = \Pr[THH] + \Pr[HTH] + \Pr[HHT] = \frac{1}{8} + \frac{1}{8} + \frac{1}{8} = 3/8.$$

Likewise $[M = 1]$ is the event $\{TTT, HHH\}$ and has probability $1/4$.

More generally, any assertion about the values of random variables defines an event. For example, the assertion that $C \leq 1$ defines

$$[C \leq 1] = \{TTT, TTH, THT, HTT\},$$

and so $\Pr[C \leq 1] = 1/2$.

Another example is the assertion that $C \cdot M$ is an odd number. If you think about it for a minute, you'll realize that this is an obscure way of saying that all three coins came up heads, namely,

$$[C \cdot M \text{ is odd}] = \{HHH\}.$$

This page titled 18.1: Random Variable Examples is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

18.2: Independence

The notion of independence carries over from events to random variables as well. Random variables R_1 and R_2 are *independent* iff for all x_1, x_2, the two events

$$[R_1 = x_1] \text{ and } [R_2 = x_2]$$

are independent.

For example, are C and M independent? Intuitively, the answer should be "no." The number of heads, C, completely determines whether all three coins match; that is, whether $M = 1$. But, to verify this intuition, we must find some $x_1, x_2 \in \mathbb{R}$ such that:

$$\Pr[C = x_1 \text{ AND } M = x_2] \neq \Pr[C = x_1] \cdot \Pr[M = x_2].$$

One appropriate choice of values is $x_1 = 2$ and $x_2 = 1$. In this case, we have:

$$\Pr[C = 2 \text{ AND } M = 1] = 0 \neq \frac{1}{4} \cdot \frac{3}{8} = \Pr[M = 1] \cdot \Pr[C = 2].$$

The first probability is zero because we never have exactly two heads ($C = 2$) when all three coins match ($M = 1$). The other two probabilities were computed earlier.

On the other hand, let H_1 be the indicator variable for the event that the first flip is a Head, so

$$[H_1 = 1] = \{HHH, HTH, HHT, HTT\}. \tag{18.2.1}$$

Then H_1 is independent of M, since

$$\Pr[M = 1] = 1/4 = \Pr[M = 1 \mid H_1 = 1] = \Pr[M = 1 \mid H_1 = 0]$$

$$\Pr[M = 0] = 3/4 = \Pr[M = 0 \mid H_1 = 1] = \Pr[M = 0 \mid H_1 = 0]$$

This example is an instance of:

Lemma 18.2.1. *Two events are independent iff their indicator variables are independent.*

The simple proof is left to Problem 18.1.

Intuitively, the independence of two random variables means that knowing some information about one variable doesn't provide any information about the other one. We can formalize what "some information" about a variable R is by defining it to be the value of some quantity that depends on R. This intuitive property of independence then simply means that functions of independent variables are also independent:

Lemma 18.2.2. *Let R and S be independent random variables, and f and g be functions such that $\text{domain}(f) = \text{codomain}(R)$ and $\text{domain}(f) = \text{codomain}(R)$. Then $f(R)$ and $g(S)$ are independent random variables.*

The proof is another simple exercise left to Problem 18.30.

As with events, the notion of independence generalizes to more than two random variables.

Definition 18.2.3

Random variables $R_1, R_2, \ldots, R_n$ are *mutually independent* iff for all $x_1, x_2, \ldots, x_n$, the n events

$$[R_1 = x_1], [R_2 = x_2], \ldots, [R_n = x_n]$$

are mutually independent. They are *k-way independent* iff every subset of k of them are mutually independent.

Lemmas 18.2.1 and 18.2.2 both extend straightforwardly to k-way independent variables.

This page titled 18.2: Independence is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

18.3: Distribution Functions

A random variable maps outcomes to values. The probability density function, $\mathbf{PDF}_R(x)$, of a random variable, R, measures the probability that R takes the value x, and the closely related cumulative distribution function, $\mathbf{CDF}_R(x)$, measures the probability that $R \leq x$. Random variables that show up for different spaces of outcomes often wind up behaving in much the same way because they have the same probability of taking different values, that is, because they have the same pdf/cdf.

Definition 18.3.1

Let R be a random variable with codomain V. The *probability density function* of R is a function $\mathbf{PDF}_R : V \to [0,1]$ defined by:

$$\mathbf{PDF}_R(x) ::= \begin{cases} \Pr[R = x] & \text{if } x \in \text{ range}(R), \\ 0 & \text{if } x \notin \text{ range}(R). \end{cases}$$

If the codomain is a subset of the real numbers, then the *cumulative distribution function* is the function $\mathbf{CDF}_R : \mathbb{R} \to [0,1]$ defined by:

$$\mathbf{CDF}_R(x) ::= \Pr[R \leq x].$$

A consequence of this definition is that

$$\sum_{x \in \text{ range}(R)} \mathbf{PDF}_R(x) = 1.$$

This is because R has a value for each outcome, so summing the probabilities over all outcomes is the same as summing over the probabilities of each value in the range of R.

As an example, suppose that you roll two unbiased, independent, 6-sided dice. Let T be the random variable that equals the sum of the two rolls. This random variable takes on values in the set $V = \{2, 3, \ldots, 12\}$. A plot of the probability density function for T is shown in Figure 18.1. The lump in the middle indicates that sums close to 7 are the most likely. The total area of all the rectangles is 1 since the dice must take on exactly one of the sums in $V = \{2, 3, \ldots, 12\}$.

The cumulative distribution function for T is shown in Figure 18.2: The height of the ith bar in the cumulative distribution function is equal to the *sum* of the heights of the leftmost i bars in the probability density function. This follows from the definitions of pdf and cdf:

$$\mathbf{CDF}_R(x) = \Pr[R \leq x] = \sum_{y \leq x} \Pr[R = y] = \sum_{y \leq x} \mathbf{PDF}_R(y).$$

It also follows from the definition that

$$\lim_{x \to \infty} \mathbf{CDF}_R(x) = 1 \text{ and } \lim_{x \to -\infty} \mathbf{CDF}_R(x) = 0.$$

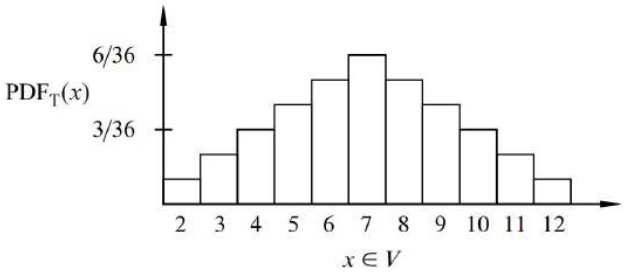


Figure 18.1 The probability density function for the sum of two 6-sided dice.

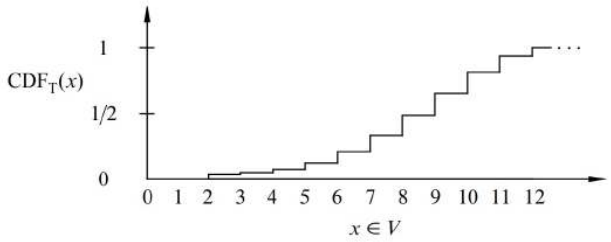


Figure 18.2 The cumulative distribution function for the sum of two 6-sided dice.

Both $\mathbf{PDF}_R$ and $\mathbf{CDF}_R$ capture the same information about R, so take your choice. The key point here is that neither the probability density function nor the cumulative distribution function involves the sample space of an experiment.

One of the really interesting things about density functions and distribution functions is that many random variables turn out to have the *same* pdf and cdf. In other words, even though R and S are different random variables on different probability spaces, it is often the case that

$$\mathbf{PDF}_R = \mathbf{PDF}_S.$$

In fact, some pdf's are so common that they are given special names. For example, the three most important distributions in computer science are the *Bernoulli distribution*, the *uniform distribution*, and the *binomial distribution*. We look more closely at these common distributions in the next several sections.

Bernoulli Distributions

A Bernoulli distribution is the distribution function for a Bernoulli variable. Specifically, the *Bernoulli distribution* has a probability density function of the form $f_p : \{0,1\} \to [0,1]$ where

$$\begin{aligned} f_p(0) &= p, \text{ and} \\ f_p(1) &= 1-p, \end{aligned} \tag{18.3.1}$$

for some $p \in [0,1]$. The corresponding cumulative distribution function is $F_p : \mathbb{R} \to [0,1]$ where

$$F_p(x) ::= \begin{cases} 0 & \text{if } x < 0 \\ p & \text{if } 0 \le x \le 1 \\ 1 & \text{if } 1 \le x \end{cases}$$

Uniform Distributions

A random variable that takes on each possible value in its codomain with the same probability is said to be *uniform*. If the codomain V has n elements, then the *uniform distribution* has a pdf of the form

$$f : V \to [0,1]$$

where

$$f(v) = \frac{1}{n}$$

for all $v \in V$.

If the elements of V in increasing order are $a_1, a_2, \ldots, a_n$, then the cumulative distribution function would be $F : \mathbb{R} \to [0,1]$ where

$$F(x) ::= \begin{cases} 0 & \text{if } x < a_1 \\ k/n & \text{if } a_k \le x \le a_{k+1} \text{ for } 1 \le k < n \\ 1 & \text{if } a_n \le x. \end{cases}$$

Uniform distributions come up all the time. For example, the number rolled on a fair die is uniform on the set $\{1,2,\ldots,6\}$. An indicator variable is uniform when its pdf is $f_{1/2}$.

The Numbers Game

Enough definitions—let's play a game! We have two envelopes. Each contains an integer in the range $0, 1, \ldots, 100$, and the numbers are distinct. To win the game, you must determine which envelope contains the larger number. To give you a fighting chance, we'll let you peek at the number in one envelope selected at random. Can you devise a strategy that gives you a better than 50% chance of winning?

For example, you could just pick an envelope at random and guess that it contains the larger number. But this strategy wins only 50% of the time. Your challenge is to do better.

So you might try to be more clever. Suppose you peek in one envelope and see the number 12. Since 12 is a small number, you might guess that the number in the other envelope is larger. But perhaps we've been tricky and put small numbers in *both* envelopes. Then your guess might not be so good!

An important point here is that the numbers in the envelopes may *not* be random. We're picking the numbers and we're choosing them in a way that we think will defeat your guessing strategy. We'll only use randomization to choose the numbers if that serves our purpose: making you lose!

Intuition Behind the Winning Strategy

People are surprised when they first learn that there is a strategy that wins more than 50% of the time, regardless of what numbers we put in the envelopes.

Suppose that you somehow knew a number x that was in between the numbers in the envelopes. Now you peek in one envelope and see a number. If it is bigger than x, then you know you're peeking at the higher number. If it is smaller than x, then you're peeking at the lower number. In other words, if you know a number x between the numbers in the envelopes, then you are certain to win the game.

The only flaw with this brilliant strategy is that you do *not* know such an x. This sounds like a dead end, but there's a cool way to salvage things: try to *guess* x! There is some probability that you guess correctly. In this case, you win 100% of the time. On the other hand, if you guess incorrectly, then you're no worse off than before; your chance of winning is still 50%. Combining these two cases, your overall chance of winning is better than 50%.

Many intuitive arguments about probability are wrong despite sounding persuasive. But this one goes the other way: it may not convince you, but it's actually correct. To justify this, we'll go over the argument in a more rigorous way—and while we're at it, work out the optimal way to play.

Analysis of the Winning Strategy

For generality, suppose that we can choose numbers from the integer interval $[0..n]$. Call the lower number L and the higher number H.

Your goal is to guess a number x between L and H. It's simplest if x does not equal L or H, so you should select x at random from among the half-integers:

$$\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \cdots, \frac{2n-1}{2}$$

But what probability distribution should you use?

The uniform distribution—selecting each of these half-integers with equal probability— turns out to be your best bet. An informal justification is that if we figured out that you were unlikely to pick some number—say $50\frac{1}{2}$—then we'd always put 50 and 51 in the envelopes. Then you'd be unlikely to pick an x between L and H and would have less chance of winning.

After you've selected the number x, you peek into an envelope and see some number T. If $T > x$, then you guess that you're looking at the larger number. If $T < x$, then you guess that the other number is larger.

All that remains is to determine the probability that this strategy succeeds. We can do this with the usual four step method and a tree diagram.

Step 1: Find the sample space.

You either choose x too low ($< L$), too high ($> H$), or just right ($L < x < H$). Then you either peek at the lower number ($T = L$) or the higher number ($T = H$). This gives a total of six possible outcomes, as show in Figure 18.3.

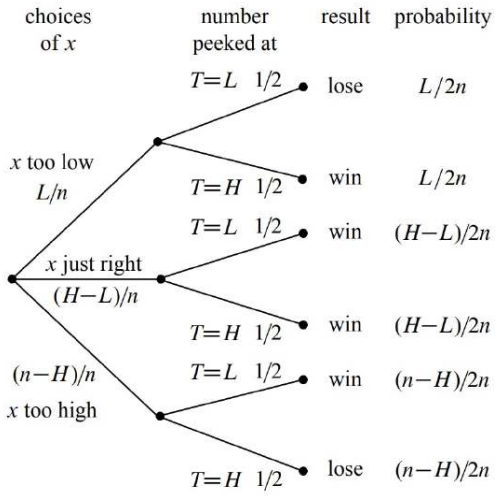


Figure 18.3 The tree diagram for the numbers game.

Step 2: Define events of interest.

The four outcomes in the event that you win are marked in the tree diagram.

Step 3: Assign outcome probabilities.

First, we assign edge probabilities. Your guess x is too low with probability L/n, too high with probability $(n - H)/n$, and just right with probability $(H - L)/n$. Next, you peek at either the lower or higher number with equal probability. Multiplying along root-to-leaf paths gives the outcome probabilities.

Step 4: Compute event probabilities.

The probability of the event that you win is the sum of the probabilities of the four outcomes in that event:

$$\begin{aligned} \Pr[\text{win}] &= \frac{L}{2n} + \frac{H-L}{2n} + \frac{H-L}{2n} + \frac{n-H}{2n} \\ &= \frac{1}{2} + \frac{H-L}{2n} \\ &\geq \frac{1}{2} + \frac{1}{2n} \end{aligned} \tag{18.3.2}$$

The final inequality relies on the fact that the higher number H is at least 1 greater than the lower number L since they are required to be distinct.

Sure enough, you win with this strategy more than half the time, regardless of the numbers in the envelopes! So with numbers chosen from the range $0, 1, \ldots, 100$, you win with probability at least $1/2 + 1/200 = 50.5\%$. If instead we agree to stick to numbers $0, 1, \ldots, 10$, then your probability of winning rises to 55%. By Las Vegas standards, those are great odds.

Randomized Algorithms

The best strategy to win the numbers game is an example of a *randomized algorithm*—it uses random numbers to influence decisions. Protocols and algorithms that make use of random numbers are very important in computer science. There are many problems for which the best known solutions are based on a random number generator.

For example, the most commonly-used protocol for deciding when to send a broadcast on a shared bus or Ethernet is a randomized algorithm known as *exponential backoff*. One of the most commonly-used sorting algorithms used in practice, called quicksort, uses random numbers. You'll see many more examples if you take an algorithms course. In each case, randomness is used to improve the probability that the algorithm runs quickly or otherwise performs well.

Binomial Distributions

The third commonly-used distribution in computer science is the *binomial distribution*. The standard example of a random variable with a binomial distribution is the number of heads that come up in n independent flips of a coin. If the coin is fair, then the number of heads has an *unbiased binomial distribution*, specified by the pdf $f_n : [0..n] \to [0,1]$:

$$f_n(k) ::= \binom{n}{k} 2^{-n}.$$

This is because there are $\binom{n}{k}$ sequences of n coin tosses with exactly k heads, and each such sequence has probability 2^{-n}.

A plot of $f_{20}(k)$ is wn in Figure 18.4. The most likely outcome is $k = 10$ heads, and the probability falls off rapidly for larger and smaller values of k. The falloff regions to the left and right of the main hump are called the *tails of the distribution*.

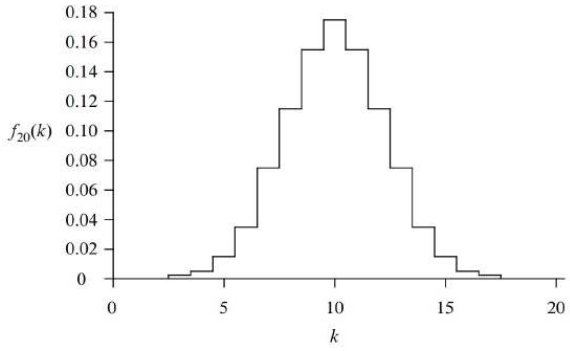


Figure 18.4 The pdf for the unbiased binomial distribution for $n = 20, f_{20}(k)$.

In many fields, including Computer Science, probability analyses come down to getting small bounds on the tails of the binomial distribution. In the context of a problem, this typically means that there is very small probability that something *bad* happens, which could be a server or communication link overloading or a randomized algorithm running for an exceptionally long time or producing the wrong result.

The tails do get small very fast. For example, the probability of flipping at most 25 heads in 100 tosses is less than 1 in 3,000,000. In fact, the tail of the distribution falls off so rapidly that the probability of flipping exactly 25 heads is nearly twice the probability of flipping exactly 24 heads *plus* the probability of flipping exactly 23 heads *plus* ... the probability of flipping no heads.

The General Binomial Distribution

If the coins are biased so that each coin is heads with probability p, then the number of heads has a *general binomial density function* specified by the pdf $f_{n,p} : [0..n] \to [0,1]$ where

$$f_{n,p}(k) = \binom{n}{k} p^k (1-p)^{n-k}. \tag{18.3.3}$$

for some $n \in \mathbb{N}^+$ and $p \in [0,1]$. This is because there are $\binom{n}{k}$ sequences with k heads and $n-k$ tails, but now $p^k(1-p)^{n-k}$ is the probability of each such sequence.

For example, the plot in Figure 18.5 shows the probability density function $f_{n,p}(k)$ corresponding to flipping $n = 20$ independent coins that are heads with probability $p = 0.75$. The graph shows that we are most likely to get $k = 15$ heads, as you might expect. Once again, the probability falls off quickly for larger and smaller values of k.

https://eng.libretexts.org/@go/page/48433

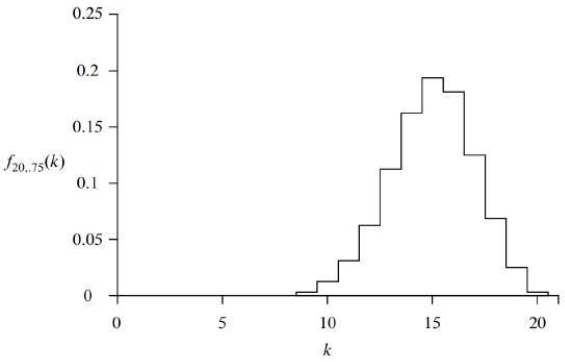


Figure 18.5 The pdf for the general binomial distribution $f_{n,p}(k)$ for $n = 20$ and $p = .75$.

This page titled 18.3: Distribution Functions is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

18.4: Great Expectations

The *expectation* or *expected value* of a random variable is a single number that reveals a lot about the behavior of the variable. The expectation of a random variable is also known as its *mean* or *average*. For example, the first thing you typically want to know when you see your grade on an exam is the average score of the class. This average score turns out to be precisely the expectation of the random variable equal to the score of a random student.

More precisely, the expectation of a random variable is its "average" value when each value is weighted according to its probability. Formally, the expected value of a random variable is defined as follows:

Definition 18.4.1

If R is a random variable defined on a sample space S, then the expectation of R is

$$\mathbf{Ex}[R] ::= \sum_{\omega \in S} R(\omega)\Pr[\omega]. \tag{18.4.1}$$

Let's work through some examples.

The Expected Value of a Uniform Random Variable

Rolling a 6-sided die provides an example of a uniform random variable. Let R be the value that comes up when you roll a fair 6-sided die. Then by (18.4.1), the expected value of R is

$$\mathbf{Ex}[R] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = \frac{7}{2}.$$

This calculation shows that the name "expected" value is a little misleading; the random variable might *never* actually take on that value. No one expects to roll a $3\frac{1}{2}$ on an ordinary die!

In general, if R_n is a random variable with a uniform distribution on $\{a_1, a_2, \ldots, a_n\}$, then the expectation of R_n is simply the average of the a_i's:

$$\mathbf{Ex}[R_n] = \frac{a_1 + a_2 + \cdots + a_n}{n}.$$

The Expected Value of a Reciprocal Random Variable

Define a random variable S to be the reciprocal of the value that comes up when you roll a fair 6-sided die. That is, $S = 1/R$ where R is the value that you roll. Now,

$$\mathbf{Ex}[S] = \mathbf{Ex}\left[\frac{1}{R}\right] = \frac{1}{1} \cdot \frac{1}{6} + \frac{1}{2} \cdot \frac{1}{6} + \frac{1}{3} \cdot \frac{1}{6} + \frac{1}{4} \cdot \frac{1}{6} + \frac{1}{5} \cdot \frac{1}{6} + \frac{1}{6} \cdot \frac{1}{6} = \frac{49}{120}.$$

Notice that

$$\mathbf{Ex}[1/R] \neq 1/\mathbf{Ex}[R].$$

The Expected Value of an Indicator Random Variable

The expected value of an indicator random variable for an event is just the probability of that event.

Lemma 18.4.2. *If I_A is the indicator random variable for event A, then*

$$\mathbf{Ex}[I_A] = \Pr[A].$$

Proof.

$$\begin{aligned} \mathbf{Ex}[I_A] &= 1 \cdot \Pr[I_A = 1] + 0 \cdot \Pr[I_A = 0] = \Pr[I_A = 1] \\ &= \Pr[A]. \qquad \text{(def of } I_A\text{)} \end{aligned} \tag{18.4.2}$$

For example, if A is the event that a coin with bias p comes up heads, then $\mathbf{Ex}[I_A] = \Pr[I_A = 1] = p$.

Alternate Definition of Expectation

There is another standard way to define expectation.

Theorem 18.4.3

For any random variable R,

$$\mathbf{Ex}[R] = \sum_{x \in \text{ range}(R)} x \cdot \Pr[R = x]. \tag{18.4.3}$$

The proof of Theorem 18.4.3, like many of the elementary proofs about expectation in this chapter, follows by regrouping of terms in equation (18.4.1):

Proof

Suppose R is defined on a sample space S. Then,

$$\begin{aligned}
\mathbf{Ex}[R] &::= \sum_{\omega \in S} R(\omega)\Pr[\omega] \\
&= \sum_{x \in \text{ range}(R)} \sum_{x \in [R=x]} R(\omega)\Pr[\omega] \\
&= \sum_{x \in \text{ range}(R)} \sum_{x \in [R=x]} x\Pr[\omega] && \text{(def of the event } [R = x]) \\
&= \sum_{x \in \text{ range}(R)} x \left(\sum_{x \in [R=x]} \Pr[\omega] \right) && \text{(factoring } x \text{ from the inner sum)} \\
&= \sum_{x \in \text{ range}(R)} x \cdot \Pr[R = x]. && \text{(def of } \Pr[R = x])
\end{aligned} \tag{18.4.4}$$

The first equality follows because the events $[R = x]$ for $x \in \text{range}(R)$ partition the sample space S, so summing over the outcomes in $[R = x]$ for $x \in \text{range}(R)$ is the same as summing over S. ■

In general, equation (18.4.3) is more useful than the defining equation (18.4.1) for calculating expected values. It also has the advantage that it does not depend on the sample space, but only on the density function of the random variable. On the other hand, summing over all outcomes as in equation (18.4.1) sometimes yields easier proofs about general properties of expectation.

Conditional Expectation

Just like event probabilities, expectations can be conditioned on some event. Given a random variable R, the expected value of R conditioned on an event A is the probability-weighted average value of R over outcomes in A. More formally:

Definition 18.4.4

The *conditional expectation* $\mathbf{Ex}[R \mid A]$ of a random variable R given event A is:

$$\mathbf{Ex}[R \mid A] ::= \sum_{r \in \text{ range}(R)} r \cdot \Pr[R = r \mid A]. \tag{18.4.5}$$

For example, we can compute the expected value of a roll of a fair die, given that the number rolled is at least 4. We do this by letting R be the outcome of a roll of the die. Then by equation (18.4.5),

$$\mathbf{Ex}[R \mid R \geq 4] = \sum_{i=1}^{6} i \cdot \Pr[R = i \mid R \geq 4] = 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot \frac{1}{3} + 5 \cdot \frac{1}{3} + 6 \cdot \frac{1}{3} = 5.$$

Conditional expectation is useful in dividing complicated expectation calculations into simpler cases. We can find a desired expectation by calculating the conditional expectation in each simple case and averaging them, weighing each case by its probability.

For example, suppose that 49.6% of the people in the world are male and the rest female—which is more or less true. Also suppose the expected height of a randomly chosen male is $5'11''$, while the expected height of a randomly chosen female is $5'5''$ What is the

expected height of a randomly chosen person? We can calculate this by averaging the heights of men and women. Namely, let H be the height (in feet) of a randomly chosen person, and let M be the event that the person is male and F the event that the person is female. Then

$$\begin{aligned}\mathbf{Ex}[H] &= \mathbf{Ex}[H \mid M]\Pr[M] + \mathbf{Ex}[H \mid F]\Pr[F] \\ &= (5 + 11/12) \cdot 0.496 + (5 + 5/12) \cdot (1 - 0.496) \\ &= 5.6646\cdots\end{aligned} \tag{18.4.6}$$

which is a little less than $5'8''$.

This method is justified by:

Theorem 18.4.5

(Law of Total Expectation). *Let R be a random variable on a sample space S, and suppose that $A_1, A_2, \ldots$, is a partition of S. Then*

$$\mathbf{Ex}[R] = \sum_i \mathbf{Ex}[R \mid A_i]\Pr[A_i].$$

Proof

$$\begin{aligned}\mathbf{Ex}[R] &= \sum_{r \in \text{range}(R)} r \cdot \Pr[R = r] && \text{(by 18.4.2)} \\ &= \sum_r r \cdot \sum_i \Pr[R = r \mid A_i]\Pr[A_i] && \text{(Law of Total Probability)} \\ &= \sum_r \sum_i r \cdot \Pr[R = r \mid A_i]\Pr[A_i] && \text{(distribute constant } r\text{)} \\ &= \sum_i \sum_r r \cdot \Pr[R = r \mid A_i]\Pr[A_i] && \text{(exchange order of summation)} \\ &= \sum_i \Pr[A_i] \sum_r r \cdot \Pr[R = r \mid A_i] && \text{(factor constant } \Pr[A_i]\text{)} \\ &= \sum_i \Pr[A_i]\mathbf{Ex}[R \mid A_i]. && \text{(Def 18.4.4 of cond. expectation)}\end{aligned} \tag{18.4.7}$$

■

Mean Time to Failure

A computer program crashes at the end of each hour of use with probability p, if it has not crashed already. What is the expected time until the program crashes? This will be easy to figure out using the Law of Total Expectation, Theorem 18.4.5. Specifically, we want to find $\mathbf{Ex}[C]$ where C is the number of hours until the first crash. We'll do this by conditioning on whether or not the crash occurs in the first hour.

