

DISCRETE MATHEMATICS FOR COMPUTER SCIENCE



Jin He
Fresno City College

Discrete Mathematics for Computer Science

Discrete Math for Computer Science

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 borne 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 were 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 adaptations contact info@LibreTexts.org or visit our main website at <https://LibreTexts.org>.

The LibreTexts name, logo, and book covers are trademarked by LibreTexts, Inc. (a 501(c)(3) not-for-profit organization) and protected against unauthorized use.

This text was compiled on 10/13/2026

TABLE OF CONTENTS

Discrete Mathematics for Computer Science

Licensing

1: Introduction to Propositional Logic

- 1.1: Propositions
- 1.2: Predicates
- 1.3: Proving an Implication
- 1.4: Proving an “If and Only If”
- 1.5: Proof by Cases
- 1.6: Proof by Contradiction
- 1.7: Propositions from Propositions
- 1.8: Propositional Logic in Computer Programs
- 1.9: Equivalence and Validity
- 1.10: The Algebra of Propositions
- 1.11: Predicate Formulas

2: Sets and Set Operations

- 2.1: Sets and Set Notations
- 2.2: Set Operations
- 2.3: Set Difference and Set Complement

3: Functions and Binary Relations

- 3.1: Functions
- 3.2: Binary Relations
- 3.3: Reverse Relations and Images
- 3.4: Cardinality and Power Set

4: Mathematical Induction

- 4.1: Introduction to Mathematical Induction
- 4.2: Proof by Using Mathematical Induction
- 4.3: Loop Invariant

5: Algorithms

- 5.1: Introduction to Algorithm
- 5.2: More Examples of Algorithms

6: Cardinality Rules

- 6.1: Counting One Thing by Counting Another
- 6.2: Counting Sequences
- 6.3: The Generalized Product Rule
- 6.4: The Division Rule
- 6.5: Counting Subsets

- 6.6: Sequences with Repetitions
- 6.7: Counting Practice - Poker Hands
- 6.8: The Pigeonhole Principle
- 6.9: Inclusion-Exclusion
- 6.10: Combinatorial Proofs

7: Graph Theory

- 7.1: Introduction
- 7.2: Path and Cycles

8: Trees

- 8.1: Introduction and Huffman Codes
- 8.2: Terms and Characterizations of Trees
- 8.3: Binary Trees and Binary Search Trees

Index

Glossary

Detailed Licensing

Detailed Licensing

Licensing

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

This page titled [Licensing](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

1: INTRODUCTION TO PROPOSITIONAL LOGIC

- 1.1: Propositions
- 1.2: Predicates
- 1.3: Proving an Implication
- 1.4: Proving an “If and Only If”
- 1.5: Proof by Cases
- 1.6: Proof by Contradiction
- 1.7: Propositions from Propositions
- 1.8: Propositional Logic in Computer Programs
- 1.9: Equivalence and Validity
- 1.10: The Algebra of Propositions
- 1.11: Predicate Formulas

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

$$p(n) ::= n^2 + n + 41.^1 \quad (1.1.1)$$

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

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

are each prime. Hmm, 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

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

Here the symbol $\forall$ is read "for all." The symbol $\mathbb{N}$ stands for the set of *nonnegative integers*: 0, 1, 2, 3, ... (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,

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

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

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

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

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 adjacent² 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.

¹The 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.

²Two 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.

³The 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.

⁴In 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 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [1.1: Propositions](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

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:

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

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!

- [1.2: Predicates](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

1.3: 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 1.3.1

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 $\square$ 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 1.3.2

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 [1.3: Proving an Implication](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [1.5: Proving an Implication](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

1.4: 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 $x_1, x_2, \dots, x_n$ is defined to be:

$$\sqrt{\frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_n - \mu)^2}{n}} \quad (1.4.1)$$

where μ is the average or mean of the values:

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

Theorem 1.4.1

The standard deviation of a sequence of values $x_1, x_2, \dots, x_n$ 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 (1.4.1) is zero:

$$\sqrt{\frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_n - \mu)^2}{n}} = 0. \quad (1.4.2)$$

Now since zero is the only number whose square root is zero, equation (1.4.2) holds iff

$$(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_n - \mu)^2 = 0. \quad (1.4.3)$$

Squares of real numbers are always nonnegative, so every term on the left hand side of equation (1.4.3) is nonnegative. This means that (1.4.3) holds iff

Every term on the left hand side of (1.4.3) is zero. (1.4.4)

But a term $(x_i - \mu)^2$ is zero iff $x_i = \mu$, so (1.4.4) is true iff

Every x_i equals the mean. ■

This page titled 1.4: Proving an “If and Only If” is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by Jin He.

- 1.6: Proving an “If and Only If” by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

1.5: 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 available at the end of this section.

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

⁶Part 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.

Answers

Theorem

The proof is by case analysis⁵. 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. ■

This page titled [1.5: Proof by Cases](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [1.7: Proof by Cases](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

1.6: 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 $0.1111 \dots = 1/9$ are rational numbers.

Theorem 1.6.1

$\sqrt{2}$ is irrational.

Proof available at the end of this section.

Answers

Theorem 1.6.1

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 [1.6: Proof by Contradiction](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [1.8: Proof by Contradiction](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

1.7: 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	$P \text{ XOR } Q$
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	$P \text{ IMPLIES } Q$	
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 $P \text{ IMPLIES } Q$ 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

$$[C_1 \text{ IMPLIES } A_1] \text{ AND } [C_2 \text{ IMPLIES } A_2] \text{ AND } \dots \text{ AND } [C_{12} \text{ IMPLIES } A_{12}] \quad (1.7.1)$$

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 (1.7.1) to come out true. Now the implications $C_2 \text{ IMPLIES } A_2$ and $C_5 \text{ IMPLIES } A_5$ are both true because both their hypotheses and their conclusions are true. But in order for (1.7.1) 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	$P \text{ IFF } Q$
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.

- [3.1: Propositions from Propositions](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

1.8: 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

$$A \text{ OR } (\text{NOT}(A) \text{ AND } B) \quad (1.8.1)$$

Truth Table Calculation

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

$$A \text{ OR } B. \quad (1.8.2)$$

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

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

These values are enough to fill in two more columns:

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

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

A	B	$A \text{ OR } (\text{NOT}(A) \text{ AND } B)$	$A \text{ OR } B$
T	T	T	T
T	F	T	T
F	T	T	T
F	F	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:

$$\begin{aligned} &\text{if } (x > 0 || y > 100) \\ &\quad \vdots \\ &\quad (further\ instructions) \end{aligned} \tag{1.8.3}$$

The equivalence of (1.8.1) and (1.8.2) 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 (1.8.1) and (1.8.2) 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, (1.8.2) 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 (1.8.1) and (1.8.2) 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.

1.9: 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 “ $\text{NOT}(Q)$ IMPLIES $\text{NOT}(P)$.” Once more, we can compare these two statements in a truth table:

P	Q	$(P \text{ IMPLIES } Q)$	$(\text{NOT}(Q))$	IMPLIES	$\text{NOT}(P)$
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 “ $\text{NOT}(Q)$ IMPLIES $\text{NOT}(P)$ ” 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	$P \text{ IMPLIES } Q$	$Q \text{ IMPLIES } P$
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	$(P \text{ IMPLIES } Q)$	AND	$(Q \text{ IMPLIES } P)$	$(P \text{ IFF } Q)$
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

$$(P \text{ IMPLIES } Q) \text{ AND } (Q \text{ IMPLIES } P)$$

is the same as the sixth column giving the truth values of $P \text{ IFF } Q$, 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 $(F \text{ IFF } G)$ 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 $\text{NOT}(P)$ is *not* valid.

- [3.3: Equivalence and Validity](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

1.10: 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,

$$(A \text{ AND } B) \text{ OR } (A \text{ AND } C). \quad (1.10.1)$$

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

$$A \text{ AND } (B \text{ OR } C) \quad (1.10.2)$$

has truth table:

A	B	C	$A \text{ AND } (B \text{ OR } C)$
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 (1.10.2) is true in the first row when A, B , and C are all true, that is, where $A \text{ AND } B \text{ AND } C$ is true. It is also true in the second row where $A \text{ AND } B \text{ AND } \overline{C}$ is true, and in the third row when $A \text{ AND } \overline{B} \text{ AND } C$ is true, and that's all. So (1.10.2) is true exactly when

$$(A \text{ AND } B \text{ AND } C) \text{ OR } (A \text{ AND } B \text{ AND } \overline{C}) \text{ OR } (A \text{ AND } \overline{B} \text{ AND } C) \quad (1.10.3)$$

is true.

Theorem 1.10.1

[Distributive Law of AND over OR]

$A \text{ AND } (B \text{ OR } C)$ is equivalent to $(A \text{ AND } B) \text{ OR } (A \text{ AND } C)$.

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 1.10.2

[Distributive Law of OR over AND]

$A \text{ OR } (B \text{ AND } C)$ is equivalent to $(A \text{ OR } B) \text{ AND } (A \text{ OR } C)$.

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

The expression (1.10.3) 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 (1.10.3) further simplifies to the equivalent disjunctive form (1.10.1) 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 (1.10.2) 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 $(\bar{A} \text{ OR } B \text{ OR } C)$ is F! Likewise, the (1.10.2) is false in the fifth row which is exactly where $(A \text{ OR } \bar{B} \text{ OR } \bar{C})$ is F. This means that (1.10.2) 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 (1.10.2) is false, we get a *conjunctive normal form (CNF)* that is equivalent to (1.10.2), namely,

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

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

Theorem 1.10.3

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, $A \text{ IMPLIES } B$ is equivalent to $\text{NOT } (A) \text{ OR } B$. 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 " $\leftrightarrow$ " 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:

$$\begin{aligned} A \text{ AND } B &\leftrightarrow B \text{ AND } A && \text{(commutativity of AND)} && (1.10.4) \\ (A \text{ AND } B) \text{ AND } C &\leftrightarrow A \text{ AND } (B \text{ AND } C) && \text{(associativity of AND)} && (1.10.5) \\ \mathbf{T} \text{ AND } A &\leftrightarrow A && \text{(identity of AND)} && \\ \mathbf{F} \text{ AND } A &\leftrightarrow \mathbf{F} && \text{(zero for AND)} && \end{aligned}$$

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

$$\begin{aligned} A \text{ AND } A &\leftrightarrow A && \text{(idempotence for AND)} \\ A \text{ AND } \bar{A} &\leftrightarrow \mathbf{F} && \text{(contradiction for AND)} && (1.10.6) \end{aligned}$$

$$\text{NOT } (\bar{A}) \leftrightarrow A \quad \text{(double negation)} \quad (1.10.7)$$

It is associativity (1.10.5) that justifies writing $A \text{ AND } B \text{ AND } C$ without specifying whether it is parenthesized as $A \text{ AND } (B \text{ AND } C)$ or $(A \text{ AND } B) \text{ AND } C$. 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 (1.10.6):

$$A \text{ OR } \bar{A} \leftrightarrow \mathbf{T} \quad \text{(validity for OR)} \quad (1.10.8)$$

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

$$\text{NOT}(A \text{ AND } B) \leftrightarrow \bar{A} \text{ OR } \bar{B} \quad \text{(DeMorgan for AND)} \quad (1.10.9)$$

$$\text{NOT}(A \text{ OR } B) \leftrightarrow \bar{A} \text{ AND } \bar{B} \quad \text{(DeMorgan for OR)} \quad (1.10.10)$$

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 (1.10.2),

$$\text{NOT}((A \text{ AND } B) \text{ OR } (A \text{ AND } C)). \quad (1.10.11)$$

into disjunctive normal form.

We start by applying DeMorgan's Law for OR (1.10.10) to (1.10.11) in order to move the NOT deeper into the formula. This gives

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

Now applying Demorgan's Law for AND (1.10.9) to the two innermost AND-terms, gives

$$(\bar{A} \text{ OR } \bar{B}) \text{ AND } (\bar{A} \text{ OR } \bar{C}). \quad (1.10.12)$$

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 (1.10.12) into a disjunctive form. To start, we'll distribute $(\bar{A} \text{ OR } \bar{B})$ over AND to get

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

Using distributivity over both AND's we get

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

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

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

Associativity now allows dropping the parentheses around the terms being OR'd to yield the following disjunctive form for (1.10.11):

$$\bar{A} \text{ OR } (\bar{B} \text{ AND } \bar{A}) \text{ OR } (\bar{A} \text{ AND } \bar{C}) \text{ OR } (\bar{B} \text{ AND } \bar{C}). \quad (1.10.13)$$

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 $(\bar{B} \text{ AND } \bar{A})$. 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

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

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

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

Doing the same thing to the other AND-terms in (1.10.13) finally gives a disjunctive normal form for (1.10.2):

$$\begin{aligned} &(\bar{A} \text{ AND } B \text{ AND } C) \text{ OR } (\bar{A} \text{ AND } B \text{ AND } \bar{C}) \text{ OR} \\ &(\bar{A} \text{ AND } \bar{B} \text{ AND } C) \text{ OR } (\bar{A} \text{ AND } \bar{B} \text{ AND } \bar{C}) \text{ OR} \\ &(\bar{B} \text{ AND } \bar{A} \text{ AND } C) \text{ OR } (\bar{B} \text{ AND } \bar{A} \text{ AND } \bar{C}) \text{ OR} \\ &(\bar{A} \text{ AND } \bar{C} \text{ AND } B) \text{ OR } (\bar{A} \text{ AND } \bar{C} \text{ AND } \bar{B}) \text{ OR} \\ &(\bar{B} \text{ AND } \bar{C} \text{ AND } A) \text{ OR } (\bar{B} \text{ AND } \bar{C} \text{ AND } \bar{A}). \end{aligned} \quad (1.10.14)$$

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

$$\begin{aligned} & (A \text{ AND } \bar{B} \text{ AND } \bar{C}) \text{ OR} \\ & (\bar{A} \text{ AND } B \text{ AND } C) \text{ OR} \\ & (\bar{A} \text{ AND } B \text{ AND } \bar{C}) \text{ OR} \\ & (\bar{A} \text{ AND } \bar{B} \text{ AND } C) \text{ OR} \\ & (\bar{A} \text{ AND } \bar{B} \text{ AND } \bar{C}). \end{aligned} \tag{1.10.15}$$

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 1.10.4

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 1.10.5

(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.

1.11: Predicate Formulas

Quantifiers

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

$$x^2 \geq 0$$

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

$$\forall x^{\text{int}} \mathbb{R}. x^2 \geq 0$$

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

$$5x^2 - 7 = 0$$

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

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

is true, while

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

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

Sometimes True

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

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. (1.11.1)

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, (1.11.2)

or maybe

you can solve *at least one* problem we come up with. (1.11.3)

To be precise, let Probs be the set of problems we come up with, $\text{Solves}(x)$ 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:

$(\forall x^{\text{int}} \text{ Probs.Solves}(x)) \text{ IMPLIES } G$, for (1.11.1),

$(\exists x^{\text{int}} \text{ Probs.Solves}(x)) \text{ IMPLIES } G$, for (1.11.2),

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:

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

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:

$$\exists d^{\text{int}} D \forall a^{\text{int}} A. H(a, d)$$

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

$$\exists a^{\text{int}} A \forall d^{\text{int}} D. H(a, d)$$

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:

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

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 $\forall x^{\text{int}} D \exists y^{\text{int}} D. Q(x, y)$ 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

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

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 an instance of a general equivalence that holds between predicate formulas:

$$\text{NOT } (\forall x. P(x)) \text{ is equivalent to } \exists x. \text{NOT}(P(x)). \quad (1.11.4)$$

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

$$\text{NOT } (\exists x. P(x)) \text{ is equivalent to } \forall x. \text{NOT}(P(x)). \quad (1.11.5)$$

The general principle is that *moving a NOT across a quantifier changes the kind of quantifier*. Note that (1.11.5) follows from negating both sides of (1.11.4).

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 (1.11.4) that gives the rule for negating a universal quantifier means that the following formula is valid:

$$\text{NOT } (\forall x. P(x)) \text{ IFF } \exists x. \text{NOT}(P(x)). \quad (1.11.6)$$

Another useful example of a valid assertion is

$$\exists x \forall y. P(x, y) \text{ IMPLIES } \forall y \exists x. P(x, y). \quad (1.11.7)$$

Here's an explanation why this is valid: Let D be the domain for the variables and P_0 be some binary predicate² on D . We need to show that if

$$\exists x^{\text{int}} D. \forall y^{\text{int}} D. P_0(x, y) \quad (1.11.8)$$

holds under this interpretation, then so does

$$\forall y^{\text{int}} D \exists x^{\text{int}} D. P_0(x, y). \quad (1.11.9)$$

So suppose (1.11.8) is true. Then by definition of $\exists$, this means that some element $d_0^{\text{int}} D$ has the property that

$$\forall y^{\text{int}} D. P_0(d_0, y).$$

By definition of $\forall$, this means that

$$P_0(d_0, d)$$

is true for all $d^{\text{int}} D$. So given any $d^{\text{int}} D$, there is an element in D , namely, d_0 , such that $P_0(d_0, d)$ is true. But that's exactly what (1.11.9) means, so we've proved that (1.11.9) 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 (1.11.7), 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 (1.11.7) into English and then appeal to the meaning, in English, of "for all" and "there exists" as justification.

In contrast to (1.11.7), the formula

$$\forall y \exists x.P(x, y) \text{ IMPLIES } \forall x \exists y.P(x, y) \quad (1.11.10)$$

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.

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

This page titled 1.11: Predicate Formulas is shared under a CC BY-NC-SA 4.0 license and was authored, remixed, and/or curated by Jin He.

- 3.6: Predicate Formulas by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed CC BY-NC-SA 4.0.

2: SETS AND SET OPERATIONS

2.1: Sets and Set Notations

2.2: Set Operations

2.3: Set Difference and Set Complement

2.1: Sets and Set Notations

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:

$$\begin{aligned} A &= \{\text{Alex, Tippy, Shells, Shadow}\} && \text{dead pets} \\ B &= \{\text{red, blue, yellow}\} && \text{primary colors} \\ C &= \{\{a, b\}, \{a, c\}, \{b, c\}\} && \text{a set of sets} \end{aligned} \quad (2.1.1)$$

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

$$D ::= \{1, 2, 4, 8, 16, \dots\} \quad \text{the powers of 2} \quad (2.1.2)$$

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 $e^{\text{int}} S$ asserts that e is an element of set S . For example, $32^{\text{int}} D$ and $\text{blue}^{\text{int}} B$, but $\text{Tailspin} \notin A$ —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

$$\begin{aligned} X &::= \{1, 2, 3\}, \\ Y &::= \{2, 3, 4\}. \end{aligned} \quad (2.1.3)$$

Definition 2.1.1

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

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

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,

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

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,

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

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 $\bar{A}$ to be the set of all elements of D *not* in A . That is,

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

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

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

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

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

We can use complement to rephrase subset in terms of equality

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

Power Set

The set of all the subsets of a set, A , is called the *power set*, $\text{pow}(A)$, of A . So

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

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

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

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:

$$\begin{aligned} A &::= \{n^{\text{int}} \mathbb{N} \mid n \text{ is a prime and } n = 4k + 1 \text{ for some integer } k\} \\ B &::= \{x^{\text{int}} \mathbb{R} \mid x^3 - 3x + 1 > 0\} \\ C &::= \{a + bi^{\text{int}} \mathbb{C} \mid a^2 + 2b^2 \leq 1\} \end{aligned} \tag{2.1.4}$$

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

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

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

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

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 $a + bi$ 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 $x \in X$ if and only if $x \in Y$, for all elements, x .² So, set equalities can be formulated and proved as “iff” theorems. For example:

Theorem 2.1.2

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

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \quad (2.1.5)$$

Proof available at the end of this section.

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 $A \text{ AND } B$ 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:

$$\overline{A \cap B} = \overline{A} \cup \overline{B}. \quad (2.1.6)$$

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 “ $X \text{ AND } Y$ ” 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 “ $P \text{ OR } Q$.”

¹It'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.

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

Answers

Theorem 2.1.2

The equality (2.1.5) is equivalent to the assertion that

$$z^{\text{int}} A \cap (B \cup C) \text{ iff } z^{\text{int}} (A \cap B) \cup (A \cap C) \quad (2.1.7)$$

for all z . Now we'll prove (2.1.7) by a chain of iff's.

Now we have

$$\begin{aligned} & z^{\text{int}} A \cap (B \cup C) \\ & \text{iff } (z^{\text{int}} A) \text{ AND } (z^{\text{int}} B \cup C) && (\text{def of } \cap) \\ & \text{iff } (z^{\text{int}} A) \text{ AND } (z^{\text{int}} B \text{ OR } z^{\text{int}} C) && (\text{def of } \cup) \\ & \text{iff } (z^{\text{int}} A \text{ AND } z^{\text{int}} B) \text{ OR } (z^{\text{int}} A \text{ AND } z^{\text{int}} C) && (\text{AND distributivity Thm 3.4.1}) \\ & \text{iff } (z^{\text{int}} A \cap B) \text{ OR } (z^{\text{int}} A \cap C) && (\text{def of } \cap) \\ & \text{iff } z^{\text{int}} (A \cap B) \cup (A \cap C) && (\text{def of } \cup) \end{aligned} \quad (2.1.8)$$

■

This page titled [2.1: Sets and Set Notations](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [4.1: Sets](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

2.2: Set Operations

In algebra, the most common operations are addition, subtraction, multiplication, and division and the operands are numbers. In this class, we will discuss operations on sets and introduce the most common operations: union, intersection, difference, complement, product and power operations. We will also introduce the laws about those operations. Those laws include commutative law, distributive law and associative law.

Union and Intersection

Definition 1. Let A and B be sets. The **union** of A and B, denoted by $A \cup B$, is a set that consists of the elements either in A or in B or in both. With set builder notation,

$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}$$

We use following graph to provide a visual presentation of the union of A and B:

preview

The areas in either the circles A or in the circle B or in both circle A and circle B represent the union of A and B, which is $A \cup B$. The graph we used here is called **Venn Diagram**.

Example

Let $A = \{4, 3, -1, 5, 6, 2, 7, -3\}$ and $B = \{10, 3, -2, 5, 4, 8, 0, -3\}$. Find $A \cup B$

$A \cup B$ is a set that includes all the elements in the set A or and all the elements in the set B. Thus, $A \cup B = \{3, 5, 4, -3, -1, 6, 2, 7, 10, 8, 0, -2\}$.