So define A to be the event that the system fails on the first step and $\overline{A}$ to be the complementary event that the system does not fail on the first step. Then the mean time to failure $\mathbf{Ex}[C]$ is

$$\mathbf{Ex}[C] = \mathbf{Ex}[C \mid A]\Pr[A] + \mathbf{Ex}[C \mid \overline{A}]\Pr[\overline{A}]. \tag{18.4.8}$$

Since A is the condition that the system crashes on the first step, we know that

$$\mathbf{Ex}[C \mid A] = 1. \tag{18.4.9}$$

Since $\overline{A}$ is the condition that the system does *not* crash on the first step, conditioning on $\overline{A}$ is equivalent to taking a first step without failure and then starting over without conditioning. Hence,

$$\mathbf{Ex}[C \mid \overline{A}] = 1 + \mathbf{Ex}[C]. \tag{18.4.10}$$

Plugging (18.4.9) and (18.4.10) into (18.4.8):

$$\begin{aligned}\mathbf{Ex}[C] &= 1 \cdot p + (1 + \mathbf{Ex}[C])(1 - p) \\ &= p + 1 - p + (1 - p)\mathbf{Ex}[C] \\ &= 1 + (1 - p)\mathbf{Ex}[C].\end{aligned} \tag{18.4.11}$$

Then, rearranging terms gives

$$1 = \mathbf{Ex}[C] - (1-p)\mathbf{Ex}[C] = p\mathbf{Ex}[C],$$

and thus

$$\mathbf{Ex}[C] = 1/p.$$

The general principle here is well-worth remembering.

Mean Time to Failure

If a system independently fails at each time step with probability p, then the expected number of steps up to the first failure is $1/p$.

So, for example, if there is a 1% chance that the program crashes at the end of each hour, then the expected time until the program crashes is $1/0.01 = 100$ hours.

As a further example, suppose a couple insists on having children until they get a boy, then how many baby girls should they expect before their first boy? Assume for simplicity that there is a 50% chance that a child will be a boy and that the genders of siblings are mutually independent.

This is really a variant of the previous problem. The question, "How many hours until the program crashes?" is mathematically the same as the question, "How many children must the couple have until they get a boy?" In this case, a crash corresponds to having a boy, so we should set $p = 1/2$. By the preceding analysis, the couple should expect a baby boy after having $1/p = 2$ children. Since the last of these will be a boy, they should expect just one girl. So even in societies where couples pursue this commitment to boys, the expected population will divide evenly between boys and girls.

There is a simple intuitive argument that explains the mean time to failure formula (18.4.12). Suppose the system is restarted after each failure. This makes the mean time to failure the same as the mean time between successive repeated failures. Now if the probability of failure at a given step is p, then after n steps we expect to have pn failures. Now, by definition, the average number of steps between failures is equal to np/p, namely, $1/p$.

For the record, we'll state a formal version of this result. A random variable like C that counts steps to first failure is said to have a *geometric distribution* with parameter p.

Definition 18.4.6

A random variable, C, has a *geometric distribution* with parameter p iff $\mathbf{codomain}(C) = \mathbb{Z}^+$ and

$$\mathbf{Pr}[C = i] = (1-p)^{i-1}p.$$

Lemma 18.4.7. *If a random variable C has a geometric distribution with parameter p, then*

$$\mathbf{Ex}[C] = \frac{1}{p}. \tag{18.4.12}$$

Expected Returns in Gambling Games

Some of the most interesting examples of expectation can be explained in terms of gambling games. For straightforward games where you win w dollars with probability p and you lose x dollars with probability $1-p$, it is easy to compute your *expected return* or *winnings*. It is simply

$$pw - (1-p)x \text{ dollars.}$$

For example, if you are flipping a fair coin and you win $1 for heads and you lose $1 for tails, then your expected winnings are

$$\frac{1}{2}\cdot 1 - \left(1 - \frac{1}{2}\right)\cdot 1 = 0.$$

In such cases, the game is said to be *fair* since your expected return is zero.

Splitting the Pot

We'll now look at a different game which is fair—but only on first analysis.

It's late on a Friday night in your neighborhood hangout when two new biker dudes, Eric and Nick, stroll over and propose a simple wager. Each player will put $2 on the bar and secretly write "heads" or "tails" on their napkin. Then you will flip a fair coin. The $6 on the bar will then be "split"—that is, be divided equally—among the players who correctly predicted the outcome of the coin toss. Pot splitting like this is a familiar feature in poker games, betting pools, and lotteries.

This sounds like a fair game, but after your regrettable encounter with strange dice (Section 16.3), you are definitely skeptical about gambling with bikers. So before agreeing to play, you go through the four-step method and write out the tree diagram to compute your expected return. The tree diagram is shown in Figure 18.6.

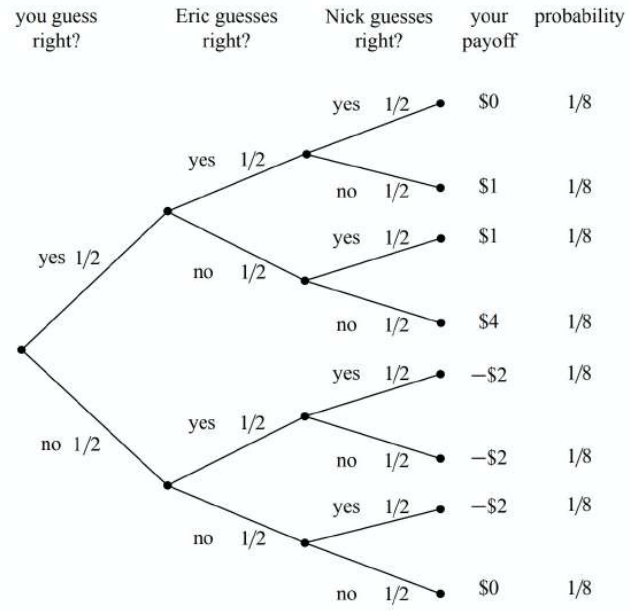


Figure 18.6 The tree diagram for the game where three players each wager $2 and then guess the outcome of a fair coin toss. The winners split the pot.

The "payoff" values in Figure 18.6 are computed by dividing the $6 pot[1] among those players who guessed correctly and then subtracting the $2 that you put into the pot at the beginning. For example, if all three players guessed correctly, then your payoff is $0, since you just get back your $2 wager. If you and Nick guess correctly and Eric guessed wrong, then your payoff is

$$\frac{6}{2} - 2 = 1.$$

In the case that everyone is wrong, you all agree to split the pot and so, again, your payoff is zero.

To compute your expected return, you use equation (18.4.3):

$$\begin{aligned} \text{Ex[playoff]} &= 0 \cdot \frac{1}{8} + 1 \cdot \frac{1}{8} + 1 \cdot \frac{1}{8} + 4 \cdot \frac{1}{8} + (-2) \cdot \frac{1}{8} + (-2) \cdot \frac{1}{8} + (-2) \cdot \frac{1}{8} + 0 \cdot \frac{1}{8} \\ &= 0. \end{aligned} \tag{18.4.13}$$

This confirms that the game is fair. So, for old time's sake, you break your solemn vow to never ever engage in strange gambling games.

The Impact of Collusion

Needless to say, things are not turning out well for you. The more times you play the game, the more money you seem to be losing. After 1000 wagers, you have lost over $500. As Nick and Eric are consoling you on your "bad luck," you do a back-of-the-envelope calculation and decide that the probability of losing $500 in 1000 fair $2 wagers is very, very small.

Now it is possible of course that you are very, very unlucky. But it is more likely that something fishy is going on. Somehow the tree diagram in Figure 18.6 is not a good model of the game.

The "something" that's fishy is the opportunity that Nick and Eric have to collude against you. The fact that the coin flip is fair certainly means that each of Nick and Eric can only guess the outcome of the coin toss with probability **1/2**. But when you look

back at the previous 1000 bets, you notice that Eric and Nick never made the same guess. In other words, Nick always guessed "tails" when Eric guessed "heads," and vice-versa. Modelling this fact now results in a slightly different tree diagram, as shown in Figure 18.7.

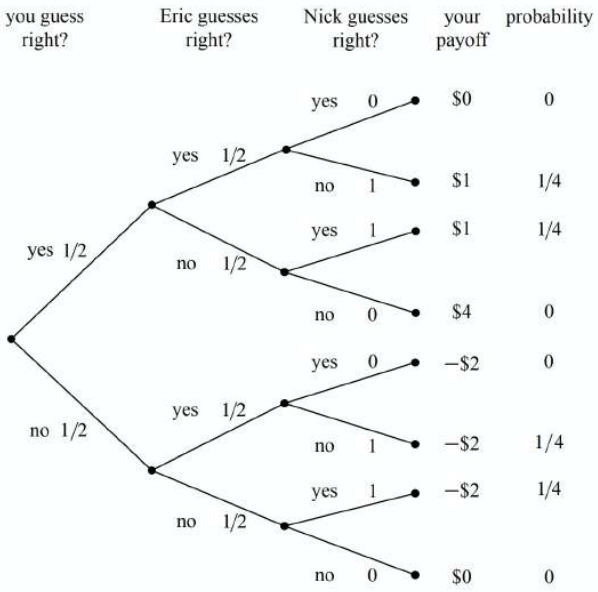


Figure 18.7 The revised tree diagram reflecting the scenario where Nick always guesses the opposite of Eric.

The payoffs for each outcome are the same in Figures 18.6 and 18.7, but the probabilities of the outcomes are different. For example, it is no longer possible for all three players to guess correctly, since Nick and Eric are always guessing differently. More importantly, the outcome where your payoff is \$4 is also no longer possible. Since Nick and Eric are always guessing differently, one of them will always get a share of the pot. As you might imagine, this is not good for you!

When we use equation (18.4.3) to compute your expected return in the collusion scenario, we find that

$$\begin{aligned}\text{Ex[playoff]} &= 0\cdot 0+1\cdot\frac{1}{4}+1\cdot\frac{1}{4}+4\cdot 0+(-2)\cdot 0+(-2)\cdot\frac{1}{4}+(-2)\cdot\frac{1}{4}+0\cdot- \\ &= -\frac{1}{2}.\end{aligned} \tag{18.4.14}$$

So watch out for these biker dudes! By colluding, Nick and Eric have made it so that you expect to lose \$.50 every time you play. No wonder you lost \$500 over the course of 1000 wagers.

How to Win the Lottery

Similar opportunities to collude arise in many betting games. For example, consider the typical weekly football betting pool, where each participant wagers \$10 and the participants that pick the most games correctly split a large pot. The pool seems fair if you think of it as in Figure 18.6. But, in fact, if two or more players collude by guessing differently, they can get an "unfair" advantage at your expense!

In some cases, the collusion is inadvertent and you can profit from it. For example, many years ago, a former MIT Professor of Mathematics named Herman Chernoff figured out a way to make money by playing the state lottery. This was surprising since the state usually takes a large share of the wagers before paying the winners, and so the expected return from a lottery ticket is typically pretty poor. So how did Chernoff find a way to make money? It turned out to be easy!

In a typical state lottery,

- all players pay \$1 to play and select 4 numbers from 1 to 36,
- the state draws 4 numbers from 1 to 36 uniformly at random,
- the states divides **1/2** of the money collected among the people who guessed correctly and spends the other half redecorating the governor's residence.

This is a lot like the game you played with Nick and Eric, except that there are more players and more choices. Chernoff discovered that a small set of numbers was selected by a large fraction of the population. Apparently many people think the same way; they pick the same numbers not on purpose as in the previous game with Nick and Eric, but based on the Red Sox winning average or today's date. The result is as though the players were intentionally colluding to lose. If any one of them guessed correctly, then they'd have to split the pot with many other players. By selecting numbers uniformly at random, Chernoff was unlikely to get one of these favored sequences. So if he won, he'd likely get the whole pot! By analyzing actual state lottery data, he determined that he could win an average of 7 cents on the dollar. In other words, his expected return was not $-\$.50$ as you might think, but $+\$.07$. [2] Inadvertent collusion often arises in betting pools and is a phenomenon that you can take advantage of.

[1]The money invested in a wager is commonly referred to as the *pot*.

[2]Most lotteries now offer randomized tickets to help smooth out the distribution of selected sequences.

This page titled 18.4: Great Expectations is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

18.5: Linearity of Expectation

Expected values obey a simple, very helpful rule called Linearity of Expectation. Its simplest form says that the expected value of a sum of random variables is the sum of the expected values of the variables.

Theorem 18.5.1

For any random variables R_1 and R_2,

$$\mathbf{Ex}[R_1 + R_2] = \mathbf{Ex}[R_1] + \mathbf{Ex}[R_2].$$

Proof

Let $T ::= R_1 + R_2$. The proof follows straightforwardly by rearranging terms in equation (18.4.1) in the definition of expectation:

$$\begin{aligned} \mathbf{Ex}[T] ::= \sum_{\omega \in S} T(\omega) \cdot \Pr[\omega] & \\ = \sum_{\omega \in S} (R_1(\omega) + R_2(\omega)) \cdot \Pr[\omega] & \quad \text{(def of } T\text{)} \\ = \sum_{\omega \in S} R_1(\omega) \cdot \Pr[\omega] + \sum_{\omega \in S} R_2(\omega) \cdot \Pr[\omega] & \quad \text{(rearranging terms)} \\ = \mathbf{Ex}[R_1] + \mathbf{Ex}[R_2]. & \quad \text{(by (18.4.1))} \quad \blacksquare \end{aligned} \tag{18.5.1}$$

A small extension of this proof, which we leave to the reader, implies

Theorem 18.5.2

For any random variables R_1, R_2 and constants $a_1, a_2 \in \mathbb{R}$,

$$\mathbf{Ex}[a_1 R_1 + a_2 R_2] = a_1 \mathbf{Ex}[R_1] + a_2 \mathbf{Ex}[R_2].$$

In other words, expectation is a linear function. A routine induction extends the result to more than two variables:

Corollary 18.5.3 (Linearity of Expectation). *For any random variables $R_1, \ldots, R_k$ and constants $a_1, \ldots, a_k \in \mathbb{R}$,*

$$\mathbf{Ex}\left[\sum_{i=1}^{k} a_i R_i\right] = \sum_{i=1}^{k} a_i \mathbf{Ex}[R_i].$$

The great thing about linearity of expectation is that *no independence is required*. This is really useful, because dealing with independence is a pain, and we often need to work with random variables that are not known to be independent.

As an example, let's compute the expected value of the sum of two fair dice.

Expected Value of Two Dice

What is the expected value of the sum of two fair dice?

Let the random variable R_1 be the number on the first die, and let R_2 be the number on the second die. We observed earlier that the expected value of one die is 3.5. We can find the expected value of the sum using linearity of expectation:

$$\mathbf{Ex}[R_1 + R_2] = \mathbf{Ex}[R_1] + \mathbf{Ex}[R_2] = 3.5 + 3.5 = 7.$$

Assuming that the dice were independent, we could use a tree diagram to prove that this expected sum is 7, but this would be a bother since there are 36 cases. And without assuming independence, it's not apparent how to apply the tree diagram approach at all. But notice that we did *not* have to assume that the two dice were independent. The expected sum of two dice is 7—even if they are controlled to act together in some way—as long as each individual controlled die remains fair.

Sums of Indicator Random Variables

Linearity of expectation is especially useful when you have a sum of indicator random variables. As an example, suppose there is a dinner party where n men check their hats. The hats are mixed up during dinner, so that afterward each man receives a random hat. In particular, each man gets his own hat with probability $1/n$. What is the expected number of men who get their own hat?

Letting G be the number of men that get their own hat, we want to find the expectation of G. But all we know about G is that the probability that a man gets his own hat back is $1/n$. There are many different probability distributions of hat permutations with this property, so we don't know enough about the distribution of G to calculate its expectation directly using equation (18.4.1) or (18.4.2). But linearity of expectation lets us sidestep this issue.

We'll use a standard, useful trick to apply linearity, namely, we'll express G as a sum of indicator variables. In particular, let G_i be an indicator for the event that the ith man gets his own hat. That is, $G_i = 1$ if the ith man gets his own hat, and $G_i = 0$ otherwise. The number of men that get their own hat is then the sum of these indicator random variables:

$$G = G_1 + G_2 + \cdots + G_n. \tag{18.5.2}$$

These indicator variables are not mutually independent. For example, if $n-1$ men all get their own hats, then the last man is certain to receive his own hat. But again, we don't need to worry about this dependence, since linearity holds regardless.

Since G_i is an indicator random variable, we know from Lemma 18.4.2 that

$$\mathrm{Ex}[G_i] = \Pr[G_i = 1] = 1/n. \tag{18.5.3}$$

By Linearity of Expectation and equation (18.5.2), this means that

$$\begin{aligned} \mathrm{Ex}[G] &= \mathrm{Ex}[G_1 + G_2 + \cdots + G_n] \\ &= \mathrm{Ex}[G_1] + \mathrm{Ex}[G_2] + \cdots + \mathrm{Ex}[G_n] \\ &= \overbrace{\frac{1}{n} + \frac{1}{n} + \cdots + \frac{1}{n}}^{n} \\ &= 1. \end{aligned} \tag{18.5.4}$$

So even though we don't know much about how hats are scrambled, we've figured out that on average, just one man gets his own hat back, regardless of the number of men with hats!

More generally, Linearity of Expectation provides a very good method for computing the expected number of events that will happen.

Theorem 18.5.4

Given any collection of events $A_1, A_2, \ldots, A_n$, the expected number of events that will occur is

$$\sum_{i=1}^{n} \Pr[A_i].$$

For example, A_i could be the event that the ith man gets the right hat back. But in general, it could be any subset of the sample space, and we are asking for the expected number of events that will contain a random sample point.

Proof

Define R_i to be the indicator random variable for A_i, where $R_i(\omega) = 1$ if $w \in A_i$ and $R_i(\omega) = 0$ if $w \notin A_i$. Let $R = R_1 + R_2 + \cdots + R_n$. Then

$$\begin{aligned} \mathrm{Ex}[R] &= \sum_{i=1}^{n} \mathrm{Ex}[R_i] && \text{(by Linearity of Expectation)} \\ &= \sum_{i=1}^{n} \Pr[R_i = 1] && \text{(by Lemma 18.4.2)} \\ &= \sum_{i=1}^{n} \Pr[A_i]. && \text{(def of indicator variable)} \end{aligned} \tag{18.5.5}$$

So whenever you are asked for the expected number of events that occur, all you have to do is sum the probabilities that each event occurs. Independence is not needed.

Expectation of a Binomial Distribution

Suppose that we independently flip n biased coins, each with probability p of coming up heads. What is the expected number of heads?

Let J be the random variable denoting the number of heads. Then J has a binomial distribution with parameters n, p, and

$$\Pr[J = k] = \binom{n}{k} p^k (1-p)^{n-k}.$$

Applying equation (18.4.2), this means that

$$\mathrm{Ex}[J] = \sum_{k=0}^{n} k \Pr[J = k] = \sum_{k=0}^{n} k \binom{n}{k} p^k (1-p)^{n-k}. \tag{18.5.6}$$

This sum looks a tad nasty, but linearity of expectation leads to an easy derivation of a simple closed form. We just express J as a sum of indicator random variables, which is easy. Namely, let J_i be the indicator random variable for the ith coin coming up heads, that is,

$$J_i ::= \begin{cases} 1 & \text{if the } i\text{th coin is heads} \\ 0 & \text{if the } i\text{th coin is tails.} \end{cases}$$

Then the number of heads is simply

$$J = J_1 + J_2 + \cdots + J_n.$$

By Theorem 18.5.4,

$$\mathrm{Ex}[J] = \sum_{i=1}^{n} \Pr[J_i] = pn. \tag{18.5.7}$$

That really was easy. If we flip n mutually independent coins, we expect to get pn heads. Hence the expected value of a binomial distribution with parameters n and p is simply pn.

But what if the coins are not mutually independent? It doesn't matter—the answer is still pn because Linearity of Expectation and Theorem 18.5.4 do not assume any independence.

If you are not yet convinced that Linearity of Expectation and Theorem 18.5.4 are powerful tools, consider this: without even trying, we have used them to prove a complicated looking identity, namely,

$$\sum_{k=0}^{n} k \binom{n}{k} p^k (1-p)^{n-k} = pn, \tag{18.5.8}$$

which follows by combining equations (18.5.6) and (18.5.7) (see also Exercise 18.26).

The next section has an even more convincing illustration of the power of linearity to solve a challenging problem.

The Coupon Collector Problem

Every time we purchase a kid's meal at Taco Bell, we are graciously presented with a miniature "Racin' Rocket" car together with a launching device which enables us to project our new vehicle across any tabletop or smooth floor at high velocity. Truly, our delight knows no bounds.

There are different colored Racin' Rocket cars. The color of car awarded to us by the kind server at the Taco Bell register appears to be selected uniformly and independently at random. What is the expected number of kid's meals that we must purchase in order to acquire at least one of each color of Racin' Rocket car?

The same mathematical question shows up in many guises: for example, what is the expected number of people you must poll in order to find at least one person with each possible birthday? The general question is commonly called the *coupon collector*

problem after yet another interpretation.

A clever application of linearity of expectation leads to a simple solution to the coupon collector problem. Suppose there are five different colors of Racin' Rocket cars, and we receive this sequence:

$$\text{blue} \quad \text{green} \quad \text{green} \quad \text{red} \quad \text{blue} \quad \text{orange} \quad \text{blue} \quad \text{orange} \quad \text{gray}.$$

Let's partition the sequence into 5 segments:

$$\underbrace{\text{blue}}_{X_0} \quad \underbrace{\text{green}}_{X_1} \quad \underbrace{\text{green} \quad \text{red}}_{X_2} \quad \underbrace{\text{blue} \quad \text{orange}}_{X_3} \quad \underbrace{\text{blue} \quad \text{orange} \quad \text{gray}}_{X_4}.$$

The rule is that a segment ends whenever we get a new kind of car. For example, the middle segment ends when we get a red car for the first time. In this way, we can break the problem of collecting every type of car into stages. Then we can analyze each stage individually and assemble the results using linearity of expectation.

In the general case there are n colors of Racin' Rockets that we're collecting. Let X_k be the length of the kth segment. The total number of kid's meals we must purchase to get all n Racin' Rockets is the sum of the lengths of all these segments:

$$T = X_0 + X_1 + \cdots + X_{n-1}.$$

Now let's focus our attention on X_k, the length of the kth segment. At the beginning of segment k, we have k different types of car, and the segment ends when we acquire a new type. When we own k types, each kid's meal contains a type that we already have with probability k/n. Therefore, each meal contains a new type of car with probability $1 - k/n = (n-k)/n$. Thus, the expected number of meals until we get a new kind of car is $n/(n-k)$ by the Mean Time to Failure rule. This means that

$$\text{Ex}[X_k] = \frac{n}{n-k}.$$

Linearity of expectation, together with this observation, solves the coupon collector problem:

$$\begin{aligned}
\text{Ex}[T] &= \text{Ex}[X_0 + X_1 + \cdots + X_{n-1}] \\
&= \text{Ex}[X_0 + \text{Ex}[X_1] + \cdots + \text{Ex}[X_{n-1}] \\
&= \frac{n}{n-0} + \frac{n}{n-1} + \cdots + \frac{n}{3} + \frac{n}{2} + \frac{n}{1} \\
&= n\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}\right) \\
&= nH_n \\
&\sim n \ln n.
\end{aligned} \tag{18.5.9}$$

Cool! It's those Harmonic Numbers again.

We can use equation (18.5.9) to answer some concrete questions. For example, the expected number of die rolls required to see every number from 1 to 6 is:

$$6H_6 = 14.7\ldots$$

And the expected number of people you must poll to find at least one person with each possible birthday is:

$$365H_{365} = 2364.6\ldots$$

Infinite Sums

Linearity of expectation also works for an infinite number of random variables provided that the variables satisfy an absolute convergence criterion.

Theorem 18.5.5

(Linearity of Expectation). *Let $R_0, R_1, \ldots$, be random variables such that*

$$\sum_{i=0}^{\infty} \text{Ex}[|R_i|]$$

converges. Then

$$\mathbf{Ex}\left[\sum_{i=0}^{\infty} R_i\right] = \sum_{i=0}^{\infty} \mathbf{Ex}[R_i].$$

Proof

Let $T ::= \sum_{i=0}^{\infty} R_i$.

We leave it to the reader to verify that, under the given convergence hypothesis, all the sums in the following derivation are absolutely convergent, which justifies rearranging them as follows:

$$\begin{aligned}
\sum_{i=0}^{\infty} \mathbf{Ex}[R_i] &= \sum_{i=0}^{\infty} \sum_{s \in S} R_i(s) \cdot \Pr[s] && \text{(Def. 18.4.1)} \\
&= \sum_{s \in S} \sum_{i=0}^{\infty} R_i(s) \cdot \Pr[s] && \text{(exchanging order of summation)} \\
&= \sum_{s \in S} \left[\sum_{i=0}^{\infty} R_i(s)\right] \cdot \Pr[s] && \text{(factoring out } \Pr[s]\text{)} \\
&= \sum_{s \in S} T(s) \cdot \Pr[s] && \text{(Def. of T)} \\
&= \mathbf{Ex}[T] && \text{(Def. 18.4.1)} \\
&= \mathbf{Ex}\left[\sum_{i=0}^{\infty} R_i\right]. && \text{(Def. of T)}
\end{aligned} \tag{18.5.10}$$

■

Gambling Paradox

One of the simplest casino bets is on “red” or “black” at the roulette table. In each play at roulette, a small ball is set spinning around a roulette wheel until it lands in a red, black, or green colored slot. The payoff for a bet on red or black matches the bet; for example, if you bet $10 on red and the ball lands in a red slot, you get back your original $10 bet plus another matching $10.

The casino gets its advantage from the green slots, which make the probability of both red and black each less than $1/2$. In the US, a roulette wheel has 2 green slots among 18 black and 18 red slots, so the probability of red is $18/38 \approx 0.473$. In Europe, where roulette wheels have only 1 green slot, the odds for red are a little better—that is, $18/37 \approx 0.486$—but still less than even.

Of course you can’t expect to win playing roulette, even if you had the good fortune to gamble against a fair roulette wheel. To prove this, note that with a *fair* wheel, you are equally likely win or lose each bet, so your expected win on any spin is zero. Therefore if you keep betting, your expected win is the sum of your expected wins on each bet: still zero.

Even so, gamblers regularly try to develop betting strategies to win at roulette despite the bad odds. A well known strategy of this kind is *bet doubling*, where you bet, say, $10 on red and keep doubling the bet until a red comes up. This means you stop playing if red comes up on the first spin, and you leave the casino with a $10 profit. If red does not come up, you bet $20 on the second spin. Now if the second spin comes up red, you get your $20 bet plus $20 back and again walk away with a net profit of $20 10 D $10. If red does not come up on the second spin, you next bet $40 and walk away with a net win of $40 20 10 D $10 if red comes up on on the third spin, and so on.

Since we’ve reasoned that you can’t even win against a fair wheel, this strategy against an unfair wheel shouldn’t work. But wait a minute! There is a 0.486 probability of red appearing on each spin of the wheel, so the mean time until a red occurs is less than three. What’s more, red will come up *eventually* with probability one, and as soon as it does, you leave the casino $10 ahead. In other words, by bet doubling you are certain to win $10, and so your expectation is $10, not zero!

Something’s wrong here.

Solution to the Paradox

The argument claiming the expectation is zero against a fair wheel is flawed by an implicit, invalid use of linearity of expectation for an infinite sum.

To explain this carefully, let B_n be the number of dollars you win on your nth bet, where B_n is defined to be zero if red comes up before the nth spin of the wheel. Now the dollar amount you win in any gambling session is

$$\sum_{n=1}^{\infty} B_n,$$

and your expected win is

$$\mathbf{Ex}\left[\sum_{n=1}^{\infty} B_n\right]. \tag{18.5.11}$$

Moreover, since we're assuming the wheel is fair, it's true that $\mathbf{Ex}[B_n] = 0$, so

$$\sum_{n=1}^{\infty} \mathbf{Ex}[B_n] = \sum_{n=1}^{\infty} 0 = 0. \tag{18.5.12}$$

The flaw in the argument that you can't win is the implicit appeal to linearity of expectation to conclude that the expectation (\label{18.5.7}) equals the sum of expectations in (\label{18.5.8}). This is a case where linearity of expectation fails to hold—even though the expectation (\label{18.5.7}) is 10 and the sum (\label{18.5.8}) of expectations converges. The problem is that the expectation of the sum of the absolute values of the bets diverges, so the condition required for infinite linearity fails. In particular, under bet doubling your nth bet is $10 \cdot 2^{n-1}$ dollars while the probability that you will make an nth bet is 2^{-n}. So

$$\mathbf{Ex}[|B_n|] = 10 \cdot 2^{n-1} 2^{-n} = 20.$$

Therefore the sum

$$\sum_{n=1}^{\infty} \mathbf{Ex}[|B_n|] = 20 + 20 + 20 + \cdots$$

diverges rapidly.

So the presumption that you can't beat a fair game, and the argument we offered to support this presumption, are mistaken: by bet doubling, you can be sure to walk away a winner. Probability theory has led to an apparently absurd conclusion. But probability theory shouldn't be rejected because it leads to this absurd conclusion.

If you only had a finite amount of money to bet with—say enough money to make k bets before going bankrupt—then it would be correct to calculate your expection by summing $B_1 + B_2 + \cdots + B_k$, and your expectation would be zero for the fair wheel and negative against an unfair wheel. In other words, in order to follow the bet doubling strategy, you need to have an infinite bankroll. So it's absurd to assume you could actually follow a bet doubling strategy, and it's entirely reasonable that an absurd assumption leads to an absurd conclusion.

Expectations of Products

While the expectation of a sum is the sum of the expectations, the same is usually not true for products. For example, suppose that we roll a fair 6-sided die and denote the outcome with the random variable R. Does $\mathbf{Ex}[R \cdot R] = \mathbf{Ex}[R] \cdot \mathbf{Ex}[R]$?

We know that $\mathbf{Ex}[R] = 3\frac{1}{2}$ and thus $\mathbf{Ex}[R]^2 = 12\frac{1}{4}$. Let's compute $\mathbf{Ex}[R]^2$ to see if we get the same result.

$$\begin{aligned} \mathbf{Ex}[R]^2 &= \sum_{\omega \in S} R^2(\omega) \Pr[w] = \sum_{i=1}^{6} i^2 \cdot \Pr[R_i = i] \\ &= \frac{1^2}{6} + \frac{2^2}{6} + \frac{3^2}{6} + \frac{4^2}{6} + \frac{5^2}{6} + \frac{6^2}{6} = 15\ 1/6 \neq 12\ 1/4. \end{aligned} \tag{18.5.13}$$

That is,

$$\mathbf{Ex}[R \cdot R] \neq \mathbf{Ex}[R] \cdot \mathbf{Ex}[R].$$

So the expectation of a product is not always equal to the product of the expectations.

There is a special case when such a relationship *does* hold however; namely, when the random variables in the product are *independent*.

Theorem 18.5.6

For any two independent random variables R_1, R_2,

$$\mathbf{Ex}[R_1 \cdot R_2] = \mathbf{Ex}[R_1] \cdot \mathbf{Ex}[R_2].$$

The proof follows by rearrangement of terms in the sum that defines $\mathbf{Ex}[R_1 \cdot R_2]$. Details appear in Problem 18.25.

Theorem 18.5.6 extends routinely to a collection of mutually independent variables.

Corollary 18.5.7. *[Expectation of Independent Product]*

If random variables $R_1, R_2, \ldots, R_k$ *are mutually independent, then*

$$\mathbf{Ex}\left[\prod_{i=1}^{k} R_i\right] = \prod_{i=1}^{k} \mathbf{Ex}[R_i].$$

This page titled 18.5: Linearity of Expectation is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

19: Deviation from the Mean

In the previous chapter, we took it for granted that expectation is useful and developed a bunch of techniques for calculating expected values. But why should we care about this value? After all, a random variable may never take a value anywhere near its expectation.

The most important reason to care about the mean value comes from its connection to estimation by sampling. For example, suppose we want to estimate the average age, income, family size, or other measure of a population. To do this, we determine a random process for selecting people—say, throwing darts at census lists. This process makes the selected person's age, income, and so on into a random variable whose *mean* equals the *actual average* age or income of the population. So, we can select a random sample of people and calculate the average of people in the sample to estimate the true average in the whole population. But when we make an estimate by repeated sampling, we need to know how much confidence we should have that our estimate is OK, and how large a sample is needed to reach a given confidence level. The issue is fundamental to all experimental science. Because of random errors—*noise*—repeated measurements of the same quantity rarely come out exactly the same. Determining how much confidence to put in experimental measurements is a fundamental and universal scientific issue. Technically, judging sampling or measurement accuracy reduces to finding the probability that an estimate *deviates* by a given amount from its expected value.

Another aspect of this issue comes up in engineering. When designing a sea wall, you need to know how strong to make it to withstand tsunamis for, say, at least a century. If you're assembling a computer network, you might need to know how many component failures it should tolerate to likely operate without maintenance for at least a month. If your business is insurance, you need to know how large a financial reserve to maintain to be nearly certain of paying benefits for, say, the next three decades. Technically, such questions come down to finding the probability of *extreme* deviations from the mean.

This issue of *deviation from the mean* is the focus of this chapter.

19.1: Markov's Theorem
19.2: Chebyshev's Theorem
19.3: Properties of Variance
19.4: Estimation by Random Sampling
19.5: Confidence versus Probability
19.6: Sums of Random Variables
19.7: Really Great Expectations

This page titled 19: Deviation from the Mean is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

19.1: Markov's Theorem

Markov's theorem gives a generally coarse estimate of the probability that a random variable takes a value *much larger* than its mean. It is an almost trivial result by itself, but it actually leads fairly directly to much stronger results.

The idea behind Markov's Theorem can be explained by considering the quantity known as *intelligence quotient*, IQ, which remains in wide use despite doubts about its legitimacy. IQ was devised so that its average measurement would be 100. This immediately implies that at most $1/3$ of the population can have an IQ of 300 or more, because if more than a third had an IQ of 300, then the average would have to be more than $1/3 \cdot 300 = 100$. So, the probability that a randomly chosen person has an IQ of 300 or more is at most 1/3. By the same logic, we can also conclude that at most 2/3 of the population can have an IQ of 150 or more.

Of course, these are not very strong conclusions. No IQ of over 300 has ever been recorded; and while many IQ's of over 150 have been recorded, the fraction of the population that actually has an IQ that high is very much smaller than $2/3$. But though these conclusions are weak, we reached them using just the fact that the average IQ is 100—along with another fact we took for granted, that IQ is never negative. Using only these facts, we can't derive smaller fractions, because there are nonnegative random variables with mean 100 that achieve these fractions. For example, if we choose a random variable equal to 300 with probability $1/3$ and 0 with probability $2/3$, then its mean is 100, and the probability of a value of 300 or more really is $1/3$.

Theorem 19.1.1

(Markov's Theorem). *If R is a nonnegative random variable, then for all $x > 0$*

$$\Pr[R \geq x] \leq \frac{\mathrm{Ex}[R]}{x}. \tag{19.1.1}$$

Proof

Let y vary over the range of R. Then for any $x > 0$

$$\begin{aligned} \mathrm{Ex}[R] ::= & \sum_y y \Pr[R = y] \\ & \geq \sum_{y \geq x} y \Pr[R = y] \geq \sum_{y \geq x} x \Pr[R = y] = x \sum_{y \geq x} \Pr[R = y] \\ & = x \Pr[R \geq x], \end{aligned} \tag{19.1.2}$$

where the first inequality follows from the fact that $R \geq 0$.

Dividing the first and last expressions in (19.1.2) by x gives the desired result. ■

Our focus is deviation from the mean, so it's useful to rephrase Markov's Theorem this way:

Corollary 19.1.2. *If R is a nonnegative random variable, then for all $c \geq 1$*

$$\Pr[R \geq c \cdot \mathrm{Ex}[R]] \leq \frac{1}{c}. \tag{19.1.3}$$

This Corollary follows immediately from Markov's Theorem(19.1.1) by letting x be $c \cdot \mathrm{Ex}[R]$.

Applying Markov's Theorem

Let's go back to the Hat-Check problem of Section 18.5.2. Now we ask what the probability is that x or more men get the right hat, this is, what the value of $\Pr[G \geq x]$ is.

We can compute an upper bound with Markov's Theorem. Since we know $\mathrm{Ex}[G] = 1$, Markov's Theorem implies

$$\Pr[G \geq x] \leq \frac{\mathrm{Ex}[G]}{x} = \frac{1}{x}.$$

For example, there is no better than a 20% chance that 5 men get the right hat, regardless of the number of people at the dinner party.

The Chinese Appetizer problem is similar to the Hat-Check problem. In this case, n people are eating different appetizers arranged on a circular, rotating Chinese banquet tray. Someone then spins the tray so that each person receives a random appetizer. What is the probability that everyone gets the same appetizer as before?

There are n equally likely orientations for the tray after it stops spinning. Everyone gets the right appetizer in just one of these n orientations. Therefore, the correct answer is $1/n$.

But what probability do we get from Markov's Theorem? Let the random variable, R, be the number of people that get the right appetizer. Then of course $\mathbf{Ex}[R] = 1$, so applying Markov's Theorem, we find:

$$\Pr[R \geq n] \leq \frac{\mathbf{Ex}[R]}{n} = \frac{1}{n}.$$

So for the Chinese appetizer problem, Markov's Theorem is precisely right!

Unfortunately, Markov's Theorem is not always so accurate. For example, it gives the same $1/n$ upper limit for the probability that everyone gets their own hat back in the Hat-Check problem, where the probability is actually $1/(n!)$. So for Hat-Check, Markov's Theorem gives a probability bound that is way too large.

Markov's Theorem for Bounded Variables

Suppose we learn that the average IQ among MIT students is 150 (which is not true, by the way). What can we say about the probability that an MIT student has an IQ of more than 200? Markov's theorem immediately tells us that no more than $150/200$ or $3/4$ of the students can have such a high IQ. Here, we simply applied Markov's Theorem to the random variable, R, equal to the IQ of a random MIT student to conclude:

$$\Pr[R > 200] \leq \frac{\mathbf{Ex}[R]}{200} = \frac{150}{200} = \frac{3}{4}.$$

But let's observe an additional fact (which may be true): no MIT student has an IQ less than 100. This means that if we let $T ::= R - 100$, then T is nonnegative and $\mathbf{Ex}[T] = 50$, so we can apply Markov's Theorem to T and conclude:

$$\Pr[R > 200] = \Pr[T > 100] \leq \frac{\mathbf{Ex}[T]}{100} = \frac{50}{100} = \frac{1}{2}.$$

So only half, not $3/4$, of the students can be as amazing as they think they are. A bit of a relief!

In fact, we can get better bounds applying Markov's Theorem to $R - b$ instead of R for any lower bound b on R (see Problem 19.3). Similarly, if we have any upper bound, u, on a random variable, S, then $u - S$ will be a nonnegative random variable, and applying Markov's Theorem to $u - S$ will allow us to bound the probability that S is much *less* than its expectation.

This page titled 19.1: Markov's Theorem is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

19.2: Chebyshev's Theorem

We've seen that Markov's Theorem can give a better bound when applied to $R - b$ rather than R. More generally, a good trick for getting stronger bounds on a random variable R out of Markov's Theorem is to apply the theorem to some cleverly chosen function of R. Choosing functions that are powers of the absolute value of R turns out to be especially useful. In particular, since $|R|^z$ is nonnegative for any real number z, Markov's inequality also applies to the event $[|R|^z \geq x^z]$. But for positive $x, z > 0$ this event is equivalent to the event $[|R| \geq x]$ for , so we have:

Lemma 19.2.1. *For any random variable R and positive real numbers x, z,*

$$\Pr[|R| \geq x] \leq \frac{\mathbf{Ex}[|R|^z]}{x^z}.$$

Rephrasing (19.2.1) in terms of $|R - \mathbf{Ex}[R]|$, the random variable that measures R's deviation from its mean, we get

$$\Pr[|R - \mathbf{Ex}[R]| \geq x] \leq \frac{\mathbf{Ex}[(R - \mathbf{Ex}[R])^z]}{x^z}. \tag{19.2.1}$$

The case when $z = 2$ turns out to be so important that the numerator of the right hand side of (19.2.1) has been given a name:

Definition 19.2.2

The *variance*, $\mathbf{Var}[R]$, of a random variable, R, is:

$$\mathbf{Var}[R] ::= \mathbf{Ex}[(R - \mathbf{Ex}[R])^2].$$

Variance is also known as *mean square deviation*.

The restatement of (19.2.1) for $z = 2$ is known as *Chebyshev's Theorem*[1]

Theorem 19.2.3

(Chebyshev). *Let R be a random variable and $x \in \mathbb{R}^+$. Then*

$$\Pr[|R - \mathbf{Ex}[R]| \geq x] \leq \frac{\mathbf{Var}[R]}{x^2}.$$

The expression $\mathbf{Ex}[(R - \mathbf{Ex}[R])^2]$ for variance is a bit cryptic; the best approach is to work through it from the inside out. The innermost expression, $R - \mathbf{Ex}[R]$, is precisely the deviation of R above its mean. Squaring this, we obtain, $(R - \mathbf{Ex}[R])^2$. This is a random variable that is near 0 when R is close to the mean and is a large positive number when R deviates far above or below the mean. So if R is always close to the mean, then the variance will be small. If R is often far from the mean, then the variance will be large.

Variance in Two Gambling Games

The relevance of variance is apparent when we compare the following two gambling games.

Game A: We win \$2 with probability $2/3$ and lose \$1 with probability $1/3$.

Game B: We win \$1002 with probability $2/3$ and lose \$2001 with probability $1/3$.

Which game is better financially? We have the same probability, $2/3$, of winning each game, but that does not tell the whole story. What about the expected return for each game? Let random variables A and B be the payoffs for the two games. For example, A is 2 with probability $2/3$ and -1 with probability $1/3$. We can compute the expected payoff for each game as follows:

$$\mathbf{Ex}[A] = 2 \cdot \frac{2}{3} + (-1) \cdot \frac{1}{3} = 1,$$

$$\mathbf{Ex}[B] = 1002 \cdot \frac{2}{3} + (-2001) \cdot \frac{1}{3} = 1.$$

The expected payoff is the same for both games, but the games are very different. This difference is not apparent in their expected value, but is captured by variance.

We can compute the $\mathbf{Var}[A]$ by working "from the inside out" as follows:

$$\begin{aligned} A - \mathbf{Ex}[A] &= \begin{cases} 1 & \text{with probability } \frac{2}{3} \\ -2 & \text{with probability } \frac{1}{3} \end{cases} \\ (A - \mathbf{Ex}[A])^2 &= \begin{cases} 1 & \text{with probability } \frac{2}{3} \\ 4 & \text{with probability } \frac{1}{3} \end{cases} \\ \mathbf{Ex}[(A - \mathbf{Ex}[A])^2] &= 1 \cdot \frac{2}{3} + 4 \cdot \frac{1}{3} \\ \mathbf{Var}[A] &= 2. \end{aligned} \tag{19.2.2}$$

Similarly, we have for $\mathbf{Var}[B]$:

$$\begin{aligned} B - \mathbf{Ex}[B] &= \begin{cases} 1001 & \text{with probability } \frac{2}{3} \\ -2002 & \text{with probability } \frac{1}{3} \end{cases} \\ (B - \mathbf{Ex}[B])^2 &= \begin{cases} 1,002,001 & \text{with probability } \frac{2}{3} \\ 4,008,004 & \text{with probability } \frac{1}{3} \end{cases} \\ \mathbf{Ex}[(B - \mathbf{Ex}[B])^2] &= 1,002,001 \cdot \frac{2}{3} + 4,008,004 \cdot \frac{1}{3} \\ \mathbf{Var}[B] &= 2,004,002. \end{aligned} \tag{19.2.3}$$

The variance of Game A is 2 and the variance of Game B is more than two million! Intuitively, this means that the payoff in Game A is usually close to the expected value of \$1, but the payoff in Game B can deviate very far from this expected value.

High variance is often associated with high risk. For example, in ten rounds of Game A, we expect to make \$10, but could conceivably lose \$10 instead. On the other hand, in ten rounds of game B, we also expect to make \$10, but could actually lose more than \$20,000!

Standard Deviation

In Game B above, the deviation from the mean is 1001 in one outcome and -2002 in the other. But the variance is a whopping 2,004,002. The happens because the "units" of variance are wrong: if the random variable is in dollars, then the expectation is also in dollars, but the variance is in square dollars. For this reason, people often describe random variables using *standard deviation* instead of variance.

Definition 19.2.4

The *standard deviation*, σ_R, of a random variable, R, is the square root of the variance:

$$\sigma_R ::= \sqrt{\mathbf{Var}[R]} = \sqrt{\mathbf{Ex}[(R - \mathbf{Ex}[R])^2]}.$$

So the standard deviation is the square root of the mean square deviation, or the *root mean square* for short. It has the same units—dollars in our example—as the original random variable and as the mean. Intuitively, it measures the average deviation from the mean, since we can think of the square root on the outside as canceling the square on the inside.

Example 19.2.5. The standard deviation of the payoff in Game B is:

$$\sigma_R = \sqrt{\mathbf{Var}[B]} = \sqrt{2,004,002} \approx 1416.$$

The random variable B actually deviates from the mean by either positive 1001 or negative 2002, so the standard deviation of 1416 describes this situation more closely than the value in the millions of the variance.

For bell-shaped distributions like the one illustrated in Figure 19.1, the standard deviation measures the "width" of the interval in which values are most likely to fall. This can be more clearly explained by rephrasing Chebyshev's Theorem in terms of standard deviation, which we can do by substituting $x = c\sigma_R$ in (19.1):

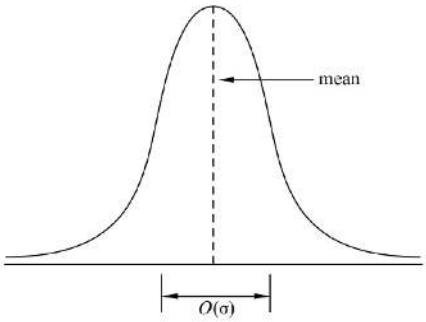


Figure 19.1 The standard deviation of a distribution indicates how wide the "main part" of it is.

Corollary 19.2.6. *Let R be a random variable, and let c be a positive real number.*

$$\Pr[|R - \mathrm{Ex}[R]| \geq c\sigma_R] \leq \frac{1}{c^2}. \tag{19.2.4}$$

Now we see explicitly how the "likely" values of R are clustered in an $O(\sigma_R)$-sized region around $\mathrm{Ex}[R]$, confirming that the standard deviation measures how spread out the distribution of R is around its mean.

The IQ Example

Suppose that, in addition to the national average IQ being 100, we also know the standard deviation of IQ's is 10. How rare is an IQ of 300 or more?

Let the random variable, R, be the IQ of a random person. So $\mathrm{Ex}[R] = 100$, $\sigma_R = 10$, and R is nonnegative. We want to compute $\Pr[R \geq 300]$.

We have already seen that Markov's Theorem 19.1.1 gives a coarse bound, namely,

$$\Pr[R \geq 300] \leq \frac{1}{3}.$$

Now we apply Chebyshev's Theorem to the same problem:

$$\Pr[R \geq 300] = \Pr[|R - 100| \geq 200] \leq \frac{\mathrm{Var}[R]}{200^2} = \frac{10^2}{200^2} = \frac{1}{400}.$$

So Chebyshev's Theorem implies that at most one person in four hundred has an IQ of 300 or more. We have gotten a much tighter bound using additional information—the variance of R—than we could get knowing only the expectation.

[1]There are Chebyshev Theorems in several other disciplines, but Theorem 19.2.3 is the only one we'll refer to.

This page titled 19.2: Chebyshev's Theorem is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

19.3: Properties of Variance

Variance is the average *of the square* of the distance from the mean. For this reason, variance is sometimes called the "mean square deviation." Then we take its square root to get the standard deviation—which in turn is called "root mean square deviation."

But why bother squaring? Why not study the actual distance from the mean, namely, the absolute value of $R - \mathbf{Ex}[R]$, instead of its root mean square? The answer is that variance and standard deviation have useful properties that make them much more important in probability theory than average absolute deviation. In this section, we'll describe some of those properties. In the next section, we'll see why these properties are important.

Formula for Variance

Applying linearity of expectation to the formula for variance yields a convenient alternative formula.

Lemma 19.3.1.

$$\mathbf{Var}[R] = \mathbf{Ex}[R^2] - \mathbf{Ex}^2[R],$$

for any random variable, R.

Here we use the notation $\mathbf{Ex}^2[R]$ as shorthand for $(\mathbf{Ex}[R])^2$.

Proof. Let $\mu = \mathbf{Ex}[R]$. Then

$$\begin{aligned}
\mathbf{Var}[R] &= \mathbf{Ex}[(R - \mathbf{Ex}[R])^2] && \text{(Def 19.2.2 of variance)} \\
&= \mathbf{Ex}[(R-\mu)^2] && \text{(def of } \mu\text{)} \\
&= \mathbf{Ex}[R^2 - 2\mu R + \mu^2] \\
&= \mathbf{Ex}[R^2] - 2\mu\mathbf{Ex}[R] + \mu^2 && \text{(linearity of expectation)} \\
&= \mathbf{Ex}[R^2] - 2\mu^2 + \mu^2 && \text{(def of } \mu\text{)} \\
&= \mathbf{Ex}[R^2] - \mu^2 \\
&= \mathbf{Ex}[R^2] - \mathbf{Ex}^2[R]. && \text{(def of } \mu\text{)}
\end{aligned} \tag{19.3.1}$$

■

A simple and very useful formula for the variance of an indicator variable is an immediate consequence.

Corollary 19.3.2. If B is a Bernoulli variable where $p ::= \Pr[B = 1]$, then

$$\mathbf{Var}[B] = p - p^2 = p(1-p). \tag{19.3.2}$$

Proof. By Lemma 18.4.2, $\mathbf{Ex}[B] = p$. But B only takes values 0 and 1, so $B^2 = B$ and equation (19.3.2) follows immediately from Lemma 19.3.1. ■

Variance of Time to Failure

According to Section 18.4.6, the mean time to failure is $1/p$ for a process that fails during any given hour with probability p. What about the variance?

By Lemma 19.3.1,

$$\mathbf{Var}[C] = \mathbf{Ex}[C^2] - (1/p)^2 \tag{19.3.3}$$

so all we need is a formula for $\mathbf{Ex}[C^2]$.

Reasoning about C using conditional expectation worked nicely in Section 18.4.6 to find mean time to failure, and a similar approach works for C^2. Namely, the expected value of C^2 is the probability, p, of failure in the first hour times 1^2, plus the probability, $(1-p)$, of non-failure in the first hour times the expected value of $(C+1)^2$. So

$$\begin{aligned}
\mathbf{Ex}[C^2] &= p \cdot 1^2 + (1-p)\mathbf{Ex}[(C+1)^2] \\
&= p + (1-p)\left(\mathbf{Ex}[C^2] + \frac{2}{p} + 1\right) \\
&= p + (1-p)\mathbf{Ex}[C^2] + (1-p)\left(\frac{2}{p} + 1\right), \quad \text{so}
\end{aligned} \tag{19.3.4}$$

$$\begin{aligned} p\mathrm{Ex}[C^2] &= p + (1-p)\left(\frac{2}{p}+1\right) \\ &= \frac{p^2+(1-p)(2+p)}{p} \quad \text{and} \\ \mathrm{Ex}[C^2] &= \frac{2-p}{p^2}. \end{aligned} \tag{19.3.5}$$

Combining this with (19.3.3) proves

Lemma 19.3.3. *If failures occur with probability p independently at each step, and C is the number of steps until the first failure*[2] *, then*

$$\mathrm{Var}[C] = \frac{1-p}{p^2}. \tag{19.3.6}$$

Dealing with Constants

It helps to know how to calculate the variance of $aR+b$:

Theorem 19.3.4

*[*Square Multiple Rule *for Variance] Let R be a random variable and a a constant. Then*

$$\mathrm{Var}[aR] = a^2\mathrm{Var}[R]. \tag{19.3.7}$$

Proof

Beginning with the definition of variance and repeatedly applying linearity of expectation, we have:

$$\begin{aligned} \mathrm{Var}[aR] ::&= \mathrm{Ex}[(aR-\mathrm{Ex}[aR])^2] \\ &= \mathrm{Ex}[(aR)^2 - 2aR\mathrm{Ex}[aR] + \mathrm{Ex}^2[aR]] \\ &= \mathrm{Ex}[(aR)^2] - \mathrm{Ex}[2aR\mathrm{Ex}[aR]] + \mathrm{Ex}^2[aR] \\ &= a^2\mathrm{Ex}[R^2] - 2\mathrm{Ex}[aR]\mathrm{Ex}[aR] + \mathrm{Ex}^2[aR] \\ &= a^2\mathrm{Ex}[R] - a^2\mathrm{Ex}^2[R] \\ &= a^2(\mathrm{Ex}[R^2] - \mathrm{Ex}^2[R]) \\ &= a^2\mathrm{Var}[R] \qquad \text{(Lemma 19.3.1)} \end{aligned} \tag{19.3.8}$$

■

It's even simpler to prove that adding a constant does not change the variance, as the reader can verify:

Theorem 19.3.5

Let R be a random variable and b a constant. Then

$$\mathrm{Var}[R+b] = \mathrm{Var}[R]. \tag{19.3.9}$$

Recalling that the standard deviation is the square root of variance, this implies that the standard deviation of $aR+b$ is simply $|a|$ times the standard deviation of R:

Corollary 19.3.6.

$$\sigma_{(aR+b)} = |a|\sigma_R.$$

Variance of a Sum

In general, the variance of a sum is not equal to the sum of the variances, but variances do add for *independent* variables. In fact, *mutual* independence is not necessary: *pairwise* independence will do. This is useful to know because there are some important situations, such as Birthday Matching in Section 16.4, that involve variables that are pairwise independent but not mutually independent.