Please note: We don't list those numbers in both sets twice.

The union of sets are easy to understand, but there are still a few things that are worth to mention:

- $A \cup A = A$. The union of a set with itself is still itself (don't double the size)
- $A \subseteq A \cup B$ and $B \subseteq A \cup B$. A is subset of $A \cup B$ and so is B
- $A \cup B = B \cup A$ Commutative Law for union
- $A \cup (B \cup C) = (A \cup B) \cup C$ Associative Law for union

Definition 2. Let A and B be sets. The **intersection** of A and B, denoted by $A \cap B$, is a set that consists of the elements that are in both A and B. With set builder notation,

$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}$$

Example

Let $A = \{4, 3, -1, 5, 6, 2, 7, -3\}$ and $B = \{10, 3, -2, 5, 4, 8, 0, -3\}$. Find $A \cap B$

$A \cap B$ is a set that contains the elements that belong to both set A and set B. Thus, $A \cap B = \{4, 3, 5, -3\}$

Following is Venn Diagram of $A \cap B$:

preview_nyai

The overlap area of both circles represents the set of $A \cap B$.

Again, there are a few things that are worth to mention:

- $A \cap A = A$ $A \cap A$ is equal to A
- $A \cap B \subseteq A$ and $A \cap B \subseteq B$ $A \cap B$ is a subset of A and $A \cap B$ is also a subset of B
- $A \cap B = B \cap A$ Commutative Law for intersection
- $A \cap (B \cap C) = (A \cap B) \cap C$ Associative Law for intersection

Those are simple facts, but they are very important properties for the intersection operation. Please take a moment to memorize them.

If two sets don't have common elements to share, then we call them disjoint sets. If two sets are disjoint, then the intersection of those sets is an empty set.

preview_z1zn

If A and B are disjoint, then $A \cap B = \emptyset$

Distributive Laws

Let A , B , and C be sets. Following two equations are referred as distributive laws for union and intersection:

- $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ Union over intersection
- $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ Intersection over union

"At this point, some of you may ask 'Why do we need to know the set as well as those set properties, operations, and laws?' Well, the set theorem is the foundation of logical thinking involved in mathematics and computer science. Without strong logical thinking, you can't become a good programmer let alone a computer scientist."

Following example also shows one of the useful properties of union and intersection operations, which we often refer as "Absorption Law"

Example

Let A and B be sets. Prove $A \cup (A \cap B) = A$

Most of us may not quite familiar with the way of proving a set equation, we may not know how to present convincing arguments to show why an equation is true. In this example, we demonstrate how to use definitions properties, and laws we have discussed in the previous lectures to make the arguments.

We first explain the logic used in the proof:

If we want to prove $A \cup (A \cap B) = A$, according to the definition of equality of two sets, we need to show the set $A \cup (A \cap B)$ is a subset of A , which means that we need to show $A \cup (A \cap B) \subseteq A$, and we also need to show A is a subset of $A \cup (A \cap B)$, which means that we need to show $A \subseteq A \cup (A \cap B)$.

According to the definition of subset, if we want to show $A \cup (A \cap B) \subseteq A$, we will need to convince people for each element in $A \cup (A \cap B)$, it is also an element of A . This is how we are going to argue: "If x is one of the element of $A \cup (A \cap B)$, then, according to the definition of the union (if you forgot about the definition of union, please review [Lecture 2.1](#)), x will be either in A or in $(A \cap B)$, but either way, x is in A (Also check the definition of intersection). At this point, we have provided enough arguments that prove $A \cup (A \cap B)$ is subset of A . What we need to do next is to convince people that A is also a subset of $A \cup (A \cap B)$.

According to the second property listed under the definition of union: $A \subseteq (A \cup B)$, we can claim $A \subseteq (A \cup (A \cap B))$. Done with the proof.

Now, we provide the formal proof of this equation.

Proof

$\forall x \in A, x \in (A \cup (A \cap B))$, thus $A \subseteq (A \cup (A \cap B))$. On the other hand, $\forall x \in (A \cup (A \cap B)), x \in A$ or $x \in (A \cap B)$, in either way, $x \in A$. Therefore, $A \cup (A \cap B) = A$

Please continue to study [Lecture 2.3](#) or click on the "Next" button down below:

2.3: Set Difference and Set Complement

Set Difference

Definition 1

Let A and B be sets. $A - B$ is the set that consists of all the elements in A but not in B.

A-B can also be written as $A \cap \overline{B}$

Example

Let $A = \{2, -5, 1, 6, 10, 0, 4\}$ and $B = \{3, 5, 4, -2, 5, 11, 8, 6\}$, then $A - B = \{2, -5, 1, 10, 0\}$

Let's also use a Venn Diagram to show the set $A - B$:

preview

The green area of the set A represents the set $A - B$, which means that everything falls in the green area is an element in set A - B.

In above Venn Diagram, there are four areas which are color coded by gray, green, army green, and red. We will talk about the area color coded by gray later.

- The green area represents the set $A - B$;
- The army green area represents the set $A \cap B$;
- The red area represents the set of $B - A$;
- The green area plus the army green area represents the set A;
- The army green area plus red area represents the set B
- The green area + the army green area + the red area represents the set $A \cup B$

To use areas to represent sets in a Venn Diagram is very helpful for understanding some of the properties of sets

Example

Let A and B be sets. One of the properties of sets says $A = (A - B) \cup (A \cap B)$. This is very useful property because it can be used to simplify the complex set expression $(A - B) \cup (A \cap B)$ to a single set expression A. But how do we know this is correct? Let's take a look at the following Venn Diagram:

preview_538k

The set $A - B$ is represented by the green area and $A \cap B$ is represented by the army green area. it is very clear that $(A - B) \cup (A \cap B)$ is represented by the areas of green area combined with army green area which is the same area that represents the set A.

Using a Venn Diagram is a very effective method for helping us to understand various set properties.

Complement Set

Let A be a set. In [Lecture 2.1](#), we have introduced the notation $\overline{A}$. We define $\overline{A}$ as a set that consists of all the elements except the ones that are in the set A. Here the phrase "all the elements" is not quite clear.

What do you mean by "all the elements"? All the elements of what?

When we say "All the elements" we really need to mention the set that we are talking about in this conversation. In other words, we will need to specify the set that all the elements belong to. Normally, the set we are referring to when we are saying "All the elements" is a set that contains all the elements that are in the concern. For example, "All the elements" could mean all the students that are currently enrolled in Fresno City College or could mean all the students that are in computer science major. "All the elements" could also mean all the numbers in a specific set and so on. The set that contains all the elements that are in

the concern is called a **Universal Set** or **Universes**. Therefore, the definition of $\overline{A}$ is depends on what universal set that we are referring to. For example, let $A = \{4, 5, 6\}$, then $\overline{A} = \{0, 2, 3, 7, 8, 9\}$ if the universal set U is $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, while $\overline{A}$ would be $\{x \mid x \in \mathbb{Z} \text{ except } 4, 5, 6\}$ if we use $\mathbb{Z}$ (the set of all integers) as the universal set.

As being discussed above, we have following more precise definition for complement set:

Definition 2

Let U be a universal set and A be a subset of U . The complement set of A , denoted by $\overline{A}$ is a set that contains all the elements in U except those in A . In other words, $\overline{A} = U - A$

In the following Venn Diagram, the gray area represents $\overline{A}$

preview_27r8

With a Venn Diagram, we can easily understand following two equations:

1. $\overline{A \cup B} = \overline{A} \cap \overline{B}$
2. $\overline{A \cap B} = \overline{A} \cup \overline{B}$

For equation 1:

preview_y152

In above Venn Diagram, the gray area represents the set on the left side of the equation: $\overline{A \cup B}$

Now, let's take a look the area that represents the set on the right side of the equation:

$\overline{A}$ is represented by the gray area + red area (anything that is outside of circle A)

$\overline{B}$ is represented by the gray area + green area (anything that is outside of circle B)

According to the definition of the intersection $\cap$

$\overline{A} \cap \overline{B}$ should be presented by the area that represents both $\overline{A}$ and $\overline{B}$ and that is gray area.

Conclusion: Since both $\overline{A \cup B}$ and $\overline{A} \cap \overline{B}$ are represented by the same piece of area in the Venn Diagram, they are the same set.

Similarly, we can use above Venn Diagram to verify the second equation. Those equations are the famous DeMorgan's Laws for set equality.

Please complete following exercises:

3: FUNCTIONS AND BINARY RELATIONS

3.1: Functions

3.2: Binary Relations

3.3: Reverse Relations and Images

3.4: Cardinality and Power Set

3.1: 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

$$f : A \rightarrow B$$

indicates that f is a function with domain, A , and codomain, B . The familiar notation " $f(a) = b$ " indicates that f assigns the element $b^{\text{int}} B$ 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

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

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:

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

$$\begin{aligned} f_5(0010) &= 3, \\ f_5(100) &= 1, \\ f_5(0000) &\text{ is undefined.} \end{aligned} \tag{3.1.1}$$

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 $f : A \rightarrow B$, 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,

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

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,

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

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 3.1.1

For functions $f : A \rightarrow B$ and $g : B \rightarrow C$, 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.

⁴There 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.

3.2: 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 $f : A \rightarrow B$ is a special case of binary relation in which an element $a^{\text{int}} A$ is related to an element $b^{\text{int}} B$ precisely when $b = f(a)$.

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

Definition 3.2.1

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 $fR : A \rightarrow B$ 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

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

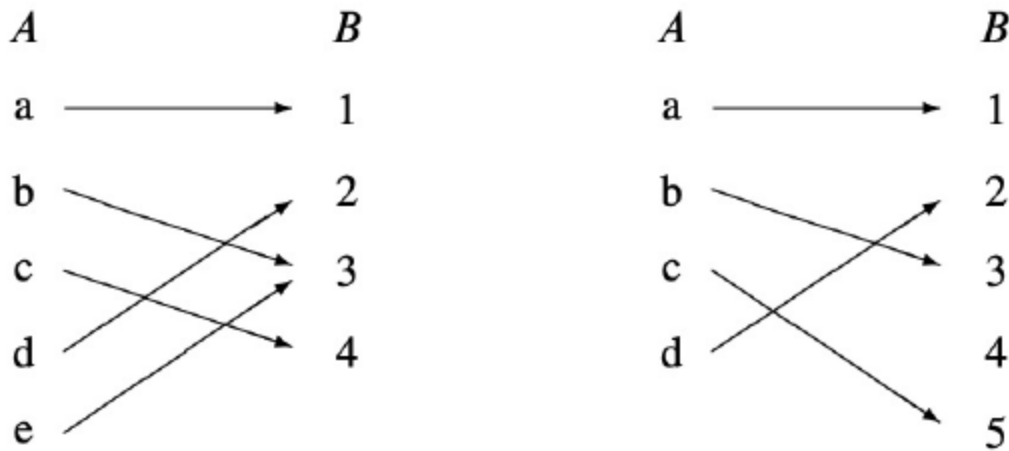
such that the faculty member named $\langle \text{instructor-name} \rangle$ is in charge of the subject with number $\langle \text{subject-num} \rangle$ that was offered in Spring ’10. So $\text{graph}(\text{Chrg})$ contains pairs like

Illegal pream-token (&)

Some subjects in the codomain, *SubNums*, do not appear among this list of pairs—that is, they are not in $\text{range}(\text{Chrg})$. 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 $\text{domain}(R)$ 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 3.2.2

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, $\text{graph}(R)$ alone does not determine whether R is total: we also need to know what $\text{domain}(R)$ is.