Theorem 19.3.7

If R and S are independent random variables, then

$$\mathbf{Var}[R+S] = \mathbf{Var}[R] + \mathbf{Var}[S]. \tag{19.3.10}$$

Proof

We may assume that $\mathbf{Ex}[R] = 0$, since we could always replace R by $R - \mathbf{Ex}[R]$ in equation (19.3.10); likewise for S. This substitution preserves the independence of the variables, and by Theorem 19.3.5, does not change the variances.

But for any variable T with expectation zero, we have $\mathbf{Var}[T] = \mathbf{Ex}[T^2]$, so we need only prove

$$\mathbf{Ex}[(R+S)^2] = \mathbf{Ex}[R^2] + \mathbf{Ex}[S^2]. \tag{19.3.11}$$

But (19.3.11) follows from linearity of expectation and the fact that

$$\mathbf{Ex}[RS] = \mathbf{Ex}[R]\mathbf{Ex}[S] \tag{19.3.12}$$

since R and S are independent:

$$\begin{aligned}
\mathbf{Ex}[(R+S)^2] &= \mathbf{Ex}[R^2 + 2RS + S^2] \\
&= \mathbf{Ex}[R^2] + 2\mathbf{Ex}[RS] + \mathbf{Ex}[S^2] \\
&= \mathbf{Ex}[R^2] + 2\mathbf{Ex}[R]\mathbf{Ex}[S] + \mathbf{Ex}[S^2] \qquad \text{(by (19.3.8))} \\
&= \mathbf{Ex}[R^2] + 2\cdot 0\cdot 0 + \mathbf{Ex}[S^2] \\
&= \mathbf{Ex}[R^2] + \mathbf{Ex}[S^2]
\end{aligned} \tag{19.3.13}$$

■

It's easy to see that additivity of variance does not generally hold for variables that are not independent. For example, if $R = S$, then equation (19.3.10) becomes $\mathbf{Var}[R+R] = \mathbf{Var}[R] + \mathbf{Var}[R]$. By the Square Multiple Rule, Theorem 19.3.4, this holds iff $4\mathbf{Var}[R] = 2\mathbf{Var}[R]$, which implies that $\mathbf{Var}[R] = 0$. So equation (19.3.10) fails when $R = S$ and R has nonzero variance.

The proof of Theorem 19.3.7 carries over to the sum of any finite number of variables. So we have:

Theorem 19.3.8

[Pairwise Independent Additivity of Variance] If $R_1, R_2, \ldots, R_n$ are pairwise *independent random variables, then*

$$\mathbf{Var}[R_1 + R_2 + \cdots + R_n] = \mathbf{Var}[R_1] + \mathbf{Var}[R_2] + \cdots + \mathbf{Var}[R_n]. \tag{19.3.14}$$

Now we have a simple way of computing the variance of a variable, J, that has an (n,p)-binomial distribution. We know that $J = \sum_{k=1}^{n} I_k$ where the I_k are mutually independent indicator variables with $\mathbf{Pr}[I_k = 1] = p$. The variance of each I_k is $p(1-p)$ by Corollary 19.3.2, so by linearity of variance, we have

Lemma 19.3.9 (Variance of the Binomial Distribution). *If J has the (n,p)-binomial distribution, then*

$$\mathbf{Var}[J] = n\mathbf{Var}[I_k] = np(1-p). \tag{19.3.15}$$

[2]That is, C has the geometric distribution with parameter p according to Definition 18.4.6.

This page titled 19.3: Properties of Variance is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

19.4: Estimation by Random Sampling

Democratic politicians were astonished in 2010 when their early polls of sample voters showed Republican Scott Brown was favored by a majority of voters and so would win the special election to fill the Senate seat that the late Democrat Teddy Kennedy had occupied for over 40 years. Based on their poll results, they mounted an intense, but ultimately unsuccessful, effort to save the seat for their party.

Voter Poll

Suppose at some time before the election that p was the fraction of voters favoring Scott Brown. We want to estimate this unknown fraction p. Suppose we have some random process for selecting voters from registration lists that selects each voter with equal probability. We can define an indicator variable, K, by the rule that $K = 1$ if the random voter most prefers Brown, and $K = 0$ otherwise.

Now to estimate p, we take a large number, n, of random choices of voters[3] and count the fraction who favor Brown. That is, we define variables $K_1, K_2, \ldots$, where K_i is interpreted to be the indicator variable for the event that the ith chosen voter prefers Brown. Since our choices are made independently, the K_i's are independent. So formally, we model our estimation process by assuming we have mutually independent indicator variables $K_1, K_2, \ldots$, each with the same probability, p, of being equal to 1. Now let S_n be their sum, that is,

$$S_n ::= \sum_{i=1}^{n} K_i. \tag{19.4.1}$$

The variable S_n/n describes the fraction of sampled voters who favor Scott Brown. Most people intuitively, and correctly, expect this sample fraction to give a useful approximation to the unknown fraction, p.

So we will use the sample value, S_n/n, as our *statistical estimate* of p. We know that S_n has a binomial distribution with parameters n and p; we can choose n, but p is unknown.

How Large a Sample?

Suppose we want our estimate to be within 0.04 of the fraction, p, at least 95% of the time. This means we want

$$\Pr\left[\left|\frac{S_n}{n} - p\right| \le 0.04\right] \ge 0.95. \tag{19.4.2}$$

So we'd better determine the number, n of times we must poll voters so that inequality (19.4.2) will hold. Chebyshev's Theorem offers a simple way to determine such a n.

Sn is binomially distributed. Equation (19.3.10), combined with the fact that $p(1-p)$ is maximized when $p = 1-p$, that is, when $p = 1/2$ (check for yourself!), gives

$$\text{Var}[S_n] = n(p(1-p)) \le n \cdot \frac{1}{4} = \frac{n}{4}. \tag{19.4.3}$$

Next, we bound the variance of S_n/n:

$$\begin{aligned} \text{Var}\left[\frac{S_n}{n}\right] &= \left(\frac{1}{n}^2\right) \text{Var}[S_n] && \text{((Square Multiple Rule for Variance (19.3.4))} \\ &\le \frac{1}{n}^2 \frac{n}{4} && \text{(by (19.4.3))} \\ &= \frac{1}{4n} \end{aligned} \tag{19.4.4}$$

Using Chebyshev's bound and (19.4.4) we have:

$$\Pr\left[\left|\frac{S_n}{n} - p\right| \ge 0.04\right] \le \frac{\text{Var}[S_n/n]}{(0.04)^2} \le \frac{1}{4n(0.04)^2} = \frac{156.25}{n} \tag{19.4.5}$$

To make our our estimate with 95% confidence, we want the righthand side of (19.4.5) to be at most $1/20$. So we choose n so that

$$\frac{156.25}{n} \leq \frac{1}{20},$$

that is,

$$n \geq 3,125.$$

Section 19.6.2 describes how to get tighter estimates of the tails of binomial distributions that lead to a bound on n that is about four times smaller than the one above. But working through this example using only the variance illustrates an approach to estimation that is applicable to arbitrary random variables, not just binomial variables.

Matching Birthdays

There are important cases where the relevant distributions are not binomial because the mutual independence properties of the voter preference example do not hold. In these cases, estimation methods based on Chebyshev's Theorem may be the best approach. Birthday Matching is an example. We already saw in Section 16.4 that in a class of 95 students, it is virtually certain that at least one pair of students will have the same birthday, which suggests that several pairs of students are likely to have the same birthday. How many matched birthdays should we expect?

As before, suppose there are n students and d days in the year, and let M be the number of pairs of students with matching birthdays. Now it will be easy to calculate the expected number of pairs of students with matching birthdays. Then we can take the same approach as we did in estimating voter preferences to get an estimate of the probability of getting a number of pairs close to the expected number.

Unlike the situation with voter preferences, having matching birthdays for different pairs of students are not mutually independent events. Knowing Alice's birthday matches Bob's tells us nothing about who Carol matches, and knowing Alice has the same birthday as Carol tells us nothing about who Bob matches. But if Alice matches Bob and Alice matches Carol, it's certain that Bob and Carol match as well! The events that various pairs of students have matching birthdays are not mutually independent, and indeed not even three-way independent. The best we can say is that they are pairwise independent. This will allow us to apply the same reasoning to Birthday Matching as we did for voter preference. Namely, let $B_1, B_2, \ldots, B_n$ be the birthdays of n independently chosen people, and let $E_{i,j}$ be the indicator variable for the event that the ith and jth people chosen have the same birthdays, that is, the event $[B_i = B_j]$. So in our probability model, the B_i's are mutually independent variables, and the $E_{i,j}$'s are pairwise independent. Also, the expectations of $E_{i,j}$ for $i \neq j$ equals the probability that $B_i = B_j$, namely, $1/d$.

Now, M, the number of matching pairs of birthdays among the n choices, is simply the sum of the $E_{i,j}$'s:

$$M ::= \sum_{1 \leq i < j \leq n} E_{i,j}. \tag{19.4.6}$$

So by linearity of expectation

$$\mathbf{Ex}[M] = \mathbf{Ex}\left[\sum_{1 \leq i < j \leq n} E_{i,j}\right] = \sum_{1 \leq i < j \leq n} \mathbf{Ex}E_{i,j} = \binom{n}{2} \cdot \frac{1}{d}.$$

Similarly,

$$\begin{aligned} \mathbf{Var}[M] &= \mathbf{Var}\left[\sum_{1 \leq i < j \leq n} E_{i,j}\right] \\ &= \sum_{1 \leq i < j \leq n} \mathbf{Var}E_{i,j} \qquad \text{((Theorem 19.3.8)} \\ &= \binom{n}{2} \cdot \frac{1}{d}\left(1 - \frac{1}{d}\right). \qquad \text{((Corollary 19.3.2)} \end{aligned} \tag{19.4.7}$$

In particular, for a class of $n = 95$ students with $d = 365$ possible birthdays, we have $\mathbf{Ex}[M] \approx 12.23$ and $\mathbf{Var}[M] \approx 12.23(1 - 1/365) < 12.2$. So by Chebyshev's Theorem

$$\mathbf{Pr}[|M - \mathbf{Ex}[M]| \geq x] < \frac{12.2}{x^2}.$$

Letting $x = 7$, we conclude that there is a better than 75% chance that in a class of 95 students, the number of pairs of students with the same birthday will be within 7 of 12.23, that is, between 6 and 19.

Pairwise Independent Sampling

The reasoning we used above to analyze voter polling and matching birthdays is very similar. We summarize it in slightly more general form with a basic result called the Pairwise Independent Sampling Theorem. In particular, we do not need to restrict ourselves to sums of zero-one valued variables, or to variables with the same distribution. For simplicity, we state the Theorem for pairwise independent variables with possibly different distributions but with the same mean and variance.

Theorem 19.4.1

(Pairwise Independent Sampling). *Let $G_1, \ldots, G_n$ be pairwise independent variables with the same mean, μ, and deviation, σ. Define*

$$S_n ::= \sum_{i=1}^{n} G_i. \tag{19.4.8}$$

Then

$$\Pr\left[\left|\frac{S_n}{n} - \mu\right| \geq x\right] \leq \frac{1}{n}\left(\frac{\sigma}{x}^2\right).$$

Proof

We observe first that the expectation of S_n/n is μ:

$$\begin{aligned} \operatorname{Ex}\left[\frac{S_n}{n}\right] &= \operatorname{Ex}\left[\frac{\sum_{i=1}^{n} G_i}{n}\right] && \text{(def of } S_n\text{)} \\ &= \frac{\sum_{i=1}^{n} \operatorname{Ex}[G_i]}{n} && \text{(linearity of expectation)} \\ &= \frac{\sum_{i=1}^{n} \mu}{n} \\ &= \frac{n\mu}{n} = \mu. \end{aligned} \tag{19.4.9}$$

The second important property of S_n/n is that its variance is the variance of G_i divided by n:

$$\begin{aligned} \operatorname{Var}\left[\frac{S_n}{n}\right] &= \left(\frac{1}{n}\right)^2 \operatorname{Var}[S_n] && \text{(Square Multiple Rule for Variance (19.3.4))} \\ &= \frac{1}{n^2} \operatorname{Var}\left[\sum_{i=1}^{n} G_i\right] && \text{(def of } S_n\text{)} \\ &= \frac{1}{n^2} \sum_{i=1}^{n} \operatorname{Var}[G_i] && \text{(pairwise independent additivity)} \\ &= \frac{1}{n^2} \cdot n\sigma^2 = \frac{\sigma^2}{n}. \end{aligned} \tag{19.4.10}$$

This is enough to apply Chebyshev's Theorem and conclude:

$$\begin{aligned} \Pr\left[\left|\frac{S_n}{n} - \mu\right| \geq x\right] &\leq \frac{\operatorname{Var}[S_n/n]}{x^2}. && \text{(Chebyshev's bound)} \\ &= \frac{\sigma^2/n}{x^2} && \text{(by (19.4.8))} \\ &= \frac{1}{n}\left(\frac{\sigma}{x}\right)^2. \end{aligned} \tag{19.4.11}$$

■

The Pairwise Independent Sampling Theorem provides a quantitative general statement about how the average of independent samples of a random variable approaches the mean. In particular, it proves what is known as the Law of Large Numbers[4]: by choosing a large enough sample size, we can get arbitrarily accurate estimates of the mean with confidence arbitrarily close to 100%.

Corollary 19.4.2. *[Weak Law of Large Numbers] Let $G_1, \ldots, G_n$ be pairwise independent variables with the same mean, μ, and the same finite deviation, and let*

$$S_n ::= \frac{\sum_{i=1}^{n} G_i}{n}.$$

Then for every $\epsilon > 0$,

$$\lim_{n \to \infty} \Pr[|S_n - \mu| \leq \epsilon] = 1.$$

[3]We're choosing a random voter n times *with replacement*. We don't remove a chosen voter from the set of voters eligible to be chosen later; so we might choose the same voter more than once! We would get a slightly better estimate if we required n *different* people to be chosen, but doing so complicates both the selection process and its analysis for little gain.

[4]This is the *Weak* Law of Large Numbers. As you might suppose, there is also a Strong Law, but it's outside the scope of 6.042.

This page titled 19.4: Estimation by Random Sampling is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

19.5: Confidence versus Probability

So Chebyshev's Bound implies that sampling 3,125 voters will yield a fraction that, 95% of the time, is within 0.04 of the actual fraction of the voting population who prefer Brown.

Notice that the actual size of the voting population was never considered because *it did not matter*. People who have not studied probability theory often insist that the population size should influence the sample size. But our analysis shows that polling a little over 3000 people people is always sufficient, regardless of whether there are ten thousand, or a million, or a billion voters. You should think about an intuitive explanation that might persuade someone who thinks population size matters.

Now suppose a pollster actually takes a sample of 3,125 random voters to estimate the fraction of voters who prefer Brown, and the pollster finds that 1250 of them prefer Brown. It's tempting, **but sloppy**, to say that this means:

False Claim. *With probability 0.95, the fraction, p, of voters who prefer Brown is $1250/3125 \pm 0.04$. Since $1250/3125 - 0.04 > 1/3$, there is a 95% chance that more than a third of the voters prefer Brown to all other candidates.*

What's objectionable about this statement is that it talks about the probability or "chance" that a real world fact is true, namely that the actual fraction, p, of voters favoring Brown is more than $1/3$. But p is what it is, and it simply makes no sense to talk about the probability that it is something else. For example, suppose p is actually 0.3; then it's nonsense to ask about the probability that it is within 0.04 of $1250/3125$. It simply isn't.

This example of voter preference is typical: we want to estimate a fixed, unknown real-world quantity. But *being unknown does not make this quantity a random variable*, so it makes no sense to talk about the probability that it has some property.

A more careful summary of what we have accomplished goes this way:

> We have described a probabilistic procedure for estimating the value of the actual fraction, p. The probability that *our estimation procedure* will yield a value within 0.04 of p is 0.95.

This is a bit of a mouthful, so special phrasing closer to the sloppy language is commonly used. The pollster would describe his conclusion by saying that

At the 95% *confidence level*, the fraction of voters who prefer Brown is $1250/3125 \pm 0.04$.

So confidence levels refer to the results of estimation procedures for real-world quantities. The phrase "confidence level" should be heard as a reminder that some statistical procedure was used to obtain an estimate, and in judging the credibility of the estimate, it may be important to learn just what this procedure was.

This page titled 19.5: Confidence versus Probability is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

19.6: Sums of Random Variables

If all you know about a random variable is its mean and variance, then Chebyshev's Theorem is the best you can do when it comes to bounding the probability that the random variable deviates from its mean. In some cases, however, we know more—for example, that the random variable has a binomial distribution— and then it is possible to prove much stronger bounds. Instead of polynomially small bounds such as $1/c^2$, we can sometimes even obtain exponentially small bounds such as $1/e^c$. As we will soon discover, this is the case whenever the random variable T is the sum of n mutually independent random variables $T_1, T_2, \ldots, T_n$ where $0 \leq T_i \leq 1$. A random variable with a binomial distribution is just one of many examples of such a T. Here is another.

Motivating Example

Fussbook is a new social networking site oriented toward unpleasant people. Like all major web services, Fussbook has a load balancing problem: it receives lots of forum posts that computer servers have to process. If any server is assigned more work than it can complete in a given interval, then it is overloaded and system performance suffers. That would be bad, because Fussbook users are *not* a tolerant bunch. So balancing the work load across mutliple servers is vital.

An early idea was to assign each server an alphabetic range of forum topics. ("That oughta work!", one programmer said.) But after the computer handling the "*pr*ivacy" and "*pr*eferred text editor" threads melted from overload, the drawback of an *ad hoc* approach was clear: it's easy to miss something that will mess up your plan.

If the length of every task were known in advance, then finding a balanced distribution would be a kind of "bin packing" problem. Such problems are hard to solve exactly, but approximation algorithms can come close. Unfortunately, in this case task lengths are not known in advance, which is typical of workload problems in the real world.

So the load balancing problem seems sort of hopeless, because there is no data available to guide decisions. So the programmers of Fussbook gave up and just randomly assigned posts to computers. Imagine their surprise when the system stayed up and hasn't crashed yet!

As it turns out, random assignment not only balances load reasonably well, but also permits provable performance guarantees. In general, a randomized approach to a problem is worth considering when a deterministic solution is hard to compute or requires unavailable information.

Specifically, Fussbook receives 24,000 forum posts in every 10-minute interval. Each post is assigned to one of several servers for processing, and each server works sequentially through its assigned tasks. It takes a server an average of $1/4$ second to process a post. Some posts, such as pointless grammar critiques and snide witticisms, are easier, but no post—not even the most protracted harangues—takes more than one full second.

Measuring workload in seconds, this means a server is overloaded when it is assigned more than 600 units of work in a given 600 second interval. Fussbook's average processing load of $24,000 \cdot 1/4 = 6000$ seconds per interval would keep 10 computers running at 100% capacity with perfect load balancing. Surely, more than 10 servers are needed to cope with random fluctuations in task length and imperfect load balance. But would 11 be enough? . . . or 15, 20, 100? We'll answer that question with a new mathematical tool.

The Chernoff Bound

The Chernoff[5] bound is a hammer that you can use to nail a great many problems. Roughly, the Chernoff bound says that certain random variables are very unlikely to significantly exceed their expectation. For example, if the expected load on a processor is just a bit below its capacity, then that processor is unlikely to be overloaded, provided the conditions of the Chernoff bound are satisfied.

More precisely, the Chernoff Bound says that *the sum of lots of little, independent random variables is unlikely to significantly exceed the mean of the sum*. The Markov and Chebyshev bounds lead to the same kind of conclusion but typically provide much weaker bounds. In particular, the Markov and Chebyshev bounds are polynomial, while the Chernoff bound is exponential.

Here is the theorem. The proof will come later in Section 19.6.6.

Theorem 19.6.1

(Chernoff Bound). *Let $T_1, \ldots, T_n$ be mutually independent random variables such that $0 \leq T_i \leq 1$ for all i. Let $T = T_1 + \cdots + T_n$. Then for all $c \geq 1$,*

$$\mathbf{Pr}[T \geq c\mathbf{Ex}[T]] \leq e^{-\beta(c)\mathbf{Ex}[T]} \tag{19.6.1}$$

where $\beta(c) ::= c\ln c - c + 1$.

The Chernoff bound applies only to distributions of sums of independent random variables that take on values in the real interval $[0,1]$. The binomial distribution is the most well-known distribution that fits these criteria, but many others are possible, because the Chernoff bound allows the variables in the sum to have differing, arbitrary, or even unknown distributions over the range $[0,1]$. Furthermore, there is no direct dependence on either the number of random variables in the sum or their expectations. In short, the Chernoff bound gives strong results for lots of problems based on little information—no wonder it is widely used!

Chernoff Bound for Binomial Tails

The Chernoff bound can be applied in easy steps, though the details can be daunting at first. Let's walk through a simple example to get the hang of it: bounding the probability that the number of heads that come up in 1000 independent tosses of a coin exceeds the expectation by 20% or more. Let T_i be an indicator variable for the event that the ith coin is heads. Then the total number of heads is

$$T = T_1 + \cdots + T_{1000}.$$

The Chernoff bound requires that the random variables T_i be mutually independent and take on values in the range $[0,1]$. Both conditions hold here. In this example the T_i's only take the two values 0 and 1, since they're indicators.

The goal is to bound the probability that the number of heads exceeds its expectation by 20% or more; that is, to bound $\mathbf{Pr}[T \geq c\mathbf{Ex}[T]]$ where $c = 1.2$. To that end, we compute $\beta(c)$ as defined in the theorem:

$$\beta(c) = c\ln(c) - c + 1 = 0.0187\ldots$$

If we assume the coin is fair, then $\mathbf{Ex}[T] = 500$ Plugging these values into the Chernoff bound gives:

$$\begin{aligned} \mathbf{Pr}[T \geq 1.2\mathbf{Ex}[T]] &\leq e^{-\beta(c)\cdot\mathbf{Ex}[T]} \\ &= e^{-(0.0187\ldots)\cdot 500} < 0.0000834. \end{aligned} \tag{19.6.2}$$

So the probability of getting 20% or more extra heads on 1000 coins is less than 1 in 10,000.

The bound rapidly becomes much smaller as the number of coins increases, because the expected number of heads appears in the exponent of the upper bound. For example, the probability of getting at least 20% extra heads on a million coins is at most

$$e^{-(0.0187\ldots)\cdot 500000} < e^{-9392},$$

which is an inconceivably small number.

Alternatively, the bound also becomes stronger for larger deviations. For example, suppose we're interested in the odds of getting 30% or more extra heads in 1000 tosses, rather than 20%. In that case, $c = 1.3$ instead of 1.2. Consequently, the parameter $\beta(c)$ rises from 0.0187 to about 0.0410, which may not seem significant, but because $\beta(c)$ appears in the exponent of the upper bound, the final probability decreases from around 1 in 10,000 to about 1 in a billion!

Chernoff Bound for a Lottery Game

Pick-4 is a lottery game in which you pay $1 to pick a 4-digit number between 0000 and 9999. If your number comes up in a random drawing, then you win $5,000. Your chance of winning is 1 in 10,000. If 10 million people play, then the expected number of winners is 1000. When there are exactly 1000 winners, the lottery keeps $5 million of the $10 million paid for tickets. The lottery operator's nightmare is that the number of winners is much greater—especially at the point where more than 2000 win and the lottery must pay out more than it received. What is the probability that will happen?

Let T_i be an indicator for the event that the ith player wins. Then $T = T_1 + \cdots + T_n$ is the total number of winners. If we assume[6] that the players' picks and the winning number are random, independent and uniform, then the indicators T_i are independent, as required by the Chernoff bound.

Since 2000 winners would be twice the expected number, we choose $c = 2$, compute $\beta(c) = 0.386\ldots$, and plug these values into the Chernoff bound:

$$\begin{aligned} \Pr[T \geq 2000] &= \Pr[T \geq 2\mathrm{Ex}[T]] \\ &\leq e^{-k\mathrm{Ex}[T]} = e^{-(0.386\ldots)\cdot 1000} \\ &< e^{-386}. \end{aligned} \tag{19.6.3}$$

So there is almost no chance that the lottery operator pays out more than it took in. In fact, the number of winners won't even be 10% higher than expected very often. To prove that, let $c = 1.1$, compute $\beta(c) = 0.00484\ldots$, and plug in again:

$$\begin{aligned} \Pr[T \geq 1.1\mathrm{Ex}[T]] &\leq e^{-k\mathrm{Ex}[T]} \\ &= e^{-(0.00484)\cdot 1000} < 0.01. \end{aligned} \tag{19.6.4}$$

So the Pick-4 lottery may be exciting for the players, but the lottery operator has little doubt as to the outcome!

Randomized Load Balancing

Now let's return to Fussbook and its load balancing problem. Specifically, we need to determine a number, m, of servers that makes it very unlikely that any server is overloaded by being assigned more than 600 seconds of work in a given interval.

To begin, let's find the probability that the first server is overloaded. Letting T be the number of seconds of work assigned to the first server, this means we want an upper bound on $\Pr[T \geq 600]$. Let T_i be the number of seconds that the first server spends on the i th task: then T_i is zero if the task is assigned to another machine, and otherwise T_i is the length of the task. So $T = \sum_{i=1}^{T_i}$ is the total number of seconds of work assigned to the first server, where $n = 24,000$.

The Chernoff bound is applicable only if the T_i are mutually independent and take on values in the range $[0, 1]$. The first condition is satisfied if we assume that assignment of a post to a server is independent of the time required to process the post. The second condition is satisfied because we know that no post takes more than 1 second to process; this is why we chose to measure work in seconds.

In all, there are 24,000 tasks, each with an expected length of $1/4$ second. Since tasks are assigned to the m servers at random, the expected load on the first server is:

$$\begin{aligned} \mathrm{Ex}[T] &= \frac{24,000 \text{ tasks } \cdot 1/4 \text{ second per task}}{m \text{ servers}} \\ &= 6000/m \text{ seconds }. \end{aligned} \tag{19.6.5}$$

So if there are fewer than 10 servers, then the expected load on the first server is greater than its capacity, and we can expect it to be overloaded. If there are exactly 10 servers, then the server is expected to run for $6000/10 = 600$ seconds, which is 100% of its capacity.

Now we can use the Chernoff bound based on the number of servers to bound the probability that the first server is overloaded. We have from (19.6.5)

$$600 = c\mathrm{Ex}[T] \quad \text{where } c ::= m/10,$$

so by the Chernoff bound

$$\Pr[T \geq 600] = \Pr[T \geq c\mathrm{Ex}[T]] \leq e^{-(c\ln(c)-c+1)\cdot 6000/m},$$

The probability that *some* server is overloaded is at most m times the probability that the first server is overloaded, by the Union Bound in Section 16.5.2. So

$$\begin{aligned} \Pr[\text{some server is overloaded}] &\leq \sum_{i=1}^{m} \Pr[\text{server } i \text{ is overloaded}] \\ &= m\Pr[\text{the first server is overloaded}] \\ &\leq me^{-(c\ln(c)-c+1)\cdot 6000/m}, \end{aligned} \tag{19.6.6}$$

where $c = m/10$. Some values of this upper bound are tabulated below:

$$\begin{aligned} m = 11 &: 0.784\ldots \\ m = 12 &: 0.000999\ldots \\ m = 13 &: 0.0000000760\ldots \end{aligned} \tag{19.6.7}$$

These values suggest that a system with $m = 11$ machines might suffer immediate overload, $m = 12$ machines could fail in a few days, but $m = 13$ should be fine for a century or two!

Proof of the Chernoff Bound

The proof of the Chernoff bound is somewhat involved. In fact, *Chernoff himself* couldn't come up with it: his friend, Herman Rubin, showed him the argument. Thinking the bound not very significant, Chernoff did not credit Rubin in print. He felt pretty bad when it became famous![7]

Proof. (of Theorem 19.6.1)

For clarity, we'll go through the proof "top down." That is, we'll use facts that are proved immediately afterward.

The key step is to exponentiate both sides of the inequality $T \geq c\mathrm{Ex}[T]$ and then apply the Markov bound:

$$\begin{aligned} \Pr[T \geq c\mathrm{Ex}[T]] &= \Pr[c^T \geq c^{c\mathrm{Ex}[T]}] \\ &\leq \frac{\mathrm{Ex}[c^T]}{c^{c\mathrm{Ex}[T]}} && \text{(Markov Bound)} \\ &\leq \frac{e^{(c-1)\mathrm{Ex}[T]}}{c^{c\mathrm{Ex}[T]}} && \text{(Lemma 19.6.2 below)} \\ &= \frac{e^{(c-1)\mathrm{Ex}[T]}}{e^{c\ln(c)\mathrm{Ex}[T]}} = e^{-(c\ln(c)-c+1)\mathrm{Ex}[T]}. && \blacksquare \end{aligned} \tag{19.6.8}$$

Algebra aside, there is a brilliant idea in this proof: in this context, exponentiating somehow supercharges the Markov bound. This is not true in general! One unfortunate side-effect of this supercharging is that we have to bound some nasty expectations involving exponentials in order to complete the proof. This is done in the two lemmas below, where variables take on values as in Theorem 19.6.1.

Lemma 19.6.2.

$$\mathrm{Ex}\left[c^T\right] \leq e^{(c-1)\mathrm{Ex}[T]}.$$

Proof.

$$\begin{aligned} \mathrm{Ex}\left[c^T\right] &= \mathrm{Ex}\left[c^{T_1+\cdots+T_n}\right] && \text{(def of } T\text{)} \\ &= \mathrm{Ex}\left[c^{T_1}\cdots c^{T_n}\right] \\ &= \mathrm{Ex}\left[c^{T_1}\right]\cdots\mathrm{Ex}\left[c^{T_n}\right] && \text{(independent product Cor 18.5.7)} \\ &\leq e^{(c-1)\mathrm{Ex}[T_1]}\ldots e^{(c-1)\mathrm{Ex}[T_n]} && \text{(Lemma 19.6.3 below)} \\ &= e^{(c-1)(\mathrm{Ex}[T_1]+\cdots+\mathrm{Ex}[T_n])} \\ &= e^{(c-1)(\mathrm{Ex}[T_1+\cdots+T_n])} \\ &= e^{(c-1)(\mathrm{Ex}[T])}. \end{aligned} \tag{19.6.9}$$

The third equality depends on the fact that functions of independent variables are also independent (see Lemma 18.2.2). $\blacksquare$

Lemma 19.6.3.

$$\mathrm{Ex}[c^{T_i}] \leq e^{(c-1)\mathrm{Ex}[T_i]}$$

Proof. All summations below range over values v taken by the random variable T_i, which are all required to be in the interval $[0, 1]$.

$$\begin{aligned}
\mathbf{Ex}[c^{T_i}] &= c^v \mathbf{Pr}[T_i = v] && \text{(def of Ex[·])} \\
&\le \sum (1 + (c-1)v)\mathbf{Pr}[T_i = v] && \text{(convexity—see below)} \\
&= \sum \mathbf{Pr}[T_i = v] + (c-1)v\mathbf{Pr}[T_i = v] \\
&= \sum \mathbf{Pr}[T_i = v] + (c-1)\sum v\mathbf{Pr}[T_i = v] \\
&= 1 + (c-1)\mathbf{Ex}[T_i] \\
&\le e^{(c-1)\mathbf{Ex}[T_i]} && \text{(since } 1 + z \le e^z\text{)}
\end{aligned} \tag{19.6.10}$$

The second step relies on the inequality

$$c^v \le 1 + (c-1)v,$$

which holds for all v in $[0,1]$ and $c \ge 1$. This follows from the general principle that a convex function, namely c^v, is less than the linear function, $1 + (c-1)v$, between their points of intersection, namely $v = 0$ and 1. This inequality is why the variables Ti are restricted to the real interval $[0,1]$. ■

Comparing the Bounds

Suppose that we have a collection of mutually independent events $A_1, A_2, \ldots, A_n$, and we want to know how many of the events are likely to occur.

Let T_i be the indicator random variable for A_i and define

$$p_i = \mathbf{Pr}[T_i = 1] = \mathbf{Pr}[A_i]$$

for $1 \le i \le n$. Define

$$T = T_1 + T_2 + \cdots + T_n$$

to be the number of events that occur.

We know from Linearity of Expectation that

$$\begin{aligned}
\mathbf{Ex}[T] &= \mathbf{Ex}[T_1] + \mathbf{Ex}[T_2] + \cdots + \mathbf{Ex}[T_n] \\
&= \sum_{i=1}^{n} p_i.
\end{aligned} \tag{19.6.11}$$

This is true even if the events are *not* independent.

By Theorem 19.3.8, we also know that

$$\begin{aligned}
\mathbf{Var}[T] &= \mathbf{Var}[T_1] + \mathbf{Var}[T_2] + \cdots + \mathbf{Var}[T_n] \\
&= \sum_{i=1}^{n} p_i(1 - p_i),
\end{aligned} \tag{19.6.12}$$

and thus that

$$\sigma_T = \sqrt{\sum_{i=1}^{n} p_i(1 - p_i)}.$$

This is true even if the events are only pairwise independent.

Markov's Theorem tells us that for any $c > 1$,

$$\mathbf{Pr}[T \ge c\mathbf{Ex}[T]] \le \frac{1}{c}.$$

Chebyshev's Theorem gives us the stronger result that

$$\mathbf{Pr}[|T - \mathbf{Ex}[T]| \ge c\sigma_T] \le \frac{1}{c^2}.$$

The Chernoff Bound gives us an even stronger result, namely, that for any $c > 0$,

$$\mathbf{Pr}[T - \mathbf{Ex}[T] \ge c\mathbf{Ex}[T]] \le e^{-(c\ln(c) - c + 1)\mathbf{Ex}[T]}.$$

In this case, the probability of exceeding the mean by $c\mathbf{Ex}[T]$ decreases as an exponentially small function of the deviation.

By considering the random variable $n - T$, we can also use the Chernoff Bound to prove that the probability that T is much *lower* than $\mathbf{Ex}[T]$ is also exponentially small.

Murphy's Law

If the expectation of a random variable is much less than 1, then Markov's Theorem implies that there is only a small probability that the variable has a value of 1 or more. On the other hand, a result that we call *Murphy's Law*[8] says that if a random variable is an independent sum of 0–1-valued variables and has a large expectation, then there is a huge probability of getting a value of at least 1.

Theorem 19.6.4

(Murphy's Law). *Let $A_1, A_2, \ldots, A_n$ be mutually independent events. Let T_i be the indicator random variable for A_i and define*

$$T ::= T_1 + T_2 + \cdots + T_n$$

to be the number of events that occur. Then

$$\Pr[T = 0] \leq e^{-\mathbf{Ex}[T]},$$

Proof

$$\begin{aligned}
\Pr[T=0] &= \Pr\left[\bar{A}_1 \cap \bar{A}_2 \cap \ldots \cap \bar{A}_n\right] && (T = 0 \text{ iff no } A_i \text{ occurs}) \\
&= \prod_{i=1}^{n} \Pr\left[\bar{A}_i\right] && (\text{independence of } A_i) \\
&= \prod_{i=1}^{n} (1 - \Pr[A_i]) \\
&\leq \prod_{i=1}^{n} e^{-\Pr[A_i]} && (\text{since } 1 - x \leq e^{-x}) \\
&= e^{-\sum_{i=1}^{n} \Pr[A_i]} \\
&= e^{-\sum_{i=1}^{n} \mathbf{Ex}[T_i]} && (\text{since } T_i \text{ is an indicator for } A_i) \\
&= e^{-\mathbf{Ex}[T]} && (\text{linearity of expectation})
\end{aligned} \tag{19.6.13}$$

■

For example, given any set of mutually independent events, if you expect 10 of them to happen, then at least one of them will happen with probability at least $1 - e^{10}$. The probability that none of them happen is at most $e^{10} < 1/22000$.

So if there are a lot of independent things that can go wrong and their probabilities sum to a number much greater than 1, then Theorem 19.6.4 proves that some of them surely will go wrong.

This result can help to explain "coincidences," "miracles," and crazy events that seem to have been very unlikely to happen. Such events do happen, in part, because there are so many possible unlikely events that the sum of their probabilities is greater than one. For example, someone *does* win the lottery.

In fact, if there are 100,000 random tickets in Pick-4, Theorem 19.6.4 says that the probability that there is no winner is less than $e^{10} < 1/22000$. More generally, there are literally millions of one-in-a-million possible events and so some of them will surely occur.

[5]Yes, this is the same Chernoff who figured out how to beat the state lottery—this guy knows a thing or two.

[6]As we noted in Chapter 18, human choices are often not uniform and they can be highly dependent. For example, lots of people will pick an important date. The lottery folks should not get too much comfort from the analysis that follows, unless they assign random 4-digit numbers to each player.

[7]See "A Conversation with Herman Chernoff," *Statistical Science* 1996, Vol. 11, No. 4, pp 335– 350.

[8]This is in reference and deference to the famous saying that "If something can go wrong, it probably will."

This page titled 19.6: Sums of Random Variables is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

19.7: Really Great Expectations

Making independent tosses of a fair coin until some desired pattern comes up is a simple process you should feel solidly in command of by now, right? So how about a bet about the simplest such process—tossing until a head comes up? Ok, you're wary of betting with us, but how about this: we'll let *you set the odds*.

Repeating Yourself

Here's the bet: you make independent tosses of a fair coin until a head comes up. Then you will repeat the process. If a second head comes up in the same or fewer tosses than the first, you have to start over yet again. You keep starting over until you finally toss a run of tails longer than your first one. The payment rules are that you will pay me 1 cent each time you start over. When you win by finally getting a run of tails longer than your first one, I will pay you some generous amount. Notice by the way that you're certain to win—whatever your initial run of tails happened to be, a longer run will eventually occur again with probability 1!

For example, if your first tosses are **TTTH**, then you will keep tossing until you get a run of 4 tails. So your winning flips might be

TTTHTHTTHHTTHTHTTTHTHHHTTTT.

In this run there are 10 heads, which means you had to start over 9 times. So you would have paid me 9 cents by the time you finally won by tossing 4 tails. Now you've won, and I'll pay you generously —how does 25 cents sound? Maybe you'd rather have $1? How about $1000?

Of course there's a trap here. Let's calculate your expected winnings.

Suppose your initial run of tails had length k. After that, each time a head comes up, you have to start over and try to get $k+1$ tails in a row. If we regard your getting $k+1$ tails in a row as a "failed" try, and regard your having to start over because a head came up too soon as a "successful" try, then the number of times you have to start over is the number of tries till the first failure. So the expected number of tries will be the mean time to failure, which is 2^{k+1}, because the probability of tossing $k+1$ tails in a row is $2^{-(k+1)}$.

Let T be the length of your initial run of tails. So $T = k$ means that your initial tosses were $T^k H$. Let R be the number of times you repeat trying to beat your original run of tails. The number of cents you expect to finish with is the number of cents in my generous payment minus $\mathbf{Ex}[R]$. It's now easy to calculate $\mathbf{Ex}[R]$ by conditioning on the value of T:

$$\mathbf{Ex}[R] = \sum_{k\in\mathbb{N}} \mathbf{Ex}[R \mid T = k] \cdot \mathbf{Pr}[T = k] = \sum_{k\in\mathbb{N}} 2^{k+1} \cdot 2^{-(k+1)} = \sum_{k\in\mathbb{N}} 1 = \infty.$$

So you can expect to pay me an infinite number of cents before winning my "generous" payment. No amount of generosity can make this bet fair! In fact this particular example is a special case of an astonishingly general one: the expected waiting time for *any* random variable to achieve a larger value is infinite.

This page titled 19.7: Really Great Expectations is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

20: Random Walks

Random Walks are used to model situations in which an object moves in a sequence of steps in randomly chosen directions. For example, physicists use threedimensional random walks to model Brownian motion and gas diffusion. In this chapter we'll examine two examples of random walks. First, we'll model gambling as a simple 1-dimensional random walk—a walk along a straight line. Then we'll explain how the Google search engine used random walks through the graph of world-wide web links to determine the relative importance of websites.

20.1: Gambler's Ruin

20.2: Random Walks on Graphs

This page titled 20: Random Walks is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

20.1: Gambler's Ruin

Suppose a gambler starts with an initial stake of n dollars and makes a sequence of \$1 bets. If he wins an individual bet, he gets his money back plus another \$1. If he loses the bet, he loses the \$1.

We can model this scenario as a random walk between integer points on the real line. The position on the line at any time corresponds to the gambler's cash-onhand, or *capital*. Walking one step to the right corresponds to winning a \$1 bet and thereby increasing his capital by \$1. Similarly, walking one step to the left corresponds to losing a \$1 bet.

The gambler plays until either he runs out of money or increases his capital to a target amount of T dollars. The amount $T - n$ is defined to be his *intended profit*.

If he reaches his target, he will have won his intended profit and is called an overall *winner*. If his capital reaches zero before reaching his target, he will have lost n dollars; this is called *going broke* or being *ruined*. We'll assume that the gambler has the same probability, p, of winning each individual \$1 bet, and that the bets are mutually independent. We'd like to find the probability that the gambler wins.

The gambler's situation as he proceeds with his \$1 bets is illustrated in Figure 20.1. The random walk has boundaries at 0 and T. If the random walk ever reaches either of these boundary values, then it terminates.

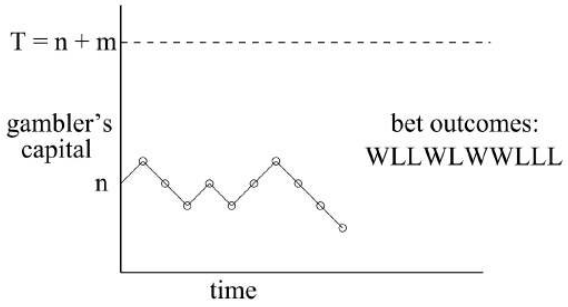


Figure 20.1 A graph of the gambler's capital versus time for one possible sequence of bet outcomes. At each time step, the graph goes up with probability p and down with probability $1 - p$. The gambler continues betting until the graph reaches either 0 or T. If he starts with \$$n$, his intended profit is \$$m$ where $T = n + m$.

In an *unbiased game*, the individual bets are fair: the gambler is equally likely to win or lose each bet—that is, $p = 1/2$. The gambler is more likely to win if $p > 1/2$ and less likely to win if $p < 1/2$; these random walks are called *biased*. We want to determine the probability that the walk terminates at boundary T —the probability that the gambler wins. We'll do this in Section 20.1.1. But before we derive the probability, let's examine what it turns out to be.

Let's begin by supposing the gambler plays an unbiased game starting with \$100 and will play until he goes broke or reaches a target of 200 dollars. Since he starts equidistant from his target and bankruptcy in this case, it's clear by symmetry that his probability of winning is $1/2$.

We'll show below that starting with n dollars and aiming for a target of $T \geq n$ dollars, the probability the gambler reaches his target before going broke is n/T. For example, suppose he wants to win the same \$100, but instead starts out with \$500. Now his chances are pretty good: the probability of his making the 100 dollars is $5/6$. And if he started with one million dollars still aiming to win \$100 dollars he almost certain to win: the probability is $1M/(1M + 100) > .9999$.

So in the unbiased game, the larger the initial stake relative to the target, the higher the probability the gambler will win, which makes some intuitive sense. But note that although the gambler now wins nearly all the time, when he loses, he loses *big*. Bankruptcy costs him \$1M, while when he wins, he wins only \$100. The gambler's average win remains zero dollars, which is what you'd expect when making fair bets.

Another useful way to describe this scenario is as a game between two players. Say Albert starts with \$500, and Eric starts with \$100. They flip a fair coin, and every time a Head appears, Albert wins \$1 from Eric, and vice versa for Tails. They play this game until one person goes bankrupt. This problem is identical to the Gambler's Ruin problem with $n = 500$ and $T = 100 + 500 = 600$. The probability of Albert winning is $500/600 = 5/6$.

Now suppose instead that the gambler chooses to play roulette in an American casino, always betting \$1 on red. Because the casino puts two green numbers on its roulette wheels, the probability of winning a single bet is a little less than $1/2$. The casino has an advantage, but the bets are close to fair, and you might expect that starting with \$500, the gambler has a reasonable chance of winning \$100—the $5/6$ probability of winning in the unbiased game surely gets reduced, but perhaps not too drastically.

This mistaken intuition is how casinos stay in business. In fact, the gambler's odds of winning \$100 by making \$1 bets against the "slightly" unfair roulette wheel are less than 1 in 37,000. If that's surprising to you, it only gets weirder from here: 1 in 37,000 is in fact an upper bound on the gambler's chance of winning *regardless of his starting stake*. Whether he starts with \$5000 or \$5 billion, he still has almost no chance of winning!

The Probability of Avoiding Ruin

We will determine the probability that the gambler wins using an idea of Pascal's dating back to the beginnings of the subject of probability.

Pascal viewed the walk as a two-player game between Albert and Eric as described above. Albert starts with a stack of n chips and Eric starts with a stack of $m = T - n$ chips. At each bet, Albert wins Eric's top chip with probability p and loses his top chip to Eric with probability $q ::= 1 - p$. They play this game until one person goes bankrupt.

Pascal's ingenious idea was to alter the worth of the chips to make the game fair regardless of p. Specifically, Pascal assigned Albert's bottom chip a worth of $r ::= q/p$ and then assigned successive chips *up* his stack worths equal to $r_2, r_3, \ldots$ up to his top chip with worth r_n. Eric's top chip gets assigned worth r^{n+1}, and the successive chips *down* his stack are worth $r^{n+2}, r^{n+3}, \ldots$ down to his bottom chip worth r^{n+m}.

The expected payoff of Albert's first bet is worth

$$r^{n+1} \cdot p - r^n \cdot q = \left(r^n \cdot \frac{q}{p}\right) \cdot p - r^n \cdot q = 0.$$

so this assignment makes the first bet a fair one in terms of worth. Moreover, whether Albert wins or loses the bet, the successive chip worths counting up Albert's stack and then down Eric's remain $r, r^2, \ldots, r^n, \ldots, r^{n+m}$, ensuring by the same reasoning that every bet has fair worth. So, Albert's expected worth at the end of the game is the sum of the expectations of the worth of each bet, which is 0.[1]

When Albert wins all of Eric's chips his total gain is worth

$$\sum_{i=n+1}^{n+m} r^i.$$

and when he loses all his chips to Eric, his total loss is worth $\sum_{i=1}^{n} r^i$. Letting w_n be Albert's probability of winning, we now have

$$0 = \text{Ex[worth of Albert's payoff]} = w_n \sum_{i=n+1}^{n+m} r^i - (1 - w_n) \sum_{i=1}^{n} r^i.$$

In the truly fair game when $r = 1$, we have $0 = mw_n - n(1 - w_n)$, so $w_n = n/(n+m)$, as claimed above.

In the biased game with $r \neq 1$, we have

$$0 = r \cdot \frac{r^{n+m} - r^n}{r - 1} \cdot w_n - r \cdot \frac{r^n - 1}{r - 1} \cdot (1 - w_n).$$

Solving for w_n gives

$$w_n = \frac{r^n - 1}{r^{n+m} - 1} = \frac{r^n - 1}{r^T - 1} \tag{20.1.1}$$

We have now proved

Theorem 20.1.1

In the Gambler's Ruin game with initial capital, n, target, T, and probability p of winning each individual bet,

$$\Pr[\textit{the gambler wins}] = \begin{cases} \dfrac{n}{T} & \text{for } p = \dfrac{1}{2}, \\ \dfrac{r^n - 1}{r^T - 1} & \text{for } p \neq \dfrac{1}{2}, \end{cases} \tag{20.1.2}$$

where $r ::= q/p$.

Recurrence for the Probability of Winning

Fortunately, you don't need to be as ingenuious Pascal in order to handle Gambler's Ruin, because linear recurrences offer a methodical approach to the basic problems.

The probability that the gambler wins is a function of his initial capital, n, his target, $T \geq n$, and the probability, p, that he wins an individual one dollar bet. For fixed p and T, let w_n be the gambler's probability of winning when his initial capital is n dollars. For example, w_0 is the probability that the gambler will win given that he starts off broke and w_T is the probability he will win if he starts off with his target amount, so clearly

$$w_0 = 0, \tag{20.1.3}$$

$$w_T = 1. \tag{20.1.4}$$

Otherwise, the gambler starts with n dollars, where $0 < n < T$. Now suppose the gambler wins his first bet. In this case, he is left with $n+1$ dollars and becomes a winner with probability w_{n+1}. On the other hand, if he loses the first bet, he is left with $n-1$ dollars and becomes a winner with probability w_{n-1}. By the Total Probability Rule, he wins with probability $w_n = pw_{n+1} + qw_{n-1}$. Solving for w_{n+1} we have

$$w_{n+1} = \frac{w_n}{p} - rw_{n-1} \tag{20.1.5}$$

where r is q/p in Section 20.1.1.

This recurrence holds only for $n+1 \leq T$, but there's no harm in using (20.1.5) to define w_{n+1} for all $n+1 > 1$. Now, letting

$$W(x) ::= w_0 + w_1 x + w_2 x + \cdots$$

be the generating function for the w_n, we derive from (20.1.5) and (20.1.3) using our generating function methods that

$$W(x) = \frac{w_1 x}{rx^2 - x/p + 1}. \tag{20.1.6}$$

But it's easy to check that the denominator factors:

$$rx^2 - \frac{x}{p} + 1 = (1-x)(1-rx).$$

Now if $p \neq q$, then using partial fractions we conclude that

$$W(x) = \frac{A}{1-x} + \frac{B}{1-rx}, \tag{20.1.7}$$

for some constants A, B. To solve for A, B, note that by (20.1.6) and (20.1.7),

$$w_1 x = A(1-rx) + B(1-x),$$

so letting $x = 1$, we get $A = w_1/(1-r)$, and letting $x = 1/r$, we get $B = w_1/(r-1)$. Therefore,

$$W(x) = \frac{w_1}{r-1}\left(\frac{1}{1-rx} - \frac{1}{1-x}\right),$$

which implies

$$w_n = w_1 \frac{r^n - 1}{r - 1}. \tag{20.1.8}$$

Finally, we can use (20.1.8) to solve for w_1 by letting $n = T$ to get

$$w_1 = \frac{r - 1}{r^T - 1}.$$

Plugging this value of w_1 into (20.1.8), we arrive at the solution:

$$w_n = \frac{r^n - 1}{r^T - 1},$$

matching Pascal's result (20.1.1).

In the unbiased case where $p = q$, we get from (20.1.6) that

$$W(x) = \frac{w_1 x}{(1 - x)^2},$$

and again can use partial fractions to match Pascal's result (20.1.2).

simpler expression for the biased case

The expression (20.1.1) for the probability that the Gambler wins in the biased game is a little hard to interpret. There is a simpler upper bound which is nearly tight when the gambler's starting capital is large and the game is biased *against* the gambler. Then $r > 1$, both the numerator and denominator in (20.1.1) are positive, and the numerator is smaller. This implies that

$$w_n < \frac{r^n}{r^T} = \left(\frac{1}{r}\right)^{T-n}$$

and gives:

Corollary 20.1.2. In the Gambler's Ruin game with initial capital, n, target, T, and probability $p < 1/2$ of winning each individual bet,

$$\Pr[\textit{the gambler wins}] < \left(\frac{1}{r}\right)^{T-n} \tag{20.1.9}$$

where $r ::= q/p > 1$.

So the gambler gains his intended profit before going broke with probability at most $1/r$ raised to the intended profit power. Notice that this upper bound does not depend on the gambler's starting capital, but only on his intended profit. This has the amazing consequence we announced above: *no matter how much money he starts with*, if he makes \$1 bets on red in roulette aiming to win \$100, the probability that he wins is less than

$$\left(\frac{18/38}{20/38}\right)^{100} = \left(\frac{9}{10}\right)^{100} < \frac{1}{37,648}.$$

The bound (20.1.9) decreases exponentially as the intended profit increases. So, for example, doubling his intended profit will square his probability of winning. In this case, the probability that the gambler's stake goes up 200 dollars before he goes broke playing roulette is at most

$$(9/10)^{200} = ((9/10)^{100})^2 < \left(\frac{1}{37,648}\right)^2,$$

which is about 1 in 1.4 billion.