Example 3.2.3

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 3.2.4

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 $Chrg(\text{A. Meyer})$. 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 $\text{graph}(Chrg)$, 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 $Chrg(\{\text{G. Freeman}, \text{T. Eng}\})$. Looking again at the list (4.4), we see that

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

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 3.2.5

The inverse, R^{-1} of a relation $R: A \rightarrow B$ 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 3.2.6

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,

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

We can't assert equality here because there may be additional pairs further down the list showing that additional instructors are co-in-charge 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,

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

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

⁵Writing 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-than relation or the addition operation rather than prefix notation like " $< (m, n)$ " or " $+(m, n)$."

This page titled [3.2: Binary Relations](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [4.4: Binary Relations](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

3.3: Reverse Relations and Images

We begin this section with a review of the term "Image". The first appearance of the term "Image" was in Section 3.1 when we discussed the topic: Functions. Now, we are going to elaborate on this term a little bit more. We begin with following example.

Example

Let A be the set of all men and B be the set of all women. A relation $f : A \longrightarrow B$ is defined as "For each element x in A , assign an element y in B to x if y is the biological mother of x ".

The set A is called the domain of this relation and the set B is called the codomain of the relation. From the previous section, Section 3.2, we know this relation

- is a function because for each x in A , there is at most one y in B that can be assigned to x (For each man, there is only one biological mother)
- is total because for each x in A , there will be one y in B that is assigned to x (Each man will have a biological mother)

But this relation is not surjective. To be surjective, for each element y in B , there must be at least one element x in A such that y is assigned to x . That means for each woman in B , she has to be a biological mother of someone in A . Obviously, this is not true because some women may not have a son.

Now, the set B (the set of all women) can be divided into two subsets: one that contains all the women who have a son and the other one contains all the women who don't have a son. We use S to represent the set that contains all the women who have a son, then S is called the **image** of this relation, and we write it as $R(A) = S$ or $S = R(A)$.

An image of a relation is a subset of the codomain. It can be the same as the codomain if all the elements in the codomain can be assigned to an element in the domain. If this is the case, then the relation is surjective.

Example

Let A be the set of all real numbers and B also be the set of all real numbers. A relation $f : A \longrightarrow B$ is defined as "For each element x in A , assign an element y in B to x if $y = x^2$ ".

Both domain and codomain of this relation is the set of all real numbers. This relation

- is a function because, for each element x in A , there is at most one element in B that can be assigned to x
- is total because for each element x in A , there will be an element y in B such that $y = x^2$

But, it is not surjective because not every y in B can be assigned to an element in A . In other words, for some element y , we can't find an x in A such that $y = x^2$. For example, for $y = -1$, there isn't anything in A that will make $-1 = x^2$ be a true statement. Again, the codomain is divided into two sets: one that contains all the positive numbers y in B for which we can find an x in A such that $y = x^2$. We use S to denote this set. The other set contains all the negative numbers y in B for which, we can't find a number x in A such that $y = x^2$.

In this example, the subset S is the image of the relation. From algebra, we know the set S consists of all the numbers that are greater than or equal to 0. That is $S = \{x \in B : x \geq 0\}$

With the help of those two examples, hope you have grasped the concept of "image" for a given relation.

Now, let's tackle the concept of "Inverse Relation"

Definition

Let A and B be sets. A relation $f : A \longrightarrow B$ assigns an element y in B to an element x in A . The inverse of this relation is

also a relation which is denoted by $f^{-1} : B \longrightarrow A$ and is defined as "Assign an element y in A to an element x in B if x was assigned to y in the relation $f : A \longrightarrow B$ "

Such precise mathematical/logical definition is always very difficult to understand. It is like those laws written in a law book. It is not really for the normal people like you and me to read. Therefore, we need interpretations and examples to help us to understand the concepts. So are those mathematical/logical definitions.

Example

Let A be a set $\{a, b, c, d, e\}$ and B be a set $\{3, -1, 2, 4\}$. A relation $f : A \longrightarrow B$ is defined by following table:

preview

Then the inverse of this relation $f^{-1} : B \longrightarrow A$ will be defined as

preview_ozeb

From this example, we can see to get an inverse of a relation, all we need to do is switch the domain to codomain and switch the codomain to domain and at the meantime, reverse the directions of those arrows.

It is going to be even easier to get an inverse relation if we use a set of ordered pairs to represent a relation.

In this example, we can use following set of the ordered pairs to represent the original relation $f : A \longrightarrow B$

$$f : \{(a, -1), (b, 3), (b, 2), (c, 2), (d, 4), (e, 4)\}$$

To get the inverse of this relation, all we need to do now is to switch the order of two elements in each ordered pair:

$$f^{-1} : \{(-1, a), (3, b), (2, b), (2, c), (4, d), (4, e)\}$$

From this example we can also see the original relation $f : A \longrightarrow B$ is a function, but its inverse is not.

Example

Let A be the set of all men and B be the set of all women. A relation $f : A \longrightarrow B$ is defined as "For each element x in A , assign an element y in B to x if y is the biological mother of x ".

Now, the inverse of this relation $f^{-1} : B \longrightarrow A$ is defined as "For each element x in B (each woman in B), assign an element y in A (a man in A) to x if y is a son of x ". We can see the original relation is a function, but its inverse is not. The original relation is not surjective, but its inverse is. The original relation is total, but its inverse is not.

Example

Let A be the set of all real numbers and B also be the set of all real numbers. A relation $f : A \longrightarrow B$ is defined as "For each element x in A , assign an element y in B to x if $y = x^2$ "

The inverse of this relation $f^{-1} : B \longrightarrow A$ will be defined as "For each element x in B , assign an element y in A to x if $y^2 = x$ ". For example, for the element $x=16$ in B , we will assign the number $y=4$ in A to it because $4^2 = 16$.

This last example may be a challenge. Actually, it is not too bad. The original relation maps each number in the domain to its square and the inverse relation maps each number in the domain to its square root. The original relation is a function, but its inverse is not because each number in the domain has two different square roots: one is positive and the other is negative.

3.4: Cardinality and Power Set

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 3.4.1

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 $R : A \rightarrow B$ 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 $A \text{ surj } B$ to mean that there is a surjective function from A to B , then we've just proved a lemma: if $A \text{ surj } B$ 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 3.4.2

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

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

Lemma 4.5.3. For finite sets A, B :

1. If $A \text{ surj } B$, then $|A| \geq |B|$.
2. If $A \text{ inj } B$, then $|A| \leq |B|$.
3. If $A \text{ bij } B$, 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 $[\leq 1 \text{ out}]$, function property, and the $[\geq 1 \text{ in}]$, surjective property, then R^{-1} is total and injective, so $A \text{ surj } B$ iff $B \text{ inj } A$. 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

$$\begin{aligned} A &= \{a_0, a_1, a_2, a_3, a_4, a_5\} \\ B &= \{b_0, b_1, b_2, b_3\} \end{aligned} \tag{3.4.1}$$

Then define a total function $f : A \rightarrow B$ by the rules $f : A \rightarrow B$ 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 $0 \leq i \leq 5$. Since $5 \geq 3$, this f is a surjection.

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

Theorem 3.4.4

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

$$|A| \geq |B| \text{ iff } A \text{ surj } B, \quad (3.4.2)$$

$$|A| \leq |B| \text{ iff } A \text{ inj } B, \quad (3.4.3)$$

$$|A| = |B| \text{ iff } A \text{ bij } B. \quad (3.4.4)$$

How Many Subsets of a Finite Set?

As an application of the bijection mapping rule (3.4.4), we can give an easy proof of:

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

Theorem 3.4.5

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 $\{0,1\}^n$, the n -bit sequences. Namely, let $a_1, a_2, \dots, a_n$ be the elements of A . The bijection maps each subset of $S \subseteq A$ to the bit sequence $(b_1, b_2, \dots, b_n)$ 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.(3.4.4),

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

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

⁶In 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 [3.4: Cardinality and Power Set](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [4.5: Finite Cardinality](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

4: MATHEMATICAL INDUCTION

4.1: Introduction to Mathematical Induction

4.2: Proof by Using Mathematical Induction

4.3: Loop Invariant

4.1: Introduction to Mathematical 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

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

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 (4.1.1) 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 4.1.1

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

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} \quad (4.1.1)$$

To prove the theorem by induction, define predicate $P(n)$ to be the equation (4.1.1). 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 (4.1.1)—in order to prove $P(n+1)$, which is the equation

$$1 + 2 + 3 + \cdots + n + (n+1) = \frac{(n+1)(n+2)}{2} \quad (4.1.2)$$

These two equations are quite similar; in fact, adding $(n+1)$ to both sides of equation (4.1.1) and simplifying the right side gives the equation (4.1.2):

$$\begin{aligned} 1 + 2 + 3 + \cdots + n + (n+1) &= \frac{n(n+1)}{2} + (n+1) \\ &= \frac{(n+2)(n+1)}{2} \end{aligned} \quad (4.1.3)$$

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 $P(n)$ is true for all nonnegative integers, n . The theorem is proved.

A Template for Induction Proofs

The proof of equation (4.1.1) 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 (4.1.1). 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:

$$P(0) \rightarrow P(1), P(1) \rightarrow P(2), P(2) \rightarrow P(3), \dots$$

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 (4.1.1).

Base case: $P(0)$ is true, because both sides of equation (4.1.1) equal zero when $n = 0$.

Inductive step: Assume that $P(n)$ is true, that is equation (4.1.1) holds for some nonnegative integer n . Then adding $n + 1$ to both sides of the equation implies that

$$\begin{aligned} 1 + 2 + 3 + \dots + n + (n + 1) &= \frac{n(n + 1)}{2} + (n + 1) \\ &= \frac{(n + 1)(n + 2)}{2} \quad (\text{by simple algebra}) \end{aligned} \tag{4.1.4}$$

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 center³ 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.

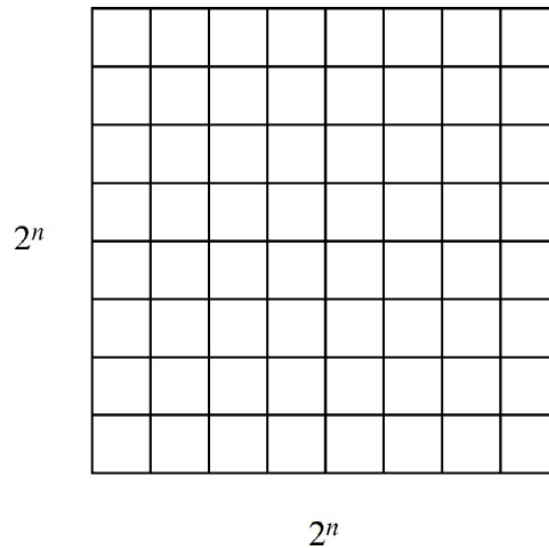


Figure 5.1 A $2^n \times 2^n$ courtyard for $n = 3$.

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.