Intuition

Why is the gambler so unlikely to make money when the game is only slightly biased against him? To answer this intuitively, we can identify two forces at work on the gambler's wallet. First, the gambler's capital has random upward and downward *swings* from runs of good and bad luck. Second, the gambler's capital will have a steady, downward *drift*, because the negative bias means an average loss of a few cents on each \$1 bet. The situation is shown in Figure 20.2.

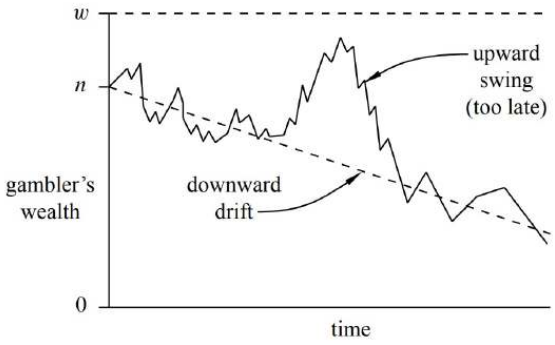


Figure 20.2 In a biased random walk, the downward drift usually dominates swings of good luck.

Our intuition is that if the gambler starts with, say, a billion dollars, then he is sure to play for a very long time, so at some point there should be a lucky, upward swing that puts him \$100 ahead. But his capital is steadily drifting downward. If the gambler does not have a lucky, upward swing early on, then he is doomed. After his capital drifts downward by tens and then hundreds of dollars, the size of the upward swing the gambler needs to win grows larger and larger. And as the size of the required swing grows, the odds that it occurs decrease exponentially. As a rule of thumb, *drift dominates swings* in the long term.

We can quantify these drifts and swings. After k rounds for $k \leq \min(m, n)$, the number of wins by our player has a binomial distribution with parameters $p < 1/2$ and k. His expected win on any single bet is $p - q = 2p - 1$ dollars, so his expected capital is $n - k(1-2p)$. Now to be a winner, his actual number of wins must exceed the expected number by $m + k(1-2p)$. But from p the formula (19.3.10), the binomial distribution has a standard deviation of only $\sqrt{kp(1-p)}$. So for the gambler to win, he needs his number of wins to deviate by

$$\frac{m + k(1-2p)}{\sqrt{kp(1-2p)}} = \Theta(\sqrt{k})$$

times its standard deviation. In our study of binomial tails, we saw that this was extremely unlikely.

In a fair game, there is no drift; swings are the only effect. In the absence of downward drift, our earlier intuition is correct. If the gambler starts with a trillion dollars then almost certainly there will eventually be a lucky swing that puts him \$100 ahead.

How Long a Walk?

Now that we know the probability, w_n, that the gambler is a winner in both fair and unfair games, we consider how many bets he needs on average to either win or go broke. A linear recurrence approach works here as well.

For fixed p and T, let e_n be the expected number of bets until the game ends when the gambler's initial capital is n dollars. Since the game is over in zero steps if $n = 0$ or T, the boundary conditions this time are $e_0 = e_T = 0$.

Otherwise, the gambler starts with n dollars, where $0 < n < T$. Now by the conditional expectation rule, the expected number of steps can be broken down into the expected number of steps given the outcome of the first bet weighted by the probability of that outcome. But after the gambler wins the first bet, his capital is $n+1$, so he can expect to make another e_{n+1} bets. That is,

$$\mathrm{Ex}[e_n \mid \text{gambler wins first bet}] = 1 + e_{n+1}.$$

Similarly, after the gambler loses his first bet, he can expect to make another en1 bets:

$$\mathrm{Ex}[e_n \mid \text{gambler loses first bet}] = 1 + e_{n-1}.$$

So we have

$$\begin{aligned} e_n &= p\mathrm{Ex}[e_n \mid \text{gambler wins first bet}] + q\mathrm{Ex}[e_n \mid \text{gambler loses first bet}] \\ &= p(1 + e_{n+1}) + q(1 + e_{n-1}) = pe_{n+1} + qe_{n-1} + 1. \end{aligned} \tag{20.1.10}$$

This yields the linear recurrence

$$e_{n+1} = \frac{1}{p}e_n - \frac{q}{p}e_{n-1} - \frac{1}{p}. \tag{20.1.11}$$

The routine solution of this linear recurrence yields:

Theorem 20.1.3

In the Gambler's Ruin game with initial capital n, target T, and probability p of winning 8 each bet,

$$\mathbf{Ex}[\textit{number of bets}] = \tag{20.1.12}$$
$$\begin{cases} n(T-n) & \textit{for } p = \dfrac{1}{2}, \\ \dfrac{w_n \cdot T - n}{p-q} & \textit{for } p \neq \dfrac{1}{2} \end{cases} \quad \textit{where } w_n = (r^n - 1)/(r^T - 1) = \Pr[\textit{the gambler wins}].$$

In the unbiased case, (20.1.12) can be rephrased simply as

$$\mathbf{Ex}[\text{number of fair bets}] = \text{initial capital} \cdot \text{intended profit} \tag{20.1.13}$$

For example, if the gambler starts with \$10 dollars and plays until he is broke or ahead \$10, then $10 \cdot 10 = 100$ bets are required on average. If he starts with \$500 and plays until he is broke or ahead \$100, then the expected number of bets until the game is over is $500 \times 100 = 50,000$. This simple formula (20.1.13) cries out for an intuitive proof, but we have not found one (where are you, Pascal?).

Quit While You Are Ahead

Suppose that the gambler never quits while he is ahead. That is, he starts with $n > 0$ dollars, ignores any target T, but plays until he is flat broke. Call this the *unbounded Gambler's ruin game*. It turns out that if the game is not favorable, that is, $p \leq 1/2$, the gambler is sure to go broke. In particular, this holds in an unbiased game with $p = 1/2$.

Lemma 20.1.4. *If the gambler starts with one or more dollars and plays a fair unbounded game, then he will go broke with probability 1.*

Proof. If the gambler has initial capital n and goes broke in a game without reaching a target T, then he would also go broke if he were playing and ignored the target. So the probability that he will lose if he keeps playing without stopping at any target T must be at least as large as the probability that he loses when he has a target $T > n$.

But we know that in a fair game, the probability that he loses is $1 - n/T$. This number can be made arbitrarily close to 1 by choosing a sufficiently large value of T. Hence, the probability of his losing while playing without any target has a lower bound arbitrarily close to 1, which means it must in fact be 1. ■

So even if the gambler starts with a million dollars and plays a perfectly fair game, he will eventually lose it all with probability 1. But there is good news: if the game is fair, he can "expect" to play forever:

Lemma 20.1.5. *If the gambler starts with one or more dollars and plays a fair unbounded game, then his expected number of plays is infinite.*

A proof appears in Problem 20.2.

So even starting with just one dollar, the expected number of plays before going broke is infinite! This sounds reassuring—you can go about your business without worrying about being doomed, because doom will be infinitely delayed. To illustrate a situation where you really needn't worry, think about mean time to failure with a really tiny probability of failure in any given second—say 10^{-100}. In this case you are unlikely to fail any time much sooner than many lifetimes of the estimated age of the universe, even though you will eventually fail with probability one.

But in general, you shouldn't feel reassured by an infinite expected time to go broke. For example, think about a variant Gambler's Ruin game which works as follows: run one second of the process that has a 10^{-100} of failing in any second. If it does *not* fail, then you go broke immediately. Otherwise, you play a fair, unbounded Gambler's Ruin game. Now there is an overwhelming probability, $1 - 10^{-100}$, that you will go broke immediately. But there is a 10^{-100} probability that you will wind up playing fair

Gambler's Ruin, so your overall expected time will be at least 10^{-100} times the expectation of fair Gambler's Ruin, namely, it will still be infinite.

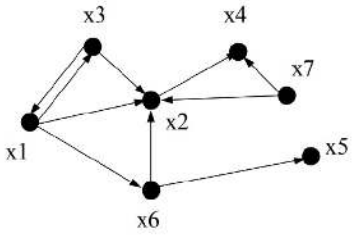
For the actual fair, unbounded Gambler's Ruin gain starting with one dollar, there is a a 50% chance the Gambler will go broke after the first bet, and a more than $15/16$ chance of going broke within five bets, for example. So infinite expected time is not much consolation to a Gambler who goes broke quickly with high probability.

[1]Here we're legitimately appealing to infinite linearity, since the payoff amounts remain bounded independent of the number of bets.

This page titled 20.1: Gambler's Ruin is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

20.2: Random Walks on Graphs

The hyperlink structure of the World Wide Web can be described as a digraph. The vertices are the web pages with a directed edge from vertex x to vertex y if x has a link to y. For example, in the following graph the vertices $x_1, \ldots, x_n$ correspond to web pages and $\langle x_i \to x_j \rangle$ is a directed edge when page x_i contains a hyperlink to x_j.



The web graph is an enormous graph with trillions of vertices. In 1995, two students at Stanford, Larry Page and Sergey Brin, realized that the structure of this graph could be very useful in building a search engine. Traditional document searching programs had been around for a long time and they worked in a fairly straightforward way. Basically, you would enter some search terms and the searching program would return all documents containing those terms. A relevance score might also be returned for each document based on the frequency or position that the search terms appeared in the document. For example, if the search term appeared in the title or appeared 100 times in a document, that document would get a higher score.

This approach works fine if you only have a few documents that match a search term. But on the web, there are many billions of documents and millions of matches to a typical search. For example, on May 2, 2012, a search on Google for " 'Mathematics for Computer Science' text" gave 482,000 hits! Which ones should we look at first? Just because a page gets a high keyword score—say because it has "Mathematics Mathematics ... Mathematics" copied 200 times across the front of the document—does not make it a great candidate for attention. The web is filled with bogus websites that repeat certain words over and over in order to attract visitors.

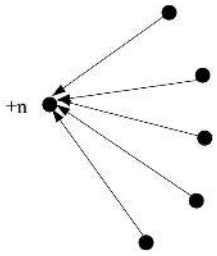
Google's enormous market capital in part derives from the revenue it receives from advertisers paying to appear at the top of search results. That top placement would not be worth much if Google's results were as easy to manipulate as keyword frquencies. Advertisers pay because Google's ranking method is consistently good at determining the most relevant web pages. For example, Google demonstrated its accuracy in our case by giving first rank[2] to our 6.042 text.

So how did Google know to pick our text to be first out of 482,000?—because back in 1995 Larry and Sergey got the idea to allow the digraph structure of the web to determine which pages are likely to be the most important.

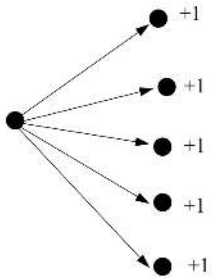
First Crack at Page Rank

Looking at the web graph, do you have an idea which vertex/page might be the best to rank first? Assume that all the pages match the search terms for now. Well, intuitively, we should choose x_2, since lots of other pages point to it. This leads us to their first idea: try defining the *page rank* of x to be $\text{indegree}(x)$, the number of links pointing to x. The idea is to think of web pages as voting for the most important page—the more votes, the better the rank.

Unfortunately, there are some problems with this idea. Suppose you wanted to have your page get a high ranking. One thing you could do is to create lots of dummy pages with links to your page.



There is another problem—a page could become unfairly influential by having lots of links to other pages it wanted to hype



So this strategy for high ranking would amount to, "vote early, vote often," which is no good if you want to build a search engine that's worth paying fees for. So, admittedly, their original idea was not so great. It was better than nothing, but certainly not worth billions of dollars.

Random Walk on the Web Graph

But then Sergey and Larry thought some more and came up with a couple of improvements. Instead of just counting the indegree of a vertex, they considered the probability of being at each page after a long random walk on the web graph. In particular, they decided to model a user's web experience as following each link on a page with uniform probability. For example, if the user is at page x, and there are three links from page x, then each link is followed with probability $1/3$. More generally, they assigned each edge $x \to y$ of the web graph with a probability conditioned on being on page x:

$$\Pr[\text{follow link } \langle x \to y \rangle \mid \text{ at page } x] ::= \frac{1}{\text{outdeg}(x)}.$$

The simulated user experience is then just a random walk on the web graph.

We can also compute the probability of arriving at a particular page, y, by summing over all edges pointing to y. We thus have

$$\begin{aligned} \Pr[\text{go to } y] &= \sum_{\text{edges } \langle x \to y \rangle} \Pr[\text{follow link } \langle x \to y \rangle \mid \text{ at page } x] \cdot \Pr[\text{at page } x] \\ &= \sum_{\text{edges } \langle x \to y \rangle} \frac{\Pr[\text{at } x]}{\text{outdeg}(x)} \end{aligned} \tag{20.2.1}$$

For example, in our web graph, we have

$$\Pr[\text{go to } x_4] = \frac{\Pr[\text{at } x_7]}{2} + \frac{\Pr[\text{at } x_2]}{1}.$$

One can think of this equation as x_7 sending half its probability to x_2 and the other half to x_4. The page x_2 sends all of its probability to x_4.

There's one aspect of the web graph described thus far that doesn't mesh with the user experience—some pages have no hyperlinks out. Under the current model, the user cannot escape these pages. In reality, however, the user doesn't fall off the end of the web into a void of nothingness. Instead, he restarts his web journey. Moreover, even if a user does not get stuck at a dead end, they will commonly get discouraged after following some unproductive path for a while and will decide to restart.

To model this aspect of the web, Sergey and Larry added a *supervertex* to the web graph and added an edge from every page to the supervertex. Moreover, the supervertex points to every other vertex in the graph with equal probability, allowing the walk to restart from a random place. This ensures that the graph is strongly connected.

If a page had no hyperlinks, then its edge to the supervertex has to be assigned probability one. For pages that had some hyperlinks, the additional edge pointing to the supervertex was assigned some specially given probability. In the original versions of Page Rank, this probability was arbitrarily set to 0.15. That is, each vertex with outdegree $n \geq 1$ got an additional edge pointing to the supervertex with assigned probability 0.15; its other n outgoing edges were still kept equally likely, that is, each of the n edges was assigned probability $0.85/n$.

Stationary Distribution & Page Rank

The basic idea behind page rank is finding a stationary distribution over the web graph, so let's define a stationary distribution.

Suppose each vertex is assigned a probability that corresponds, intuitively, to the likelihood that a random walker is at that vertex at a randomly chosen time. We assume that the walk never leaves the vertices in the graph, so we require that

$$\sum_{\text{vertices } x} \Pr[\text{at } x] = 1. \tag{20.2.2}$$

Definition 20.2.1

An assignment of probabilities to vertices in a digraph is a *stationary distribution* if for all vertices x

$$\Pr[\text{at } x] = \Pr[\text{go to } x \text{ at next step}]$$

Sergey and Larry defined their page ranks to be a stationary distribution. They did this by solving the following system of linear equations: find a nonnegative number, $\mathbf{Rank}(x)$, for each vertex, x, such that

$$\mathbf{Rank}(x) = \sum_{\text{edges } \langle y \to x \rangle} \frac{\mathbf{Rank}(y)}{\mathbf{outdeg}(y)} \tag{20.2.3}$$

corresponding to the intuitive equations given in (20.2.1). These numbers must also satisfy the additional constraint corresponding to (20.2.2):

$$\sum_{\text{vertices } x} \mathbf{Rank}(x) = 1. \tag{20.2.4}$$

So if there are n vertices, then equations (20.2.3) and (20.2.4) provide a system of $n+1$ linear equations in the n variables, $\mathbf{Rank}(x)$. Note that constraint (20.2.4) is needed because the remaining constraints (20.2.3) could be satisfied by letting $\mathbf{Rank}(x) ::= 0$ for all x, which is useless.

Sergey and Larry were smart fellows, and they set up their page rank algorithm so it would always have a meaningful solution. Strongly connected graphs have *unique* stationary distributions (Problem 20.12), and their addition of a supervertex ensures this. Moreover, starting from *any* vertex and taking a sufficiently long random walk on the graph, the probability of being at each page will get closer and closer to the stationary distribution. Note that general digraphs without super-vertices may have neither of these properties: there may not be a unique stationary distribution, and even when there is, there may be starting points from which the probabilities of positions during a random walk do not converge to the stationary distribution (Problem 20.8).

Now just keeping track of the digraph whose vertices are trillions of web pages is a daunting task. That's why in 2011 Google invested $168,000,000 in a solar power plant—the electrical power drawn by Google's servers in 2011 would have supplied the needs of 200,000 households.[3] Indeed, Larry and Sergey named their system Google after the number 10^{100}—which is called a "googol"—to reflect the fact that the web graph is so enormous.

Anyway, now you can see how this text ranked first out of 378,000 matches. Lots of other universities used our notes and presumably have links to the MIT Mathematics for Computer Science Open Course Ware site, and the university sites themselves are legitimate, which ultimately leads to the text getting a high page rank in the web graph.

[2]First rank for some reason was an early version archived at Princeton; the Spring 2010 version on the MIT Open Courseware site ranked 4th and 5th.

[3]*Google Details, and Defends, Its Use of Electricity, New York Times, September 8, 2011.*

This page titled 20.2: Random Walks on Graphs is shared under a CC BY-NC-SA license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

SECTION OVERVIEW

5: Recurrences

21: Recurrences

21.1: The Towers of Hanoi

21.2: Merge Sort

21.3: Linear Recurrences

21.4: Divide-and-Conquer Recurrences

21.5: A Feel for Recurrences

This page titled 5: Recurrences is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

CHAPTER OVERVIEW

21: Recurrences

21.1: The Towers of Hanoi

21.2: Merge Sort

21.3: Linear Recurrences

21.4: Divide-and-Conquer Recurrences

21.5: A Feel for Recurrences

This page titled 21: Recurrences is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Eric Lehman, F. Thomson Leighton, & Alberty R. Meyer (MIT OpenCourseWare) .

21.1: The Towers of Hanoi

There are several methods for solving recurrence equations. The simplest is to *guess* the solution and then *verify* that the guess is correct with an induction proof.

For example, as a alternative to the generating function derivation in Section 15.4.2 of the value of the number, T_n, of moves in the Tower of Hanoi problem with n disks, we could have tried guessing. As a basis for a good guess, let's look for a pattern in the values of T_n computed above: 1, 3, 7, 15, 31, 63. A natural guess is $T_n = 2^n - 1$. But whenever you guess a solution to a recurrence, you should always verify it with a proof, typically by induction. After all, your guess might be wrong. (But why bother to verify in this case? After all, if we're wrong, its not the end of the... no, let's check.)

Claim 21.1.1. $T_n = 2^n - 1$ satisfies the recurrence:

$$\begin{aligned} T_1 &= 1 \\ T_n &= 2T_{n-1} + 1 \qquad (for\ n \geq 2). \end{aligned} \tag{21.1.1}$$

Proof. The proof is by induction on n. The induction hypothesis is that $T_n = 2^n - 1$. This is true for $n = 1$ because $T_1 = 1 = 2^1 - 1$. Now assume that $T_{n-1} = 2^{n-1} - 1$ in order to prove that $T_n = 2^n - 1$, where $n \geq 2$:

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &= 2(2^{n-1} - 1) + 1 \\ &= 2^n - 1. \end{aligned} \tag{21.1.2}$$

The first equality is the recurrence equation, the second follows from the induction assumption, and the last step is simplification. ■

Such verification proofs are especially tidy because recurrence equations and induction proofs have analogous structures. In particular, the base case relies on the first line of the recurrence, which defines T_1. And the inductive step uses the second line of the recurrence, which defines T_n as a function of preceding terms.

Our guess is verified. So we can now resolve our remaining questions about the 64-disk puzzle. Since $T_{64} = 2^{64} - 1$, the monks must complete more than 18 billion billion steps before the world ends. Better study for the final

The Upper Bound Trap

When the solution to a recurrence is complicated, one might try to prove that some simpler expression is an upper bound on the solution. For example, the exact solution to the Towers of Hanoi recurrence is $T_n = 2^n - 1$. Let's try to prove the "nicer" upper bound $T_n \leq 2^n$, proceeding exactly as before.

Proof. (Failed attempt.) The proof is by induction on n. The induction hypothesis is that $T_n \leq 2^n$. This is true for $n = 1$ because $T_1 = 1 \leq 2^1$. Now assume that $T_{n-1} \leq 2^{n-1}$ in order to prove that $T_n \leq 2^n$, where $n \geq 2$:

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &\leq 2(2^{n-1}) + 1 \\ &\not\leq 2^n \qquad \text{IMPLIES Uh-oh!} \end{aligned} \tag{21.1.3}$$

The first equality is the recurrence relation, the second follows from the induction hypothesis, and the third step is a flaming train wreck. ■

The proof doesn't work! As is so often the case with induction proofs, the argument only goes through with a *stronger* hypothesis. This isn't to say that upper bounding the solution to a recurrence is hopeless, but this is a situation where induction and recurrences do not mix well.

Plug and Chug

Guess-and-verify is a simple and general way to solve recurrence equations. But there is one big drawback: you have to *guess right*. That was not hard for the Towers of Hanoi example. But sometimes the solution to a recurrence has a strange form that is quite difficult to guess. Practice helps, of course, but so can some other methods.

Plug-and-chug is another way to solve recurrences. This is also sometimes called “expansion” or “iteration.” As in guess-and-verify, the key step is identifying a pattern. But instead of looking at a sequence of *numbers*, you have to spot a pattern in a sequence of *expressions*, which is sometimes easier. The method consists of three steps, which are described below and illustrated with the Towers of Hanoi example.

Step 1: Plug and Chug Until a Pattern Appears

The first step is to expand the recurrence equation by alternately “plugging” (applying the recurrence) and “chugging” (simplifying the result) until a pattern appears. Be careful: too much simplification can make a pattern harder to spot. The rule to remember—indeed, a rule applicable to the whole of college life—is *chug in moderation*.

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &= 2(2T_{n-2} + 1) + 1 && \text{plug} \\ &= 4T_{n-2} + 2 + 1 && \text{chug} \\ &= 4(2T_{n-3} + 1) + 2 + 1 && \text{plug} \\ &= 8T_{n-3} + 4 + 2 + 1 && \text{chug} \\ &= 8(2T_{n-4} + 1) + 4 + 2 + 1 && \text{plug} \\ &= 16T_{n-4} + 8 + 4 + 2 + 1 && \text{chug} \end{aligned} \tag{21.1.4}$$

Above, we started with the recurrence equation. Then we replaced T_{n-1} with $2T_{n-2} + 1$, since the recurrence says the two are equivalent. In the third step, we simplified a little—but not too much! After several similar rounds of plugging and chugging, a pattern is apparent. The following formula seems to hold:

$$\begin{aligned} T_n &= 2^k T_{n-k} + 2^{k-1} + 2^{k-2} + \cdots + 2^2 + 2^1 + 2^0 \\ &= 2^k T_{n-k} + 2^k - 1 \end{aligned} \tag{21.1.5}$$

Once the pattern is clear, simplifying is safe and convenient. In particular, we’ve collapsed the geometric sum to a closed form on the second line.

Step 2: Verify the Pattern

The next step is to verify the general formula with one more round of plug-andchug.

$$\begin{aligned} T_n &= 2^k T_{n-k} + 2^k - 1 \\ &= 2^k(2T_{n-(k+1)} + 1) + 2^k - 1 && \text{plug} \\ &= 2^{k+1} T_{n-(k+1)} + 2^{k+1} - 1 && \text{chug} \end{aligned} \tag{21.1.6}$$

The final expression on the right is the same as the expression on the first line, except that k is replaced by $k+1$. Surprisingly, this effectively *proves* that the formula is correct for all k. Here is why: we know the formula holds for $k = 1$, because that’s the original recurrence equation. And we’ve just shown that if the formula holds for some $k \geq 1$, then it also holds for $k+1$. So the formula holds for all $k \geq 1$ by induction.

Step 3: Write T_n Using Early Terms with Known Values

The last step is to express T_n as a function of early terms whose values are known. Here, choosing $k = n-1$ expresses T_n in terms of T_1, which is equal to 1. Simplifying gives a closed-form expression for T_n:

$$\begin{aligned} T_n &= 2^{n-1} T_1 + 2^{n-1} - 1 \\ &= 2^{n-1} \cdot 1 + 2^{n-1} - 1 \\ &= 2^n - 1. \end{aligned} \tag{21.1.7}$$

We’re done! This is the same answer we got from guess-and-verify.

Let’s compare guess-and-verify with plug-and-chug. In the guess-and-verify method, we computed several terms at the beginning of the sequence, T_1, T_2, T_3, etc., until a pattern appeared. We generalized to a formula for the nth term, T_n. In contrast, plug-and-chug works backward from the nth term. Specifically, we started with an expression for T_n involving the preceding term, T_{n-1}, and rewrote this using progressively earlier terms, T_{n-2}, T_{n-3}, etc. Eventually, we noticed a pattern, which allowed us to express T_n using the very first term, T_1, whose value we knew. Substituting this value gave a closed-form expression for T_n. So guess-and-verify and plug-and-chug tackle the problem from opposite directions.

21.1: The Towers of Hanoi is shared under a not declared license and was authored, remixed, and/or curated by LibreTexts.

21.2: Merge Sort

Algorithms textbooks traditionally claim that sorting is an important, fundamental problem in computer science. Then they smack you with sorting algorithms until life as a disk-stacking monk in Hanoi sounds delightful. Here, we'll cover just *one* well-known sorting algorithm, *Merge Sort*. The analysis introduces another kind of recurrence.

Here is how Merge Sort works. The input is a list of n numbers, and the output is those same numbers in nondecreasing order. There are two cases:

- If the input is a single number, then the algorithm does nothing, because the list is already sorted.
- Otherwise, the list contains two or more numbers. The first half and the second half of the list are each sorted recursively. Then the two halves are merged to form a sorted list with all n numbers.

Let's work through an example. Suppose we want to sort this list:

$$10, 7, 23, 5, 2, 8, 6, 9.$$

Since there is more than one number, the first half (10, 7, 23, 5) and the second half (2, 8, 6, 9) are sorted recursively. The results are 5, 7, 10, 23 and 2, 6, 8, 9. All that remains is to merge these two lists. This is done by repeatedly emitting the smaller of the two leading terms. When one list is empty, the whole other list is emitted. The example is worked out below. In this table, underlined numbers are about to be emitted.