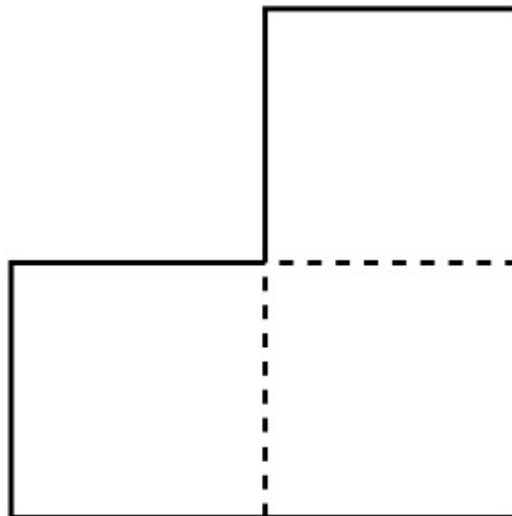


Figure 5.2 The special L-shaped tile.

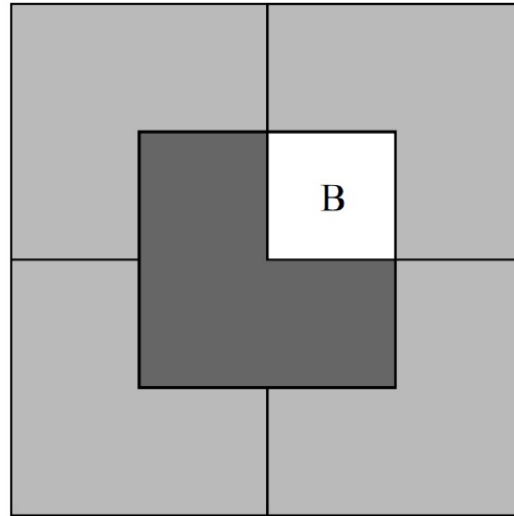


Figure 5.3 A tiling using L-shaped tiles for $n = 2$ with Bill in a center square.

Theorem 4.1.2

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

Proof available at the end of this section.

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.

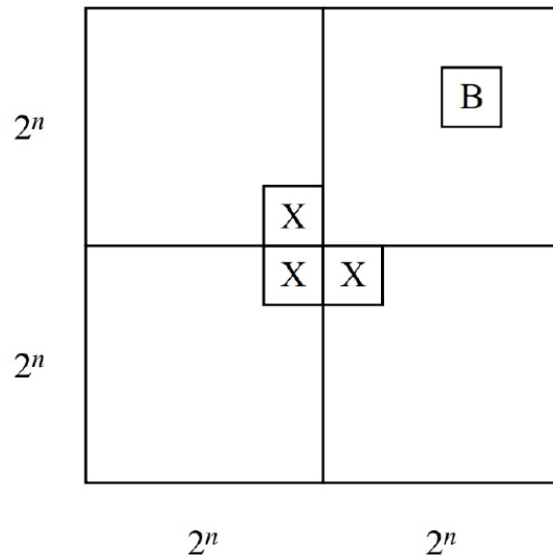


Figure 5.4 Using a stronger inductive hypothesis to prove Theorem 5.1.2.

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 than $n \geq 0$, 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

$$\text{In every set of } n \text{ horses, all are the same color.} \quad (4.1.5)$$

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:

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

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

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

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

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

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

So h_1 is the same color as the remaining horses besides h_{n+1} —that is, $h_2, \dots, h_n$. Likewise, h_{n+1} is the same color as the remaining horses besides h_1 —that is, $h_2, \dots, h_n$, again. Since h_1 and h_{n+1} are the same color as $h_2, \dots, h_n$, 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 $h_2, \dots, h_n$ ” The ellipsis notation (“...”) in the expression “ $h_1, h_2, \dots, h_n, h_{n+1}$ ” creates the impression that there are some remaining horses—namely $h_2, \dots, h_n$ —besides h_1 and h_{n+1} . However, this is not true when $n = 1$. In that case, $h_1, h_2, \dots, h_n, h_{n+1}$ 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 $P(2) \rightarrow P(3)$, $P(3) \rightarrow P(4)$, etc. But we failed to prove $P(1) \rightarrow P(2)$, 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.

¹But see Section 5.3.

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

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

⁴5-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.

Answers

Theorem 4.1.2

(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 ... ■

This page titled [4.1: Introduction to Mathematical Induction](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [5.1: Ordinary Induction](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

4.2: Proof by Using Mathematical Induction

In this section, we will continue the discussion on mathematical induction. Mathematical induction is a specific logic that people use for making an assertion or proving the truth value of a proposition.

Example

If we know the first object in a sequence of objects is a circle and we also know that if any one of the objects in this sequence is a circle, then next object in the sequence is also a circle. Then we can claim that all the objects in this sequence are circles.

We don't use this logic in our daily life very often, so it may not sound very convincing. But, it is used quite often in discrete math and computer science for making an argument or proving a proposition.

Example

Use induction to show that $5^n - 1$ is divisible by 4 for all $n \geq 1$

First, we should understand the question. Here, $5^n - 1$, $n = 1, 2, 3, \dots, n, n+1, \dots$ is a sequence of numbers:

4, 24, 124, ...

What we want to prove is: "All the numbers in this sequence are divisible by 4". To use mathematical induction, we need to prove

1. The first number in the sequence is divisible by 4
2. If any one of the numbers in the sequence is divisible by 4, then the next number in the sequence will be divisible by 4

If we can prove above two propositions are true, then we have proved the proposition "All the numbers in the sequence are divisible by 4"

The logic we use here is exactly the same as we did in the previous example: If we tell people

1. The first object in the sequence is a circle
2. If any one of the objects in the sequence is a circle, then the next object in the sequence will be a circle

We actually tell people "All the objects in the sequence are circles"

Now, go back to our second example, let's prove following propositions:

1. The first number in the sequence is divisible by 4
2. If any one of the numbers in the sequence is divisible by 4, then the next number in the sequence will be divisible by 4

Since the first number in the sequence is 4, it is divisible by 4.

We can use $5^n - 1$ to represent any one of the numbers in this sequence because that is how the sequence is formed.

The second proposition says "If $5^n - 1$ is divisible by 4, then the next number in the sequence, which is $5^{n+1} - 1$ is also divisible by 4" We want to prove this proposition is true.

Proof.

$5^{n+1} - 1 = 5 \cdot 5^n - 1 = 4 \cdot 5^n + 5^n - 1$. According to our assumption, $5^n - 1$ is divisible by 4 and it is easy to see $4 \cdot 5^n$ is divisible by 4, therefore, $(4 \cdot 5^n) + (5^n - 1)$ is divisible by 4, and $4 \cdot 5^n + 5^n - 1$ is $5^{n+1} - 1$. Proof is completed.

Example

Prove the predicate " $n^3 + 2n$ is a multiple of 3" is true for any positive integer n .

We will use the same logic we used in the previous examples to prove this predicate is true. Again, here we are talking about a sequence of numbers:

3, 12, 33, ...

What we want to do is to prove all the numbers in this sequence are multiple of 3 (all the numbers are divisible by 3). To use mathematical induction, we need to prove following two propositions:

1. The first number in the sequence is a multiple of 3
2. If any one of the number in the sequence is a multiple of 3, then the next number in the sequence is also a multiple of 3

It is obvious that the first number in the sequence is a multiple of 3.

Now, we let n^3+2n to represent any one of the numbers in the sequence, then the next number in the sequence will be $(n+1)^3+2(n+1)$. The second proposition says: "If n^3+2n is a multiple of 3, then $(n+1)^3+2(n+1)$ is also a multiple of 3"

We will need to fresh our algebra skills a little bit.

In algebra, $(n+1)^3 = n^3 + 3n^2 + 3n + 1$ (You can check it out from your old algebra book or simply go to ChatGPT to find out).

Thus,

$$\begin{aligned}(n+1)^3+2(n+1) &= n^3 + 3n^2 + 3n + 1 + 2(n+1) \\ &= n^3 + 3n^2 + 3n + 1 + 2n + 2 \\ &= (n^3 + 2n) + 3n^2 + 3n + 3 \\ &= (n^3 + 2n) + 3(n^2 + n + 1)\end{aligned}$$

From the last expression, $(n^3 + 2n)$ is multiple of 3 (that is our assumption). Since $(n^2 + n + 1)$ is an integer, $3(n^2 + n + 1)$ must be a multiple of 3. Therefore, $(n^3 + 2n) + 3(n^2 + n + 1)$ must be a multiple of 3 and that is $(n+1)^3+2(n+1)$. The proof is completed

The way we prove this predicate may seem a little bit difficult for you to understand. It is because it requires some algebraic skills which you may forgot. And the most difficult is the induction logic we used in the proving process. But, the logic we used in this proof is exactly the same as the logic we used in the previous examples.

4.3: Loop Invariant

Mathematical induction is a foundation for the logic we use when we are using a looping statement in a program. In our programming class, we are frequently asked to use a looping statement in a program, most of times, we code the program with a looping statement, and then test it with a few sample data. If the looping statement produces correct results with those testing data, we simply accept the correctness of the looping statement we use and assume it will work for all cases.

Example

Write a function that receives a positive integer (a natural number), and then produces the sum of squares of all the numbers from 0 to the number the function receives. For example, if the function receives 5, then the function should produce $0^2+1^2+2^2+3^2+4^2+5^2$ which is 55

This is a simple programming assignment; the code is shown as following:

preview

We normally check the correctness of this function by entering a few of testing data, such as 5, 6, and 7 in the main(). If the results are all correct, then we assume this function will work for any positive integers we may enter. For some simple calculation, like this one, it will be fine. However, for some complicated calculation, the correctness may not be obvious. Especially, if the result of a function is related to a life and death situation, then we can't just accept the correctness of the function by the results that come from a few of sample data. We will need a solid proof.

In this example, even we don't have to, we would like to introduce the way of proving the correctness of the looping statement we used in the function **sumSquares()**.

First, we explain what is a "Loop Invariant"

Definition

A loop invariant is a predicate that is true before the loop begins and it is also true after each iteration of the loop.

For the looping statement we use in the function **sumSquares()**, consider the predicate " $s = 0^2+1^2+2^2+3^2, \dots, + i^2$ ", where **i** is the value of the looping variable after each iteration", if we can prove it is a loop invariant, then it will be true before the looping statement begins and it also true after each iteration. In particular, it will be true after the last iteration. That means $s = 0^2+1^2+2^2+3^2, \dots, + n^2$ is true and this proves the looping statement produces the result we expect.

However, the process of proving is not always simple, but we want to get it a try.

Proof

To prove the predicate " $s = 0^2+1^2+2^2+3^2, \dots, + i^2$ " is a loop invariant, we need to prove it is true before the loop begins and it is also true after each iteration.

Before the loop begins, $s = 0$ and $i = 0$, so the predicate " $s = 0^2$ " is true.

Now, we need to prove the predicate is also true after each iteration of the loop. We will use mathematical induction for the proof.