First Half	Second Half	Output
5, 7, 10, 23	2, 6, 8, 9	
5, 7, 10, 23	6, 8, 9	2
7, 10, 23	6, 8, 9	2, 5
7, 10, 23	8, 9	2, 5, 6
10, 23	8, 9	2, 5, 6, 7
10, 23	9	2, 5, 6, 7, 8
10, 23		2, 5, 6, 7, 8, 9
		2, 5, 6, 7, 8, 9, 10, 23

The leading terms are initially 5 and 2. So we output 2. Then the leading terms are 5 and 6, so we output 5. Eventually, the second list becomes empty. At that point, we output the whole first list, which consists of 10 and 23. The complete output consists of all the numbers in sorted order.

Finding a Recurrence

A traditional question about sorting algorithms is, "What is the maximum number of comparisons used in sorting n items?" This is taken as an estimate of the running time. In the case of Merge Sort, we can express this quantity with a recurrence. Let T_n be the maximum number of comparisons used while Merge Sorting a list of n numbers. For now, assume that n is a power of 2. This ensures that the input can be divided in half at every stage of the recursion.

- If there is only one number in the list, then no comparisons are required, so $T_1 = 0$.
- Otherwise, T_n includes comparisons used in sorting the first half (at most $T_{n/2}$), in sorting the second half (also at most $T_{n/2}$), and in merging the two halves. The number of comparisons in the merging step is at most $n-1$. This is because at least one number is emitted after each comparison and one more number is emitted at the end when one list becomes empty. Since n items are emitted in all, there can be at most $n-1$ comparisons.

Therefore, the maximum number of comparisons needed to Merge Sort n items is given by this recurrence:

$$\begin{aligned} T_1 &= 0 \\ T_n &= 2T_{n/2} + n - 1 \qquad (\text{for } n \geq 2 \text{ and a power of } 2). \end{aligned} \tag{21.2.1}$$

This fully describes the number of comparisons, but not in a very useful way; a closed-form expression would be much more helpful. To get that, we have to solve the recurrence.

Solving the Recurrence

Let's first try to solve the Merge Sort recurrence with the guess-and-verify technique. Here are the first few values:

$$\begin{aligned} T_1 &= 0 \\ T_2 &= 2T_1 + 2 - 1 = 1 \\ T_4 &= 2T_2 + 4 - 1 = 5 \\ T_8 &= 2T_4 + 8 - 1 = 17 \\ T_{16} &= 2T_8 + 16 - 1 = 49. \end{aligned} \tag{21.2.2}$$

We're in trouble! Guessing the solution to this recurrence is hard because there is no obvious pattern. So let's try the plug-and-chug method instead.

Step 1: Plug and Chug Until a Pattern Appears

First, we expand the recurrence equation by alternately plugging and chugging until a pattern appears.

$$\begin{aligned} T_n &= 2T_{n/2} + n - 1 \\ &= 2(2T_{n/4} + n/2 - 1) + (n-1) && \text{plug} \\ &= 4T_{n/4} + (n-2) + (n-1) && \text{chug} \\ &= 4(2T_{n/8} + n/4 - 1) + (n-2) + (n-1) && \text{plug} \\ &= 8T_{n/8} + (n-4) - (n-2) + (n-1) && \text{chug} \\ &= 8(2T_{n/16} + n/8 - 1) + (n-4) - (n-2) + (n-1) && \text{plug} \\ &= 16T_{n/16} + (n-8) + (n-4) - (n-2) + (n-1) && \text{chug} \end{aligned} \tag{21.2.3}$$

A pattern is emerging. In particular, this formula seems holds:

$$\begin{aligned} T_n &= 2^k T_{n/2^k} + (n - 2^{k-1}) + (n - 2^{k-2}) + \cdots (n - 2^0) \\ &= 2^k T_{n/2^k} + kn - 2^{k-1} - 2^{k-2} \cdots - 2^0 \\ &= 2^k T_{n/2^k} + kn - 2^k + 1. \end{aligned} \tag{21.2.4}$$

On the second line, we grouped the n terms and powers of 2. On the third, we collapsed the geometric sum.

Step 2: Verify the Pattern

Next, we verify the pattern with one additional round of plug-and-chug. If we guessed the wrong pattern, then this is where we'll discover the mistake.

$$\begin{aligned} T_n &= 2^k T_{n/2^k} + kn - 2^k + 1 \\ &= 2^k (2T_{n/2^{k+1}} + n/2^k - 1) + kn - 2^k + 1 && \text{plug} \\ &= 2^{k+1} T_{n/2^{k+1}} + (k+1)n - 2^{k+1} + 1. && \text{chug} \end{aligned} \tag{21.2.5}$$

The formula is unchanged except that k is replaced by $k+1$. This amounts to the induction step in a proof that the formula holds for all $k \geq 1$.

Step 3: Write T_n Using Early Terms with Known Values

Finally, we express T_n using early terms whose values are known. Specifically, if we let $k = \log n$, then $T_{n/2^k} = T_1$, which we know is 0:

$$\begin{aligned} T_n &= 2^k T_{n/2^k} + kn - 2^k + 1 \\ &= 2^{\log n} T_{n/2^{\log n}} + n \log n - 2^{\log n} + 1 \\ &= nT_1 + n \log n - n + 1 \\ &= n \log n - n + 1. \end{aligned} \tag{21.2.6}$$

We're done! We have a closed-form expression for the maximum number of comparisons used in Merge Sorting a list of n numbers. In retrospect, it is easy to see why guess-and-verify failed: this formula is fairly complicated.

As a check, we can confirm that this formula gives the same values that we computed earlier:

n	T_n	$n\log n - n + 1$
1	0	$1\log 1 - 1 + 1 = 0$
2	1	$2\log 2 - 2 + 1 = 1$
4	5	$4\log 4 - 4 + 1 = 5$
8	17	$8\log 8 - 8 + 1 = 17$
16	49	$16\log 16 - 16 + 1 = 49$

As a double-check, we could write out an explicit induction proof. This would be straightforward, because we already worked out the guts of the proof in step 2 of the plug-and-chug procedure.

21.2: Merge Sort is shared under a not declared license and was authored, remixed, and/or curated by LibreTexts.

21.3: Linear Recurrences

So far we've solved recurrences with two techniques: guess-and-verify and plugand-chug. These methods require spotting a pattern in a sequence of numbers or expressions. In this section and the next, we'll give cookbook solutions for two large classes of recurrences. These methods require no flash of insight; you just follow the recipe and get the answer

Climbing Stairs

How many different ways are there to climb n stairs, if you can either step up one stair or hop up two? For example, there are five different ways to climb four stairs:

1. step, step, step, step
2. hop, hop
3. hop, step, step
4. step, hop, step
5. step, step, hop

Working through this problem will demonstrate the major features of our first cookbook method for solving recurrences. We'll fill in the details of the general solution afterward.

Finding a Recurrence

As special cases, there is 1 way to climb 0 stairs (do nothing) and 1 way to climb 1 stair (step up). In general, an ascent of n stairs consists of either a step followed by an ascent of the remaining $n-1$ stairs or a hop followed by an ascent of $n-2$ stairs. So the total number of ways to climb n stairs is equal to the number of ways to climb $n-1$ plus the number of ways to climb $n-2$. These observations define a recurrence:

$$\begin{aligned} f(0) &= 1 \\ f(1) &= 1 \\ f(n) &= f(n-1) + f(n-2) \qquad \text{for } n \geq 2. \end{aligned} \tag{21.3.1}$$

Here, $f(n)$ denotes the number of ways to climb n stairs. Also, we've switched from subscript notation to functional notation, from T_n to f_n. Here the change is cosmetic, but the expressiveness of functions will be useful later.

This is the Fibonacci recurrence, the most famous of all recurrence equations. Fibonacci numbers arise in all sorts of applications and in nature. Fibonacci introduced the numbers in 1202 to study rabbit reproduction. Fibonacci numbers also appear, oddly enough, in the spiral patterns on the faces of sunflowers. And the input numbers that make Euclid's GCD algorithm require the greatest number of steps are consecutive Fibonacci numbers.

Solving the Recurrence

The Fibonacci recurrence belongs to the class of linear recurrences, which are essentially all solvable with a technique that you can learn in an hour. This is somewhat amazing, since the Fibonacci recurrence remained unsolved for almost six centuries!

In general, a *homogeneous* linear recurrence has the form

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_d f(n-d)$$

where $a_1, a_2, \ldots, a_d$ and d are constants. The *order* of the recurrence is d. Commonly, the value of the function f is also specified at a few points; these are called *boundary conditions*. For example, the Fibonacci recurrence has order $d = 2$ with coefficients $a_1 = a_2 = 1$ and $g(n) = 0$. The boundary conditions are $f(0) = 1$ and $f(1) = 1$. The word "homogeneous" sounds scary, but effectively means "the simpler kind." We'll consider linear recurrences with a more complicated form later.

Let's try to solve the Fibonacci recurrence with the benefit centuries of hindsight. In general, linear recurrences tend to have exponential solutions. So let's guess that

$$f(n) = x^n$$

where x is a parameter introduced to improve our odds of making a correct guess. We'll figure out the best value for x later. To further improve our odds, let's neglect the boundary conditions, $f(0) = 0$ and $f(1) = 1$, for now. Plugging this guess into the

recurrence $f(n) = f(n-1) + f(n-2)$ gives

$$x^n = x^{n-1} + x^{n-2}.$$

Dividing both sides by x^{n-2} leaves a quadratic equation:

$$x^2 = x + 1.$$

Solving this equation gives *two* plausible values for the parameter x:

$$x = \frac{1 \pm \sqrt{5}}{2}.$$

This suggests that there are at least two different solutions to the recurrence, neglecting the boundary conditions.

$$f(n) = \left(\frac{1+\sqrt{5}}{2}\right)^n \text{ or } f(n) = \left(\frac{1-\sqrt{5}}{2}\right)^n$$

A charming features of homogeneous linear recurrences is that any linear combination of solutions is another solution.

Theorem 21.3.1

If $f(n)$ and $g(n)$ are both solutions to a homogeneous linear recurrence, then $h(n) = sf(n) + tg(n)$ is also a solution for all $s, t \in \mathbb{R}$.

Proof

$$\begin{aligned} h(n) &= sf(n) + tg(n) \\ &= s(a_1 f(n-1) + \cdots + a_d f(n-d)) + t(a_1 g(n-1) + \cdots + a_d g(n-d)) \\ &= a_1(sf(n-1) + tg(n-1)) + \cdots a_d(sf(n-d) + tg(n-d)) \\ &= a_1 h(n-1) + \cdots + a_d h(n-d) \end{aligned} \tag{21.3.2}$$

The first step uses the definition of the function h, and the second uses the fact that f and g are solutions to the recurrence. In the last two steps, we rearrange terms and use the definition of h again. Since the first expression is equal to the last, h is also a solution to the recurrence. ■

The phenomenon described in this theorem—a linear combination of solutions is another solution—also holds for many differential equations and physical systems. In fact, linear recurrences are so similar to linear differential equations that you can safely snooze through that topic in some future math class.

Returning to the Fibonacci recurrence, this theorem implies that

$$f(n) = s\left(\frac{1+\sqrt{5}}{2}\right)^n + t\left(\frac{1-\sqrt{5}}{2}\right)^n$$

is a solution for all real numbers s and t. The theorem expanded two solutions to a whole spectrum of possibilities! Now, given all these options to choose from, we can find one solution that satisfies the boundary conditions, $f(0) = 1$ and $f(1) = 1$. Each boundary condition puts some constraints on the parameters s and t. In particular, the first boundary condition implies that

$$f(0) = s\left(\frac{1+\sqrt{5}}{2}\right)^0 + t\left(\frac{1-\sqrt{5}}{2}\right)^0 = s + t = 1.$$

Similarly, the second boundary condition implies that

$$f(1) = s\left(\frac{1+\sqrt{5}}{2}\right)^1 + t\left(\frac{1-\sqrt{5}}{2}\right)^1 = 1.$$

Now we have two linear equations in two unknowns. The system is not degenerate, so there is a unique solution:

$$s = \frac{1}{\sqrt{5}} \cdot \frac{1+\sqrt{5}}{2} \quad s = -\frac{1}{\sqrt{5}} \cdot \frac{1-\sqrt{5}}{2}.$$

These values of s and t identify a solution to the Fibonacci recurrence that also satisfies the boundary conditions:

$$\begin{aligned} f(n) &= \frac{1}{\sqrt{5}} \cdot \frac{1+\sqrt{5}}{2} \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \cdot \frac{1-\sqrt{5}}{2} \left(\frac{1-\sqrt{5}}{2}\right)^n \\ &= \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}. \end{aligned} \tag{21.3.3}$$

It is easy to see why no one stumbled across this solution for almost six centuries. All Fibonacci numbers are integers, but this expression is full of square roots of five! Amazingly, the square roots always cancel out. This expression really does give the Fibonacci numbers if we plug in $n = 0, 1, 2,$ etc.

This closed form for Fibonacci numbers is known as Binet's formula and has some interesting corollaries. p The first term tends to infinity because the base of the exponential, $(1+\sqrt{5})/2 = 1.618\ldots$ is greater than one. This value is often denoted ϕ and called the "golden ratio." The second term tends to zero, because $(1-\sqrt{5})/2 = -0.618033988\ldots$ has absolute value less than 1. This implies that the nth Fibonacci number is:

$$f(n) = \frac{\phi^{n+1}}{\sqrt{5}} + o(1).$$

Remarkably, this expression involving irrational numbers is actually very close to an integer for all large n—namely, a Fibonacci number! For example:

$$\frac{\phi^{20}}{\sqrt{5}} = 6765.000029\cdots \approx f(19).$$

This also implies that the ratio of consecutive Fibonacci numbers rapidly approaches the golden ratio. For example:

$$\frac{f(20)}{f(19)} = \frac{10946}{6765} = 1.618033998\ldots.$$

Solving Homogeneous Linear Recurrences

The method we used to solve the Fibonacci recurrence can be extended to solve any homogeneous linear recurrence; that is, a recurrence of the form

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots + a_d f(n-d)$$

where $a_1, a_2, \ldots, a_d$ and d are constants. Substituting the guess $f(n) = x_n$, as with the Fibonacci recurrence, gives

$$x^n = a_1 x^{n-1} + a_2 x^{n-2} + \cdots + a_d x^{n-d}.$$

Dividing by x^{n-d} gives

$$x^d = a_1 x^{d-1} + a_2 x^{d-2} + \cdots + a_{d-1} x + a_d.$$

This is called the *characteristic equation* of the recurrence. The characteristic equation can be read off quickly since the coefficients of the equation are the same as the coefficients of the recurrence.

The solutions to a linear recurrence are defined by the roots of the characteristic equation. Neglecting boundary conditions for the moment:

If r is a nonrepeated root of the characteristic equation, then r^n is a solution to the recurrence.

If r is a repeated root with multiplicity k then $r^n, nr^n, n^2 r^n, \ldots, n^{k-1} r^n$ are all solutions to the recurrence.

Theorem 21.3.1 implies that every linear combination of these solutions is also a solution.

For example, suppose that the characteristic equation of a recurrence has roots $s, t,$ and u twice. These four roots imply four distinct solutions:

$$f(n) = s^n \quad f(n) = t^n \quad f(n) = u^n \quad f(n) = nu^n.$$

Furthermore, every linear combination

$$f(n) = a \cdot s^n + b \cdot t^n + c \cdot u^n + d \cdot nu^n \tag{21.3.4}$$

is also a solution.

All that remains is to select a solution consistent with the boundary conditions by choosing the constants appropriately. Each boundary condition implies a linear equation involving these constants. So we can determine the constants by solving a system of linear equations. For example, suppose our boundary conditions were $f(0) = 0$, $f(1) = 1$, $f(2) = 4$, and $f(3) = 9$. Then we would obtain four equations in four unknowns:

$$\begin{array}{lll} f(0) = 0 & \text{implies} & a \cdot s^0 + b \cdot t^0 + c \cdot u^0 + d \cdot 0u^0 = 0 \\ f(1) = 1 & \text{implies} & a \cdot s^1 + b \cdot t^1 + c \cdot u^1 + d \cdot 0u^1 = 1 \\ f(2) = 4 & \text{implies} & a \cdot s^2 + b \cdot t^2 + c \cdot u^2 + d \cdot 0u^2 = 4 \\ f(3) = 9 & \text{implies} & a \cdot s^3 + b \cdot t^3 + c \cdot u^3 + d \cdot 0u^3 = 9 \end{array} \tag{21.3.5}$$

This looks nasty, but remember that s, t, and u are just constants. Solving this system gives values for a, b, c, and d that define a solution to the recurrence consistent with the boundary conditions.

Solving General Linear Recurrences

We can now solve all linear homogeneous recurrences, which have the form

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots a_d f(n-d).$$

Many recurrences that arise in practice do not quite fit this mold. For example, the Towers of Hanoi problem led to this recurrence:

$$\begin{array}{l} f(1) = 1 \\ f(n) = 2f(n-1) + 1 \qquad (\text{for } n \geq 2). \end{array} \tag{21.3.6}$$

The problem is the extra $+1$; that is not allowed in a homogeneous linear recurrence. In general, adding an extra function $g(n)$ to the right side of a linear recurrence gives an *inhomogeneous linear recurrence*:

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + \cdots a_d f(n-d) + g(n).$$

Solving inhomogeneous linear recurrences is neither very different nor very difficult. We can divide the whole job into five steps:

1. Replace $g(n)$ by 0, leaving a homogeneous recurrence. As before, find roots of the characteristic equation.
2. Write down the solution to the homogeneous recurrence, but do not yet use the boundary conditions to determine coefficients. This is called the *homogeneous solution*.
3. Now restore $g(n)$ and find a single solution to the recurrence, ignoring boundary conditions. This is called a *particular solution*. We'll explain how to find a particular solution shortly.
4. Add the homogeneous and particular solutions together to obtain the *general solution*.
5. Now use the boundary conditions to determine constants by the usual method of generating and solving a system of linear equations.

As an example, let's consider a variation of the Towers of Hanoi problem. Suppose that moving a disk takes time proportional to its size. Specifically, moving the smallest disk takes 1 second, the next-smallest takes 2 seconds, and moving the nth disk then requires n seconds instead of 1. So, in this variation, the time to complete the job is given by a recurrence with $a + n$ term instead of $a + 1$:

$$\begin{array}{l} f(1) = 1 \\ f(n) = 2f(n-1) + n \qquad (\text{for } n \geq 2). \end{array} \tag{21.3.7}$$

Clearly, this will take longer, but how much longer? Let's solve the recurrence with the method described above.

In Steps 1 and 2, dropping the $+n$ leaves the homogeneous recurrence $f(n) = 2f(n-1)$. The characteristic equation is $x = 2$. So the homogeneous solution is $f(n) = c2^n$.

In Step 3, we must find a solution to the full recurrence $f(n) = 2f(n-1) + n$, without regard to the boundary condition. Let's guess that there is a solution of the form $f(n) = an + b$ for some constants a and b. Substituting this guess into the recurrence gives

$$\begin{aligned} an + b &= 2(a(n-1) + b) + n \\ 0 &= (a+1)n + (b-2a). \end{aligned} \tag{21.3.8}$$

The second equation is a simplification of the first. The second equation holds for all n if both $a + 1 = 0$ (which implies $a = -1$) and $b - 2a = 0$ (which implies that $b = -2$). So $f(n) = an + b = n - 2$ is a particular solution.

In the Step 4, we add the homogeneous and particular solutions to obtain the general solution

$$f(n) = c2^n - n - 2.$$

Finally, in step 5, we use the boundary condition, $f(1) = 1$, determine the value of the constant c:

$$\begin{aligned} f(1) = 1 \quad &\text{IMPLIES} \quad c2^1 - 1 - 2 = 1 \\ &\text{IMPLIES} \quad c = 2. \end{aligned} \tag{21.3.9}$$

Therefore, the function $f(n) = 2 \cdot 2^n - n - 2$ solves this variant of the Towers of Hanoi recurrence. For comparison, the solution to the original Towers of Hanoi problem was $2^n - 1$. So if moving disks takes time proportional to their size, then the monks will need about twice as much time to solve the whole puzzle.

How to Guess a Particular Solution

Finding a particular solution can be the hardest part of solving inhomogeneous recurrences. This involves guessing, and you might guess wrong.[1] However, some rules of thumb make this job fairly easy most of the time.

- Generally, look for a particular solution with the same form as the inhomogeneous term $g(n)$.
- If $g(n)$ is a constant, then guess a particular solution $f(n) = c$. If this doesn't work, try polynomials of progressively higher degree: $f(n) = bn + c$, then $f(n) = an^2 + bn + c$, etc.
- More generally, if $g(n)$ is a polynomial, try a polynomial of the same degree, then a polynomial of degree one higher, then two higher, etc. For example, if $g(n) = 6n + 5$, then try $f(n) = bn + c$ and then $f(n) = an^2 + bn + c$.
- If $g(n)$ is an exponential, such as 3^n, then first guess that $f(n) = c3^n$. Failing that, try $f(n) = bn3^n + c3^n$ and then $an^2 3^n + bn3^n + c3^n$, etc.

The entire process is summarized on the following page.

[1]Chapter 15 explains how to solve linear recurrences with generating functions—it's a little more complicated, but it does not require guessing.

21.3: Linear Recurrences is shared under a not declared license and was authored, remixed, and/or curated by LibreTexts.

21.4: Divide-and-Conquer Recurrences

We now have a recipe for solving general linear recurrences. But the Merge Sort recurrence, which we encountered earlier, is not linear:

$$\begin{aligned} T(1) &= 0 \\ T(n) &= 2T(n/2) + n - 1 \qquad (\text{for } n \geq 2). \end{aligned} \tag{21.4.1}$$

In particular, $T(n)$ is not a linear combination of a fixed number of immediately preceding terms; rather, $T(n)$ is a function of $T(n/2)$, a term halfway back in the sequence.

Short Guide to Solving Linear Recurrences

A linear recurrence is an equation

$$f(n) = \underbrace{a_1 f(n-1) + a_2 f(n-2) + \cdots + a_d f(n-d)}_{\text{homogeneous part}} \quad \underbrace{+g(n)}_{\text{inhomogeneous part}}$$

together with boundary conditions such as $f(0) = b_0, f(1) = b_1$, etc. Linear recurrences are solved as follows:

1. Find the roots of the characteristic equation

$$x^n = a_1 x^{n-1} + a_2 x^{n-2} + \cdot + a_k x^{n-k}.$$

2. Write down the homogeneous solution. Each root generates one term and the homogeneous solution is their sum. A nonrepeated root r generates the term cr^n, where c is a constant to be determined later. A root r with multiplicity k generates the terms

$$d_1 r^n \quad d_2 n r^n \quad d_2 n^2 r^n \quad \ldots \quad d_k n^{k-1} r^n$$

where $d_1, \ldots, d_k$ are constants to be determined later.

3. Find a particular solution. This is a solution to the full recurrence that need not be consistent with the boundary conditions. Use guess-and-verify. If $g(n)$ is a constant or a polynomial, try a polynomial of the same degree, then of one higher degree, then two higher. For example, if $g(n) = n$, then try $f(n) = bn + c$ and then $an^2 + bn + c$. If $g(n)$ is an exponential, such as 3^n, then first guess $f(n) = c3^n$. Failing that, try $f(n) = (bn + c)3^n$ and then $(an^2 + bn + c)3^n$, etc.

4. Form the general solution, which is the sum of the homogeneous solution and the particular solution. Here is a typical general solution:

$$f(n) = \underbrace{c2^n + d(-1)^n}_{\text{homogeneous solution}} + \underbrace{3n+1}_{\text{inhomogeneous solution}}$$

5. Substitute the boundary conditions into the general solution. Each boundary condition gives a linear equation in the unknown constants. For example, substituting $f(1) = 2$ into the general solution above gives

$$\begin{aligned} 2 &= c \cdot 2^1 + d \cdot (-1)^1 + 3 \cdot 1 + 1 \\ \text{IMPLIES} \quad -2 &= 2c - d. \end{aligned} \tag{21.4.2}$$

Determine the values of these constants by solving the resulting system of linear equations.

Merge Sort is an example of a divide-and-conquer algorithm: it divides the input, “conquers” the pieces, and combines the results. Analysis of such algorithms commonly leads to *divide-and-conquer* recurrences, which have this form:

$$T(n) = \sum_{i=1}^{k} a_i T(b_i n) + g(n)$$

Here $a_1, \ldots, a_k$ are positive constants, $b_1, \ldots, b_k$ are constants between 0 and 1, and $g(n)$ is a nonnegative function. For example, setting $a_1 = 2, b_1 = 1/2$, and $g(n) = n - 1$ gives the Merge Sort recurrence.

The Akra-Bazzi Formula

The solution to virtually all divide and conquer solutions is given by the amazing *Akra-Bazzi formula*. Quite simply, the asymptotic solution to the general divideand-conquer recurrence

$$T(n) = \sum_{i=1}^{k} a_i T(b_i n) + g(n)$$

is

$$T(n) = \Theta\left(n^p \left(1 + \int_1^n \frac{g(u)}{u^{p+1}} du\right)\right) \tag{21.4.3}$$

where p satisfies

$$\sum_{i=1}^{k} a_i b_i^p = 1. \tag{21.4.4}$$

A rarely-troublesome requirement is that the function $g(n)$ must not grow or oscillate too quickly. Specifically, $|g'(n)|$ must be bounded by some polynomial. So, for example, the Akra-Bazzi formula is valid when $g(n) = x^2 \log n$, but not when $g(n) = 2^n$.

Let's solve the Merge Sort recurrence again, using the Akra-Bazzi formula instead of plug-and-chug. First, we find the value p that satisfies

$$2 \cdot (1/2)^p = 1.$$

Looks like $p = 1$ does the job. Then we compute the integral:

$$\begin{aligned} T(n) &= \Theta\left(n\left(1 + \int_1^n \frac{u-1}{u^2} du\right)\right) \\ &= \Theta\left(n\left(1 + \left[\log u + \frac{1}{u}\right]_1^n\right)\right) \\ &= \Theta\left(n\left(\log n + \frac{1}{n}\right)\right) \\ &= \Theta(n \log n) \end{aligned} \tag{21.4.5}$$

The first step is integration and the second is simplification. We can drop the $1/n$ term in the last step, because the $\log n$ term dominates. We're done!

Let's try a scary-looking recurrence:

$$T(n) = 2T(n/2) + (8/9)T(3n/4) + n^2.$$

Here, $a_1 = 2, b_1 = 1/2, a_2 = 8/9$, and $b_2 = 3/4$. So we find the value p that satisfies

$$2 \cdot (1/2)^p + (8/9)(3/4)^p = 1.$$

Equations of this form don't always have closed-form solutions, so you may need to approximate p numerically sometimes. But in this case the solution is simple: $p = 2$. Then we integrate:

$$\begin{aligned} T(n) &= \Theta\left(n^2 \left(1 + \int_1^n \frac{u^2}{u^3} du\right)\right) \\ &= \Theta(n^2(1 + \log n)) \\ &= \Theta(n^2 \log n). \end{aligned} \tag{21.4.6}$$

That was easy!