It is not too difficult to see that, after the first iteration, the value of the looping variable **i** is 1, and, since $s=s+i*i$, $s=1$. Thus $s = 0^2+1^2$, so the predicate is true

Assume that the predicate is true after the **i** iterations, that means, after **i** iterations, $s = 0^2+1^2+2^2+3^2, \dots, + i^2$. Now we need to prove that, after **i+1** iterations, we have $s = 0^2+1^2+2^2+3^2, \dots, + (i+1)^2$

In the **i**th iteration, the value of the looping variable is **i** and in the **(i+1)th** iteration, the value of the looping variable will be **i+1**,

thus, after $i+1$ iterations, $s = s + (i+1)(i+1)$. On the right side of equal sign, the value of s is $0^2+1^2+2^2+3^2, \dots, + i^2$, thus, after the $i+1$ iterations, $s = 0^2+1^2+2^2+3^2, \dots, + i^2 + (i+1)(i+1) = 0^2+1^2+2^2+3^2, \dots, + (i+1)^2$. The proof is completed.

With such strict proof, we can make sure the loop we use produces correct result for any given positive integer.

It may seem difficult for you to grasp the logic we use in this proof. No worries, we can't become an expert after one logic class. As you become mathematically mature, you will have much more strength to handle such task.

What we do need to know here is we can't prove the correctness of a looping statement just by testing it with sample data. Sometimes, even with a large set of sample data, we still can't make sure a looping statement will produce the correct result.

Example

Following program contains a user-defined function `getLargest()` which receives an integer array along with its length, and then returns the largest number in the array it receives

preview_st8v

In the `main()`, we test this function with an array, it works fine. The function produces correct result for this sample data. We can test the function with over hundreds of sample data. Even the results are all correct, we still can't say the function will work for all integer arrays. We can easily find an integer array, such as `y[5] = {-5, -2, -10, -4, -7}`, which the function will produce an incorrect result.

Prove the correctness of programs is a big topic and mathematical induction is only one of the tools we can use in this area.

5: ALGORITHMS

5.1: Introduction to Algorithm

5.2: More Examples of Algorithms

5.1: Introduction to Algorithm

We have frequently used the word **algorithm** in our programming class to mean the idea, or the method of solving a programming problem. More precisely, an algorithm is a step-by-step method of solving some problem. In computer science, algorithm typically refers to a solution that can be executed by a computer.

Algorithms typically have the following characteristics:

- An algorithm normally requires some input data, which means an algorithm should specify what kind of data it works on
- An algorithm normally produces an output. Such output can be multiple data values, multiple data types. It can also be a display
- The steps in an algorithm are precisely stated
- The intermediate results of each step of execution are unique and are determined only by the inputs and results of the preceding steps. It means, if an algorithm receives the same set of inputs, it should produce the same set of outputs.
- The output produced by an algorithm should be correct
- The algorithm should be applied to a set of inputs. In other words, an algorithm should not just work for a specific input.

But, what exactly an algorithm is? Let's take a look at an example.

Example

Suppose we are given a program task: "Write a user-defined function that receives three numbers and returns the maximum of those three numbers the function receives."

Before we proceed, we need to note:

1. We don't have to write an algorithm for each programming task before the coding. It is just like we don't have to have speech notes before we give a speech. But, preparing notes for a speech will definitely help us to give a better speech.
2. We may have different ways of solving a programming problem. Thus, there may be different algorithms for a single programming task.

The programming task given in this example can be accomplished in many different ways, therefore, we can write different algorithms for this programming task. Here we only provide one of them which is considered as the best.

Algorithm of Finding the Maximum of Three Numbers

Input: three numbers a, b, and c

Output: The largest of a, b, and c

1. *large = a*
2. *if b > large, then large = b;*
3. *if c > large, then large = c;*

return large

Of course, we don't really need to write such algorithm to code the function. But, by studying the algorithm of this simple programming task which we already know how to code, we will understand what an algorithm is and how it should be written.

Writing an algorithm is like writing a program in normal language. But, not exactly. It is more like to write a plan for a task.

Writing an algorithm is providing a solution to a programming problem. One of important benefits of writing an algorithm as the solution is the algorithm we provide is not programming language specific. It can be used to implement the programming solution with different programming languages. For example, if we provide the solution of this task with C++ code directly, then the people who are using python may not understand what we are talking about.

The idea of above algorithm is to inspect the numbers one by one and copy the largest value seen into a variable **large**. At the

conclusion of the algorithm, the value of the variable **large** will then be equal to the largest of the three numbers.

It is easy to understand this algorithm because we already know how to solve this problem before we see the algorithm. For some unfamiliar programming tasks, even someone provides the solution in the form of an algorithm, it may still be difficult to understand what the algorithm is talking about. To be a good programmer, following two abilities are essential:

1. Understand an algorithm written by someone else
2. To be able to write an algorithm for the solution of a programming problem.

One of the common way to study an algorithm is to execute the algorithm for some specific values of the variables in the algorithm. Such simulation is called a **trace**. In the process of the tracing, we record the values of each variable at each step.

Here, we show how to trace the algorithm presented in this example.

First suppose that

$a = 1, b = 5, c = 3$

At line 1: we set the variable **large** to a , so the value of the variable **large** now is 1

At line 2: we compare the value of the next number, b , with the value of the variable **large**. If $b > \text{large}$, then change the value of **large** to b , otherwise, do nothing. Since $b = 5, \text{large} = 1$, we change the value of **large** to 5

At line 3: we compare the value of the third number, c , with the value of the variable **large**. If $b > \text{large}$, then change the value of **large** to c , otherwise, do nothing. Since $c = 3, \text{large} = 5$, we don't do anything.

At this point, we can see the algorithm ensures the value of the variable **large** holds the largest numbers of a, b , and c .

For an unfamiliar algorithm or a more complicated algorithm, we often need to trace the algorithm several times with different sets of input data.

Although ordinary language is sometimes adequate to specify an algorithm, most programmers and computer scientists prefer a so called **pseudocode** because of its precision, structure, and universality. Pseudocode is so named because it resembles the actual code of computer languages such as C++, Java.

Here we provide the pseudocode version of the algorithm we presented in this example:

Algorithm of Finding the Maximum of Three Numbers (written in pseudocode)

This algorithm finds the largest of the numbers a, b , and c

Input: three numbers a, b , and c

Output: large (the largest of a, b , and c)

```
1. maxThree(a, b, c){
2.   large = a;
3.   if(b>large) //if b is greater than large, update large
4.     large=b
5.   if(c>large) //if c is greater than large, update large
6.     large=c;
7.   return large
8. }
```

As the pseudocode of the algorithm, here we

- Given a title to the algorithm
- Provide a brief description of the algorithm
- List the input and outputs of the algorithm
- Provide a function that contains step by step instructions

- for convenience, label each line of the pseudocode.

Example

Programming Task: "Find the largest number in a given sequence" Now, let's write an algorithm for this task.

Algorithm Title: Finding the Maximum Value in a Sequence

Algorithm Description: This algorithm finds the largest of the numbers $s_1, s_2, \dots, s_n$

List of Inputs and output: The sequence s , which represents $s_1, s_2, \dots, s_n$ and size or the length of the sequence, n

Function:

```
getMax(s, n)
{
  large=s1
  for k=2 to n
  if (sk>large)
  large=sk
  return large
}
```

If we eliminate those labels and descriptions, the algorithm can be trimmed down to

Finding the Maximum Value in a Sequence

This algorithm finds the largest of the numbers $s_1, s_2, \dots, s_n$

Input: s, n

Output: large

```
getMax(s, n)
{
  large=s1
  for k=2 to n
  if (sk>large)
  large=sk
  return large
}
```

This algorithm is using the same logic as the one we used in the previous example. But this algorithm can be used for solving more general problem: "Find the largest value from any sequence of data"

5.2: More Examples of Algorithms

Example

Write an algorithm that returns the index of the first occurrence of a specific character in a string (a sequence of characters): $s_1, s_2, \dots, s_n$

Here is the word "index" is referring to the position of a character in a sequence. For example, the index of 'i' in the sequence "abcd&dcba" is 5

Algorithm: Character Search

This algorithm is used for finding the index of the first occurrence of a specific character in a string

Input: a string of character $s = s_1, s_2, \dots, s_n$ and a character ch

Output: k (an integer that is the index of the first occurrence of ch in the string s) If the character ch is not in the string s , then the output is $\emptyset$;

```
getIndex(s, ch)
{
  for k=1 to n
  if( $s_k==ch$ )
  return k;
  return  $\emptyset$ ;
}
```

Example

Write an algorithm that returns the index of the last occurrence of a specific character in a string (a sequence of characters): $s_1, s_2, \dots, s_n$

Algorithm: Find the index of the last occurrence of a specific character in a string

This algorithm is used for finding the index of the last occurrence of a specific character in a string

Input: a string of character $s = s_1, s_2, \dots, s_n$ and a character ch

Output: index (an integer that is the index of the last occurrence of ch in the string s) If the character ch is not in the string s , then $index=\emptyset$;

```
getIndex(s, ch)
{
  int index
  for k=1 to n
  if( $s_k==ch$ ){
    index=k;
  }
  return index;
}
```

Examples presented above are algorithms of simple programming tasks. It seems trivial if we need to write those algorithms for the tasks. Unfortunately, it is not always the case. Let's study an algorithm that is slightly more difficult than the previous two examples.

Searching Algorithm

One of major computer programming task is searching. Examples are searching a student record from a school data file, searching a website from the internet, searching a book from a library and so on. All of those tasks can be stated as a searching problem. Here we will discuss an algorithm to solve a text-searching problem.

Example

Suppose we are given a text document and look for the first occurrence of a specific word. Instead of coding a program directly, we write an algorithm that provides a solution to this searching problem.

Algorithm: Word Search

This algorithm searches for an occurrence of a word W in a text document D . It returns the smallest index i such that W occurs in D starting at index i . If the word W does not occur in D , it returns $\emptyset$.

Input: W (indexed from 1 to m), m , D (indexed from 1 to n), n

Output: i

```
1 word_search( $W, m, D, n$ )
2 {
3   for  $i=1$  to  $n-m+1$  {
4      $j=1$ 
5     while( $D_{i+j-1}==W_j$ ){
6        $j++$ ;
7       if( $j>m$ )
8         return  $i$ 
9     }
10  }
11  return  $\emptyset$ 
12 }
```

If we are in the real world and facing such programming task. That is all we will get as a guideline for coding the program. It will leave us to figure out the logic of the algorithm. Since we are now in the classroom situation and we are learning it, we will provide further assistance to explain the logic of this algorithm.

1. All the characters in the document D are indexed as D_i , with $i=1, 2, 3, \dots, n$ and all the characters in the word W are indexed as W_j , with $j=1, 2, 3, \dots, m$
2. Inputs of the function are: the text document. It may be an array with n elements or a Linked list with n nodes or some other types of data structures; the length of the array, which is n ; the word that we are looking for. It may also be an array with m elements or a linked list with m nodes or some other types of data structures; the length of the array, which is m . So the inputs are D, n, W, m
3. Output: i , which is the index of the first character of the word when it first occurs in the document.
4. Line 1 specifies the name of the function as well as its inputs. The function has total of four inputs: The word that we are searching for, the length of the word, the document that we will be searching, the length of the document, which is the total number of characters in the document.
5. From line 3 to line 9, it is a nested looping statement with an outer for loop and an inner while loop with outer looping variable i and an inner looping variable j
6. The outer loop varies the value of i from 1 to $n-m+1$ because the number of characters in the word is m , if the word is in the document, then the index of the first character in the word can not be greater than $n-m+1$. For example, if the document has 100 characters, then $n=100$. If the word that we are searching for has 5 characters, then $m=5$. So, the largest possible value of i can not be greater than $100-5+1$, which is 96. Since i is the index of the first character in the word, and word has 5 characters, if $i=97$, then we won't have the full word occurs in the document.
7. The nested looping statement begins with D_1 , which is the first character in the document. The inner while loop compares this character with each character in the word W_j with $j=1, 2, \dots, m$. If there is a match, then compare with the next character in the word ($j=j+1$). If there is a mismatch, inner loop stops, increase the value i (which means moving to the next character in the document) and then repeat the inner loop (which is comparing the next character in the document with each character in the word as we did in the previous step)
8. If one of the character and onward four consecutive characters all match, we would have $j>m$, then the function returns i , which will be the index of the first character of the word when it first occurs in the document.

9. If the algorithm never return i , then it means the document does not contain the word, so the function returns $\emptyset$

Hope those explanations help you better understand the logic of the algorithm. You can also use some sample inputs to trace the algorithm.

6: CARDINALITY RULES

6.1: Counting One Thing by Counting Another

6.2: Counting Sequences

6.3: The Generalized Product Rule

6.4: The Division Rule

6.5: Counting Subsets

6.6: Sequences with Repetitions

6.7: Counting Practice - Poker Hands

6.8: The Pigeonhole Principle

6.9: Inclusion-Exclusion

6.10: Combinatorial Proofs

This page titled [6: Cardinality Rules](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [14: Cardinality Rules](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

6.1: 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:

$$\begin{aligned} A &= \text{all ways to select a dozen donuts when five varieties are available} \\ B &= \text{all 16-bit sequences with exactly 4 ones} \end{aligned} \quad (6.1.1)$$

An example of an element of set A is:

$\underbrace{00}_{\text{chocolate}} \quad \underbrace{\quad}_{\text{lemon-filled}} \quad \underbrace{000000}_{\text{sugar}} \quad \underbrace{00}_{\text{glazed}} \quad \underbrace{00}_{\text{plain}}$

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:

$\underbrace{00}_{\text{chocolate}} \quad 1 \quad \underbrace{\quad}_{\text{lemon-filled}} \quad 1 \underbrace{000000}_{\text{sugar}} 1 \quad \underbrace{00}_{\text{glazed}} \quad 1 \quad \underbrace{00}_{\text{plain}}$

and close up the gaps:

00110000000100100.

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 chocolate, l lemon-filled, s sugar, g glazed, and p plain

to the sequence:

$\underbrace{0 \dots 0}_{c} 1 \underbrace{0 \dots 0}_{l} 1 \underbrace{0 \dots 0}_{s} 1 \underbrace{0 \dots 0}_{g} 1 \underbrace{0 \dots 0}_{p}$

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 [6.1: 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 [Jin He](#).

- [14.1: Counting One Thing by Counting Another](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

6.2: 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 $P_1, P_2, \dots, P_n$ 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 $P_1, P_2, \dots, P_n$ 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:

$$\begin{aligned} B &= \{\text{pancakes, bacon and eggs, bagel, Doritos}\} \\ L &= \{\text{burger and fries, garden salad, Doritos}\} \\ D &= \{\text{macaroni, pizza, frozen burrito, pasta, Doritos}\} \end{aligned} \tag{6.2.1}$$

Then $B \times L \times D$ is the set of all possible daily diets. Here are some sample elements:

(pancakes, burger and fries, pizza)

(bacon and eggs, garden salad, pasta)

(Doritos, Doritos, frozen burrito)

The Product Rule tells us how many different daily diets are possible:

$$\begin{aligned} |B \times L \times D| &= |B| \cdot |L| \cdot |D| \\ &= 4 \cdot 3 \cdot 5 \\ &= 60. \end{aligned} \tag{6.2.2}$$

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 transformed 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 $A_1, A_2, \dots, A_n$ are disjoint sets, then:*

$$\begin{aligned} |A_1 \cup A_2 \cup \dots \cup A_n| &= |A_1| + |A_2| + \dots + |A_n| \\ |C \cup I \cup S| &= |C| + |I| + |S| \\ &= 20 + 40 + 60 \\ &= 120 \text{ days.} \end{aligned} \tag{6.2.3}$$

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.

$$\begin{aligned} F &= \{a, b, \dots, z, A, B, \dots, Z\} \\ S &= \{a, b, \dots, z, A, B, \dots, Z, 0, 1, \dots, 9\} \end{aligned} \tag{6.2.4}$$

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:

$$\begin{aligned} |(F \times S^5) \cup (F \times S^6) \cup (F \times S^7)| &= |F \times S^5| + |F \times S^6| + |F \times S^7| && \text{Sum Rule} \\ &= |F| \times |S|^5 + |F| \times |S|^6 + |F| \times |S|^7 && \text{Product Rule} \\ &= 52 \cdot 62^5 + 52 \cdot 62^6 + 52 \cdot 62^7 \\ &\approx 1.8 \cdot 10^{14} \text{ different passwords.} \end{aligned} \tag{6.2.5}$$

¹The notation S^5 means $S \times S \times S \times S \times S$.

6.3: 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:

“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) $\text{int } P^3$. 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:

$$S = \{(x, y, z) \text{ int } P^3 \mid x, y, \text{ and } z \text{ are different people}\}$$

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,

- n_k possible k th 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).

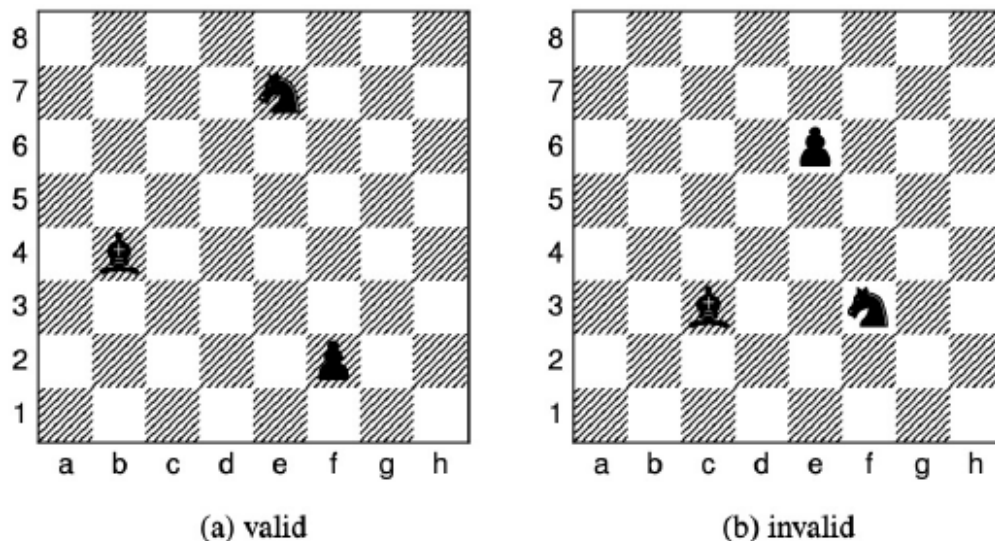


Figure 14.1 Two ways of placing a pawn, a knight, and a bishop on a chessboard. The configuration shown in (b) is invalid because the bishop and the knight are in the same row.

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 $\{a, b, c\}$:

$$\begin{array}{ccc} (a, b, c) & (a, c, b) & (b, a, c) \\ (b, c, a) & (c, a, b) & (c, b, a) \end{array} \quad (6.3.1)$$

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 [6.3: The Generalized Product Rule](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by Jin He.

- [14.3: The Generalized Product Rule](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

6.4: 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 $f : A \rightarrow B$ 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).

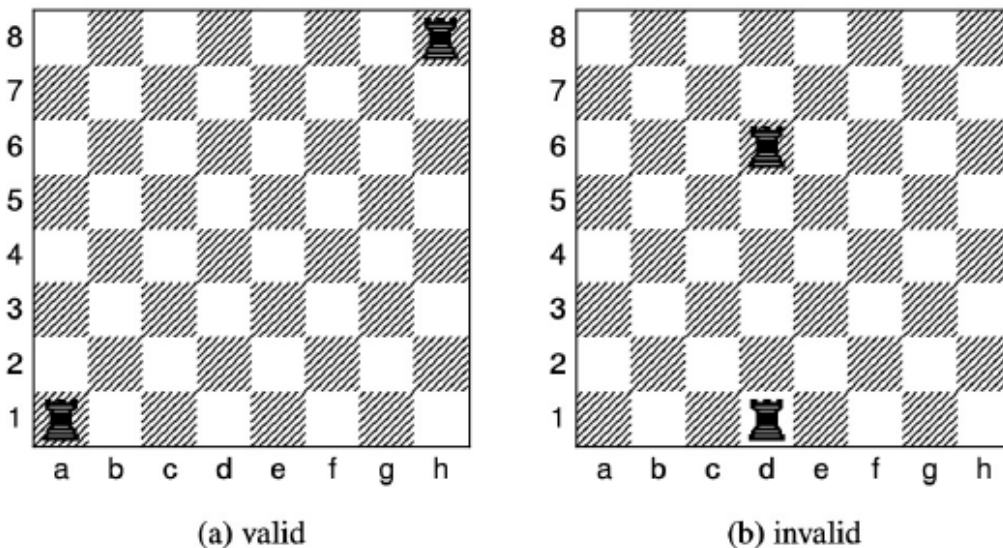


Figure 14.2 Two ways to place 2 rooks on a chessboard. The configuration in (b) is invalid because the rooks are in the same column.

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:

$$|B| = \frac{|A|}{2} = \frac{(8 \cdot 7)^2}{2}.$$

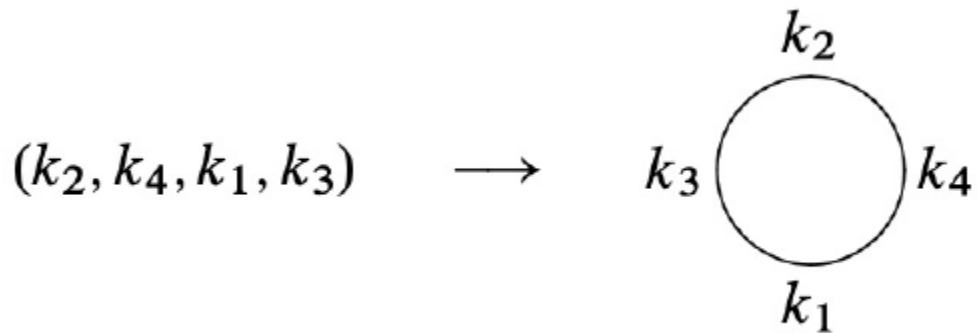
On the second line, we've computed the size of A using the General Product Rule just as in the earlier chess problem.