Two Technical Issues

Until now, we've swept a couple issues related to divide-and-conquer recurrences under the rug. Let's address those issues now.

First, the Akra-Bazzi formula makes no use of boundary conditions. To see why, let's go back to Merge Sort. During the plug-and-chug analysis, we found that

$$T_n = nT_1 + n \log n - n + 1.$$

This expresses the nth term as a function of the first term, whose value is specified in a boundary condition. But notice that $T_n = \Theta(n \log n)$ for *every* value of T_1. The boundary condition doesn't matter!

This is the typical situation: *the asymptotic solution to a divide-and-conquer recurrence is independent of the boundary conditions*. Intuitively, if the bottomlevel operation in a recursive algorithm takes, say, twice as long, then the overall running time will at most double. This matters in practice, but the factor of 2 is concealed by asymptotic notation. There are corner-case exceptions. For example, the solution to $T(n) = 2T(n/2)$ is either $\Theta(n)$ or zero, depending on whether $T(1)$ is zero. These cases are of little practical interest, so we won't consider them further.

There is a second nagging issue with divide-and-conquer recurrences that does not arise with linear recurrences. Specifically, dividing a problem of size n may create subproblems of non-integer size. For example, the Merge Sort recurrence contains the term $T(n/2)$. So what if n is 15? How long does it take to sort sevenand-a-half items? Previously, we dodged this issue by analyzing Merge Sort only when the size of the input was a power of 2. But then we don't know what happens for an input of size, say, 100.

Of course, a practical implementation of Merge Sort would split the input *approximately* in half, sort the halves recursively, and merge the results. For example, a list of 15 numbers would be split into lists of 7 and 8. More generally, a list of n numbers would be split into approximate halves of size $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$. So the maximum number of comparisons is actually given by this recurrence:

$$\begin{aligned} T(1) &= 0 \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n - 1 \qquad (\text{for } n \geq 2). \end{aligned} \tag{21.4.7}$$

This may be rigorously correct, but the ceiling and floor operations make the recurrence hard to solve exactly.

Fortunately, *the asymptotic solution to a divide and conquer recurrence is unaffected by floors and ceilings*. More precisely, the solution is not changed by replacing a term $T(b_i n)$ with either $T(\lceil b_i n \rceil)$ or $T(\lfloor b_i n \rfloor)$. So leaving floors and ceilings out of divide-and-conquer recurrences makes sense in many contexts; those are complications that make no difference.

The Akra-Bazzi Theorem

The Akra-Bazzi formula together with our assertions about boundary conditions and integrality all follow from the Akra-Bazzi Theorem, which is stated below.

Theorem 21.4.1

(Akra-Bazzi). *Suppose that the function $T : \mathbb{R} \to \mathbb{R}$ is nonnegative and bounded for $0 \leq x \leq x_0$ and satisfies the recurrence*

$$T(x) \sum_{i=1}^{k} a_i T(b_i x + h_i(x)) + g(x) \quad \text{for } x > x_0, \tag{21.4.8}$$

where:

1. *x_0 is large enough so that T is well-defined,*
2. *$a_1, \ldots, a_k$ are positive constants,*
3. *$b_1, \ldots, b_k$ are constants between 0 and 1,*
4. *$g(x)$ is a nonnegative function such that $|g'(x)|$ is bounded by a polynomial,*
5. *$|h_i(x)| = O(x/\log^2 x)$.*

Then

$$T(x) = \Theta\left(x^p \left(1 + \int_1^x \frac{g(u)}{u^{p+1}} du\right)\right)$$

where p satisfies

$$\sum_{i=1}^{k} a_i b_i^p = 1.$$

The Akra-Bazzi theorem can be proved using a complicated induction argument, though we won't do that here. But let's at least go over the statement of the theorem.

All the recurrences we've considered were defined over the integers, and that is the common case. But the Akra-Bazzi theorem applies more generally to functions defined over the real numbers.

The Akra-Bazzi formula is lifted directed from the theorem statement, except that the recurrence in the theorem includes extra functions, h_i. These functions extend the theorem to address floors, ceilings, and other small adjustments to the sizes of subproblems. The trick is illustrated by this combination of parameters

$$\begin{array}{lll} a_1 = 1 & b_1 = 1/2 & h_1(x) = \lceil \frac{x}{2} \rceil - \frac{x}{2} \\ a_2 = 1 & b_2 = 1/2 & h_2(x) = \lfloor \frac{x}{2} \rfloor - \frac{x}{2} \\ & g(x) = x - 1 & \end{array}$$

which corresponds the recurrence

$$\begin{aligned} T(x) &= 1 \cdot T\left(\frac{x}{2} + \left(\lceil \frac{x}{2} \rceil - \frac{x}{2}\right)\right) + \cdot T\left(\frac{x}{2} + \left(\lfloor \frac{x}{2} \rfloor - \frac{x}{2}\right)\right) + x - 1 \\ &= T\left(\lceil \frac{x}{2} \rceil\right) + T\left(\lfloor \frac{x}{2} \rfloor\right) + x - 1. \end{aligned} \tag{21.4.9}$$

This is the rigorously correct Merge Sort recurrence valid for all input sizes, complete with floor and ceiling operators. In this case, the functions $h_1(x)$ and $h_2(x)$ are both at most 1, which is easily $O(x/\log^2 x)$ as required by the theorem statement. These functions h_i do not affect—or even appear in—the asymptotic solution to the recurrence. This justifies our earlier claim that applying floor and ceiling operators to the size of a subproblem does not alter the asymptotic solution to a divide-and-conquer recurrence.

The Master Theorem

There is a special case of the Akra-Bazzi formula known as the Master Theorem that handles some of the recurrences that commonly arise in computer science. It is called the *Master* Theorem because it was proved long before Akra and Bazzi arrived on the scene and, for many years, it was the final word on solving divideand-conquer recurrences. We include the Master Theorem here because it is still widely referenced in algorithms courses and you can use it without having to know anything about integration.

Theorem 21.4.1

(Master Theorem). *Let T be a recurrence of the form*

$$T(n) = aT\left(\frac{n}{b}\right) + g(n).$$

Case 1: *If $g(n) = O\left(n^{\log_b(a)-\epsilon}\right)$ for some constant $\epsilon > 0$, then*

$$T(n) = \Theta\left(n^{\log_b(a)}\right).$$

Case 2: *If $g(n) = O\left(n^{\log_b(a)} \log^k(n)\right)$ for some constant $k > 0$, then*

$$T(n) = \Theta\left(n^{\log_b(a)} \log^{k+1}(n)\right).$$

Case 3: *If $g(n) = \Omega\left(n^{\log_b(a)+\epsilon}\right)$ for some constant $\epsilon > 0$ and $ag(n/b) < cg(n)$ for some constant $c < 1$ and sufficiently large n, then*

$$T(n) = \Theta(g(n)).$$

The Master Theorem can be proved by induction on n or, more easily, as a corollary of Theorem 21.4.1. We will not include the details here.

21.4: Divide-and-Conquer Recurrences is shared under a not declared license and was authored, remixed, and/or curated by LibreTexts.

21.5: A Feel for Recurrences

We've guessed and verified, plugged and chugged, found roots, computed integrals, and solved linear systems and exponential equations. Now let's step back and look for some rules of thumb. What kinds of recurrences have what sorts of solutions?

Here are some recurrences we solved earlier:

	Recurrence	Solution
Towers of Hanoi	$T_n = 2T_{n-1} + 1$	$T_n \sim 2^n$
Merge Sort	$T_n = 2T_{n/2} + n - 1$	$T_n \sim n \log n$
Hanoi variation	$T_n = 2T_{n-1} + n$	$T_n \sim 2 \cdot 2^n$
Fibonacci	$T_n = T_{n-1} + T_{n-2}$	$T_n \sim (1.618\ldots)^{n+1}/\sqrt{5}$

Notice that the recurrence equations for Towers of Hanoi and Merge Sort are somewhat similar, but the solutions are radically different. Merge Sorting $n = 64$ items takes a few hundred comparisons, while moving $n = 64$ disks takes more than 10^{19} steps!

Each recurrence has one strength and one weakness. In the Towers of Hanoi, we broke a problem of size n into two subproblem of size $n-1$ (which is large), but needed only 1 additional step (which is small). In Merge Sort, we divided the problem of size n into two subproblems of size $n/2$ (which is small), but needed $n-1$ additional steps (which is large). Yet, Merge Sort is faster by a mile!

This suggests that *generating smaller subproblems is far more important to algorithmic speed than reducing the additional steps per recursive call*. For example, shifting to the variation of Towers of Hanoi increased the last term from $+1$ to $+n$, but the solution only doubled. And one of the two subproblems in the Fibonacci recurrence is just *slightly* smaller than in Towers of Hanoi (size $n-2$ instead of $n-1$). Yet the solution is exponentially smaller! More generally, linear recurrences (which have big subproblems) typically have exponential solutions, while divideand-conquer recurrences (which have small subproblems) usually have solutions bounded above by a polynomial.

All the examples listed above break a problem of size n into two smaller problems. How does the number of subproblems affect the solution? For example, suppose we increased the number of subproblems in Towers of Hanoi from 2 to 3, giving this recurrence:

$$T_n = 3T_{n-1} + 1$$

This increases the root of the characteristic equation from 2 to 3, which raises the solution exponentially, from $\Theta(2^n)$ to $\Theta(3^n)$.

Divide-and-conquer recurrences are also sensitive to the number of subproblems. For example, for this generalization of the Merge Sort recurrence:

$$\begin{aligned} T_1 &= 0 \\ T_n &= aT_{n/2} + n - 1. \end{aligned} \tag{21.5.1}$$

the Akra-Bazzi formula gives:

$$T_n = \begin{cases} \Theta(n) & \text{for } a < 2 \\ \Theta(n \log n) & \text{for } a = 2 \\ \Theta(n^{\log a}) & \text{for } a > 2. \end{cases}$$

So the solution takes on three completely different forms as a goes from 1.99 to 2.01!

How do boundary conditions affect the solution to a recurrence? We've seen that they are almost irrelevant for divide-and-conquer recurrences. For linear recurrences, the solution is usually dominated by an exponential whose base is determined by the number and size of subproblems. Boundary conditions matter greatly only when they give the dominant term a zero coefficient, which changes the asymptotic solution.

So now we have a rule of thumb! The performance of a recursive procedure is usually dictated by the size and number of subproblems, rather than the amount of work per recursive call or time spent at the base of the recursion. In particular, if subproblems are smaller than the original by an additive factor, the solution is most often exponential. But if the subproblems are only a fraction the size of the original, then the solution is typically bounded by a polynomial.

21.5: A Feel for Recurrences is shared under a not declared license and was authored, remixed, and/or curated by LibreTexts.

Index

A

adjacency matrices
- 9.3: Adjacency Matrices

B

Bayes' rule
- 17.4: Why Tree Diagrams Work

binary relations
- 4.4: Binary Relations

C

cardinality
- 4.5: Finite Cardinality

conditional probability
- 17: Conditional Probability

D

divisibility
- 8.1: Divisibility

E

Enigma
- 8.8: Turing's Code (Version 2.0)

equilvalence
- 3.3: Equivalence and Validity

Euler's theorem
- 8.10: Euler's Theorem

F

functions
- 4.3: Functions

G

greatest common divisor
- 8.2: The Greatest Common Divisor

I

induction
- 5.1: Ordinary Induction

M

Master theorem
- 21.4: Divide-and-Conquer Recurrences

merge sort
- 21.2: Merge Sort

modular arithmetic
- 8.6: Modular Arithmetic

P

Pascal's triangle identity
- 14.10: Combinatorial Proofs

primality testing
- 8.5: Alan Turing

R

recursive definitions
- 6.1: Recursive Definitions and Structural Induction

remainder
- 8.7: Remainder Arithmetic

remainder arithmetic
- 8.7: Remainder Arithmetic

RSA public key encryption
- 8.11: RSA Public Key Encryption

S

set
- 4.1: Sets

structural induction
- 6.1: Recursive Definitions and Structural Induction

T

test for satisfiability
- 8.12: What has SAT got to do with it?

towers of Hanoi
- 15.4: Solving Linear Recurrences
- 21.1: The Towers of Hanoi

tree diagram
- 17.4: Why Tree Diagrams Work

truth tables
- 3.2: Propositional Logic in Computer Programs

Turing, Alan
- 8.5: Alan Turing

V

validity
- 3.3: Equivalence and Validity

variance
- 19.3: Properties of Variance

W

walk relation
- 9.4: Walk Relations

well ordering proofs
- 2.2: Template for Well Ordering Proofs

Glossary

Sample Word 1 | Sample Definition 1

Detailed Licensing

Overview

Title: Mathematics for Computer Science (Lehman, Leighton, and Meyer)

Webpages: 177

Applicable Restrictions: Noncommercial

All licenses found:

- CC BY-NC-SA 4.0: 94.4% (167 pages)
- Undeclared: 5.6% (10 pages)

By Page

- Mathematics for Computer Science (Lehman, Leighton, and Meyer) - *CC BY-NC-SA 4.0*
  - Front Matter - *CC BY-NC-SA 4.0*
    - TitlePage - *CC BY-NC-SA 4.0*
    - InfoPage - *CC BY-NC-SA 4.0*
    - Table of Contents - *Undeclared*
    - Licensing - *Undeclared*
  - 1: Proofs - *CC BY-NC-SA 4.0*
    - 1: Intro to Proofs - *CC BY-NC-SA 4.0*
      - 1.1: Propositions - *CC BY-NC-SA 4.0*
      - 1.2: Predicates - *CC BY-NC-SA 4.0*
      - 1.3: The Axiomatic Method - *CC BY-NC-SA 4.0*
      - 1.4: Our Axioms - *CC BY-NC-SA 4.0*
      - 1.5: Proving an Implication - *CC BY-NC-SA 4.0*
      - 1.6: Proving an “If and Only If” - *CC BY-NC-SA 4.0*
      - 1.7: Proof by Cases - *CC BY-NC-SA 4.0*
      - 1.8: Proof by Contradiction - *CC BY-NC-SA 4.0*
      - 1.9: Good Proofs in Practice - *CC BY-NC-SA 4.0*
    - 2: Well Ordering Principle - *CC BY-NC-SA 4.0*
      - 2.1: Well Ordering Proofs - *CC BY-NC-SA 4.0*
      - 2.2: Template for Well Ordering Proofs - *CC BY-NC-SA 4.0*
      - 2.3: Factoring into Primes - *CC BY-NC-SA 4.0*
      - 2.4: Well Ordered Sets - *Undeclared*
    - 3: Logical Formulas - *CC BY-NC-SA 4.0*
      - 3.1: Propositions from Propositions - *CC BY-NC-SA 4.0*
      - 3.2: Propositional Logic in Computer Programs - *CC BY-NC-SA 4.0*
      - 3.3: Equivalence and Validity - *CC BY-NC-SA 4.0*
      - 3.4: The Algebra of Propositions - *CC BY-NC-SA 4.0*
      - 3.5: The SAT Problem - *CC BY-NC-SA 4.0*
      - 3.6: Predicate Formulas - *CC BY-NC-SA 4.0*
    - 4: Mathematical Data Types - *CC BY-NC-SA 4.0*
      - 4.1: Sets - *CC BY-NC-SA 4.0*
      - 4.2: Sequences - *CC BY-NC-SA 4.0*
      - 4.3: Functions - *CC BY-NC-SA 4.0*
      - 4.4: Binary Relations - *CC BY-NC-SA 4.0*
      - 4.5: Finite Cardinality - *CC BY-NC-SA 4.0*
    - 5: Induction - *CC BY-NC-SA 4.0*
      - 5.1: Ordinary Induction - *CC BY-NC-SA 4.0*
      - 5.2: Strong Induction - *CC BY-NC-SA 4.0*
      - 5.3: Strong Induction vs. Induction vs. Well Ordering - *CC BY-NC-SA 4.0*
      - 5.4: State Machines - *CC BY-NC-SA 4.0*
    - 6: Recursive Data Types - *CC BY-NC-SA 4.0*
      - 6.1: Recursive Definitions and Structural Induction - *CC BY-NC-SA 4.0*
      - 6.2: Strings of Matched Brackets - *CC BY-NC-SA 4.0*
      - 6.3: Recursive Functions on Nonnegative Integers - *CC BY-NC-SA 4.0*
      - 6.4: Arithmetic Expressions - *CC BY-NC-SA 4.0*
      - 6.5: Induction in Computer Science - *CC BY-NC-SA 4.0*
    - 7: Infinite Sets - *CC BY-NC-SA 4.0*
      - 7.1: Infinite Cardinality - *CC BY-NC-SA 4.0*
      - 7.2: The Halting Problem - *CC BY-NC-SA 4.0*
      - 7.3: The Logic of Sets - *CC BY-NC-SA 4.0*
      - 7.4: Does All This Really Work? - *CC BY-NC-SA 4.0*
  - 2: Structures - *CC BY-NC-SA 4.0*
    - 8: Number Theory - *CC BY-NC-SA 4.0*
      - 8.1: Divisibility - *CC BY-NC-SA 4.0*
      - 8.2: The Greatest Common Divisor - *CC BY-NC-SA 4.0*

- 8.3: Prime Mysteries - *CC BY-NC-SA 4.0*
- 8.4: The Fundamental Theorem of Arithmetic - *CC BY-NC-SA 4.0*
- 8.5: Alan Turing - *CC BY-NC-SA 4.0*
- 8.6: Modular Arithmetic - *CC BY-NC-SA 4.0*
- 8.7: Remainder Arithmetic - *CC BY-NC-SA 4.0*
- 8.8: Turing's Code (Version 2.0) - *CC BY-NC-SA 4.0*
- 8.9: Multiplicative Inverses and Cancelling - *CC BY-NC-SA 4.0*
- 8.10: Euler's Theorem - *CC BY-NC-SA 4.0*
- 8.11: RSA Public Key Encryption - *CC BY-NC-SA 4.0*
- 8.12: What has SAT got to do with it? - *CC BY-NC-SA 4.0*

- 9: Directed graphs and Partial Orders - *CC BY-NC-SA 4.0*
  - 9.1: Vertex Degrees - *CC BY-NC-SA 4.0*
  - 9.2: Walks and Paths - *CC BY-NC-SA 4.0*
  - 9.3: Adjacency Matrices - *CC BY-NC-SA 4.0*
  - 9.4: Walk Relations - *CC BY-NC-SA 4.0*
  - 9.5: Directed Acyclic Graphs and Scheduling - *CC BY-NC-SA 4.0*
  - 9.6: Partial Orders - *CC BY-NC-SA 4.0*
  - 9.7: Representing Partial Orders by Set Containment - *CC BY-NC-SA 4.0*
  - 9.8: Linear Orders - *CC BY-NC-SA 4.0*
  - 9.9: Product Orders - *CC BY-NC-SA 4.0*
  - 9.10: Equivalence Relations - *CC BY-NC-SA 4.0*
  - 9.11: Summary of Relational Properties - *CC BY-NC-SA 4.0*
- 10: Communication Networks - *CC BY-NC-SA 4.0*
  - 10.1: Complete Binary Tree - *CC BY-NC-SA 4.0*
  - 10.2: Routing Problems - *CC BY-NC-SA 4.0*
  - 10.3: Network Diameter - *CC BY-NC-SA 4.0*
  - 10.4: Switch Count - *CC BY-NC-SA 4.0*
  - 10.5: Network Latency - *CC BY-NC-SA 4.0*
  - 10.6: Congestion - *CC BY-NC-SA 4.0*
  - 10.7: 2-D Array - *CC BY-NC-SA 4.0*
  - 10.8: Butterfly - *CC BY-NC-SA 4.0*
  - 10.9: Beneš Network - *CC BY-NC-SA 4.0*
- 11: Simple Graphs - *CC BY-NC-SA 4.0*
  - 11.1: Vertex Adjacency and Degrees - *CC BY-NC-SA 4.0*
  - 11.2: Sexual Demographics in America - *CC BY-NC-SA 4.0*
  - 11.3: Some Common Graphs - *CC BY-NC-SA 4.0*
  - 11.4: Isomorphism - *CC BY-NC-SA 4.0*
  - 11.5: Bipartite Graphs and Matchings - *CC BY-NC-SA 4.0*
  - 11.6: The Stable Marriage Problem - *CC BY-NC-SA 4.0*
  - 11.7: Coloring - *CC BY-NC-SA 4.0*
  - 11.8: Simple Walks - *CC BY-NC-SA 4.0*
  - 11.9: Connectivity - *CC BY-NC-SA 4.0*
  - 11.10: Forests and Trees - *CC BY-NC-SA 4.0*
- 12: Planar Graphs - *CC BY-NC-SA 4.0*
  - 12.1: Drawing Graphs in the Plane - *CC BY-NC-SA 4.0*
  - 12.2: Definitions of Planar Graphs - *CC BY-NC-SA 4.0*
  - 12.3: Euler's Formula - *CC BY-NC-SA 4.0*
  - 12.4: Bounding the Number of Edges in a Planar Graph - *CC BY-NC-SA 4.0*
  - 12.5: Returning to K5 and K3,3 - *CC BY-NC-SA 4.0*
  - 12.6: Coloring Planar Graphs - *CC BY-NC-SA 4.0*
  - 12.7: Classifying Polyhedra - *CC BY-NC-SA 4.0*
  - 12.8: Another Characterization for Planar Graphs - *CC BY-NC-SA 4.0*

- 3: Counting - *CC BY-NC-SA 4.0*
  - 13: Sums and Asymptotics - *CC BY-NC-SA 4.0*
    - 13.1: The Value of an Annuity - *CC BY-NC-SA 4.0*
    - 13.2: Sums of Powers - *CC BY-NC-SA 4.0*
    - 13.3: Approximating Sums - *CC BY-NC-SA 4.0*
    - 13.4: Hanging Out Over the Edge - *CC BY-NC-SA 4.0*
    - 13.5: Products - *CC BY-NC-SA 4.0*
    - 13.6: Double Trouble - *CC BY-NC-SA 4.0*
    - 13.7: Asymptotic Notation - *Undeclared*
  - 14: Cardinality Rules - *CC BY-NC-SA 4.0*
    - 14.1: Counting One Thing by Counting Another - *CC BY-NC-SA 4.0*
    - 14.2: Counting Sequences - *CC BY-NC-SA 4.0*
    - 14.3: The Generalized Product Rule - *CC BY-NC-SA 4.0*
    - 14.4: The Division Rule - *CC BY-NC-SA 4.0*
    - 14.5: Counting Subsets - *CC BY-NC-SA 4.0*
    - 14.6: Sequences with Repetitions - *CC BY-NC-SA 4.0*
    - 14.7: Counting Practice - Poker Hands - *CC BY-NC-SA 4.0*
    - 14.8: The Pigeonhole Principle - *CC BY-NC-SA 4.0*
    - 14.9: Inclusion-Exclusion - *CC BY-NC-SA 4.0*
    - 14.10: Combinatorial Proofs - *CC BY-NC-SA 4.0*
  - 15: Generating Functions - *CC BY-NC-SA 4.0*

- 15.1: Infinite Series - *CC BY-NC-SA 4.0*
- 15.2: Counting with Generating Functions - *CC BY-NC-SA 4.0*
- 15.3: Partial Fractions - *CC BY-NC-SA 4.0*
- 15.4: Solving Linear Recurrences - *CC BY-NC-SA 4.0*
- 15.5: Formal Power Series - *CC BY-NC-SA 4.0*

4: Probability - *CC BY-NC-SA 4.0*

- 16: Events and Probability Spaces - *CC BY-NC-SA 4.0*
  - 16.1: Let's Make a Deal - *CC BY-NC-SA 4.0*
  - 16.2: The Four Step Method - *CC BY-NC-SA 4.0*
  - 16.3: Strange Dice - *CC BY-NC-SA 4.0*
  - 16.4: The Birthday Principle - *CC BY-NC-SA 4.0*
  - 16.5: Set Theory and Probability - *CC BY-NC-SA 4.0*
- 17: Conditional Probability - *CC BY-NC-SA 4.0*
  - 17.1: Monty Hall Confusion - *CC BY-NC-SA 4.0*
  - 17.2: Definition and Notation - *CC BY-NC-SA 4.0*
  - 17.3: The Four-Step Method for Conditional Probability - *CC BY-NC-SA 4.0*
  - 17.4: Why Tree Diagrams Work - *CC BY-NC-SA 4.0*
  - 17.5: The Law of Total Probability - *CC BY-NC-SA 4.0*
  - 17.6: Simpson's Paradox - *CC BY-NC-SA 4.0*
  - 17.7: Independence - *CC BY-NC-SA 4.0*
  - 17.8: Mutual Independence - *CC BY-NC-SA 4.0*
- 18: Random Variables - *CC BY-NC-SA 4.0*
  - 18.1: Random Variable Examples - *CC BY-NC-SA 4.0*
  - 18.2: Independence - *CC BY-NC-SA 4.0*
  - 18.3: Distribution Functions - *CC BY-NC-SA 4.0*
  - 18.4: Great Expectations - *CC BY-NC-SA 4.0*
  - 18.5: Linearity of Expectation - *CC BY-NC-SA 4.0*
- 19: Deviation from the Mean - *CC BY-NC-SA 4.0*
  - 19.1: Markov's Theorem - *CC BY-NC-SA 4.0*
  - 19.2: Chebyshev's Theorem - *CC BY-NC-SA 4.0*
  - 19.3: Properties of Variance - *CC BY-NC-SA 4.0*
  - 19.4: Estimation by Random Sampling - *CC BY-NC-SA 4.0*
  - 19.5: Confidence versus Probability - *CC BY-NC-SA 4.0*
  - 19.6: Sums of Random Variables - *CC BY-NC-SA 4.0*
  - 19.7: Really Great Expectations - *CC BY-NC-SA 4.0*
- 20: Random Walks - *CC BY-NC-SA 4.0*
  - 20.1: Gambler's Ruin - *CC BY-NC-SA 4.0*
  - 20.2: Random Walks on Graphs - *CC BY-NC-SA 4.0*

5: Recurrences - *CC BY-NC-SA 4.0*

- 21: Recurrences - *CC BY-NC-SA 4.0*
  - 21.1: The Towers of Hanoi - *Undeclared*
  - 21.2: Merge Sort - *Undeclared*
  - 21.3: Linear Recurrences - *Undeclared*
  - 21.4: Divide-and-Conquer Recurrences - *Undeclared*
  - 21.5: A Feel for Recurrences - *Undeclared*

Back Matter - *CC BY-NC-SA 4.0*

- Index - *CC BY-NC-SA 4.0*
- Glossary - *CC BY-NC-SA 4.0*
- Detailed Licensing - *Undeclared*