Knights of the Round Table

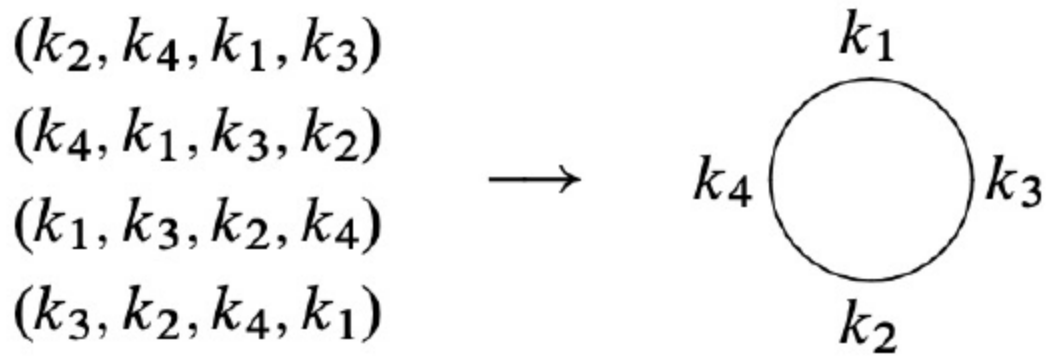
In how many ways can King Arthur arrange to seat his n different knights at his round table? A seating defines who sits where. Two seatings are considered to be the same *arrangement* if each knight sits between the same two knights in both seatings. An equivalent way to say this is that two seatings 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 arrangements 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 '#'

6.5: 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 $\{a_1, \dots, a_n\}$ into a k -element subset simply by taking the first k elements of the permutation. That is, the permutation $a_1 a_2 \dots a_n$ will map to the set $\{a_1, a_2, \dots, a_k\}$.

Notice that any other permutation with the same first k elements $a_1, \dots, a_k$ 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 $\{a_1, a_2, \dots, a_k\}$ if its first k elements are the elements $a_1, \dots, a_k$ 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 convention² 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 $\{x_1, x_2, \dots, x_8\}$ and the associated 8-bit sequence:

$$\begin{aligned} &\{x_1, \quad x_4, x_5 \quad\} \\ &(1, 0, 0, 1, 1, 0, 0, 0) \end{aligned}$$

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*

$$\binom{n + (k - 1)}{n}.$$

²We don't use it here, but a *sum* of zero terms equals 0.

- 14.5: Counting Subsets by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed CC BY-NC-SA 4.0.

6.6: 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 $k_1, k_2, \dots, k_m$ be nonnegative integers whose sum is n . A $(k_1, k_2, \dots, k_m)$ -split of A is a sequence

$$(A_1, A_2, \dots, A_m)$$

where the A_i are disjoint subsets of A and $|A_i| = k_i$ for $i = 1, \dots, m$.

To count the number of splits we take the same approach as for the Subset Rule. Namely, we map any permutation $a_1 a_2 \dots a_n$ of an n -element set A into a $(k_1, k_2, \dots, k_m)$ -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, $\dots$, and the m th subset of the split be the final k_m elements of the permutation. This map is a $k_1! k_2! \dots k_m!$ -to-1 function from the $n!$ permutations to the $(k_1, k_2, \dots, k_m)$ -splits of A , so from the Division Rule we conclude the *Subset Split Rule*:

Definition 6.6.1

For $n, k_1, \dots, k_m \in \mathbb{N}$, such that $k_1 + k_2 + \dots + k_m = n$, define the *multinomial coefficient*

$$n \text{ choose } k_1, k_2, \dots, k_m ::= \frac{n!}{k_1! k_2! \dots k_m!}.$$

Rule 14.6.2 (Subset Split Rule). The number of $(k_1, k_2, \dots, k_m)$ -splits of an n -element set is

$$\binom{n}{k_1, \dots, k_m}.$$

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 $\{1, 2, \dots, 10\}$. 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 $l_1, \dots, l_m$ 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

$$\binom{k_1 + k_2 + \dots + k_m}{k_1, \dots, k_m}.$$

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

$$\begin{aligned} (a + b)^4 = & aaaa + aaab + aaba + aabb \\ & + abaa + abab + abba + abbb \\ & + baaa + baab + baba + babb \\ & + bbaa + bbab + bbb a + bbbb \end{aligned} \quad (6.6.1)$$

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 6.6.4

(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 6.6.5

(*Multinomial Theorem*). For all $n \in \mathbb{N}$,

$$(z_1 + z_2 + \cdots + z_m)^n = \sum_{\substack{k_1, \dots, k_m \in \mathbb{N} \\ k_1 + \cdots + k_m = n}} \binom{n}{k_1, k_2, \dots, k_m} z_1^{k_1} z_2^{k_2} \cdots z_m^{k_m}.$$

6.7: 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:

$$\begin{aligned} &\{8\spadesuit, 8\diamondsuit, Q\heartsuit, 8\clubsuit\} \\ &\{A\clubsuit, 2\clubsuit, 2\heartsuit, 2\diamondsuit, 2\spadesuit\} \end{aligned} \tag{6.7.1}$$

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:

$$\begin{aligned} (8, Q, \heartsuit) &\leftrightarrow \{8\spadesuit, 8\diamondsuit, 8\heartsuit, 8\clubsuit, Q\heartsuit\} \\ (2, A, \clubsuit) &\leftrightarrow \{2\clubsuit, 2\heartsuit, 2\diamondsuit, 2\spadesuit, A\clubsuit\} \end{aligned} \tag{6.7.2}$$

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:

$$\begin{aligned} &\{2\spadesuit, 2\clubsuit, 2\diamondsuit, J\clubsuit, J\diamondsuit\} \\ &\{5\diamondsuit, 5\clubsuit, 5\heartsuit, 7\heartsuit, 7\clubsuit\} \end{aligned} \tag{6.7.3}$$

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:

$$\begin{aligned} (2, \{\spadesuit, \clubsuit, \diamondsuit\}, J, \{\clubsuit, \diamondsuit\}) &\leftrightarrow \{2\spadesuit, 2\clubsuit, 2\diamondsuit, J\clubsuit, J\diamondsuit\} \\ (5, \{\diamondsuit, \clubsuit, \heartsuit\}, 7, \{\heartsuit, \clubsuit\}) &\leftrightarrow \{5\diamondsuit, 5\clubsuit, 5\heartsuit, 7\heartsuit, 7\clubsuit\} \end{aligned} \tag{6.7.4}$$

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:

$$\begin{aligned} &\{3\spadesuit, 3\heartsuit, Q\clubsuit, Q\spadesuit, A\clubsuit\} \\ &\{9\heartsuit, 9\spadesuit, 5\heartsuit, 5\clubsuit, K\spadesuit\} \end{aligned} \tag{6.7.5}$$

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:

$$\begin{aligned} (3, \{\spadesuit, \heartsuit\}, Q, \{\clubsuit, \spadesuit\}, A, \clubsuit) &\searrow \\ (Q, \{\spadesuit, \heartsuit\}, 3, \{\spadesuit, \heartsuit\}, A, \clubsuit) &\nearrow \\ &\{3\spadesuit, 3\heartsuit, Q\clubsuit, Q\spadesuit, A\clubsuit\} \\ (9, \{\heartsuit, \spadesuit\}, 5, \{\heartsuit, \clubsuit\}, K, \spadesuit) &\searrow \\ (5, \{\heartsuit, \clubsuit\}, 9, \{\heartsuit, \spadesuit\}, K, \spadesuit) &\nearrow \\ &\{9\heartsuit, 9\spadesuit, 5\heartsuit, 5\clubsuit, K\spadesuit\} \end{aligned}$$

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 $f : A \rightarrow B$ 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 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:

$$\begin{aligned} (\{3, Q\}, \{\diamond, \spadesuit\}, \{\diamond, \heartsuit\}, A, \clubsuit) &\leftrightarrow \{3\diamond, 3\spadesuit, Q\diamond, Q\heartsuit, A\clubsuit\} \\ (\{9, 5\}, \{\heartsuit, \clubsuit\}, \{\heartsuit, \diamond\}, K, \spadesuit) &\leftrightarrow \{9\heartsuit, 9\diamond, 5\heartsuit, 5\clubsuit, K\spadesuit\} \end{aligned} \quad (6.7.6)$$

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:

$$\{7\diamond, K\clubsuit, 3\diamond, A\heartsuit, 2\spadesuit\}$$

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:

$$(7, K, A, 2, \diamond, 3) \leftrightarrow \{7\diamond, K\clubsuit, A\heartsuit, 2\spadesuit, 3\diamond\}.$$

Are there other sequences that correspond to the same hand? There is one more! We could equally well regard either the $3\diamond$ or the $7\diamond$ as the extra card, so this is actually a 2-to-1 mapping. Here are the two sequences corresponding to the example hand:

$$\begin{array}{ccc} (7, K, A, 2, \diamond, 3) & \searrow & \{7\diamond, K\clubsuit, A\heartsuit, 2\spadesuit, 3\diamond\} \\ (3, K, A, 2, \diamond, 7) & \nearrow & \end{array}$$

Therefore, the number of hands with every suit is:

$$\frac{13^4 \cdot 4 \cdot 12}{2}.$$

³There are 52 cards in a standard deck. Each card has a *suit* and a *rank*. There are four suits:

♠ (spades) ♥ (hearts) ♣ (clubs) ♦ (diamonds)

And there are 13 ranks, listed here from lowest to highest:

Ace
A, 2, 3, 4, 5, 6, 7, 8, 9, Jack *J*, Queen *Q*, King *K*.

Thus, for example, 8♥ is the 8 of hearts and *A*♠ is the ace of spades.

This page titled [6.7: Counting Practice - Poker Hands](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by Jin He.

- [14.7: Counting Practice - Poker Hands](#) by Eric Lehman, F. Thomson Leighton, and Alerty R. Meyer is licensed [CC BY-NC-SA 4.0](#).

6.8: 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.

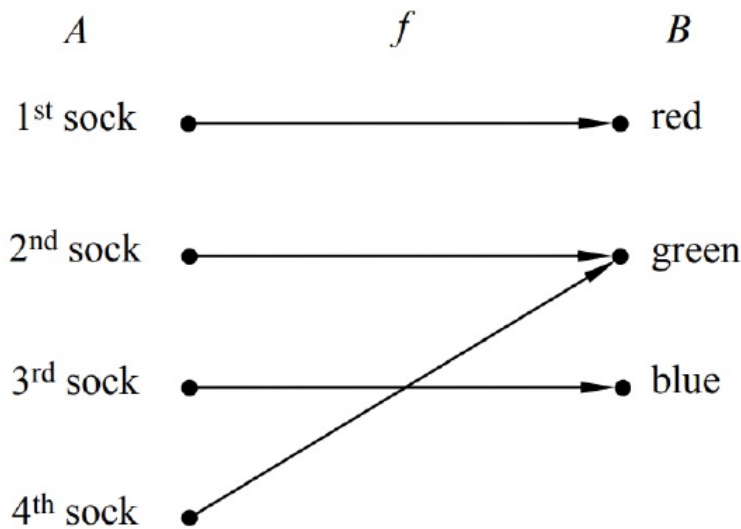


Figure 14.3 One possible mapping of four socks to three colors.

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 $f : A \rightarrow B$, 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 $f : A \rightarrow B$ 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 $B = \{10,000, 10,001, \dots, 200,000\}$, 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
5800949123548899122628663	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	423599683112377788211249
6949632451365987152423541	9137845566925526349897794
1763580219131985963102365	467093944574943964211220
712821143613619828415650	9153762966803189291934419
1826227795601842231029694	4815379351865384279613427
7173920083651862307925394	9270880194077636406984249
1843971862675102037201420	4837052948212922604442190
7215654874211755676220587	9324301480722103490379204
2396951193722134526177237	5106389423855018550671530
7256932847164391040233050	9436090832146695147140581
278139456826859801096354	5142368192004769218069910
7332822657075235431620317	9475308159734538249013238
2796605196713610405408019	518123409613014408401856
7426441829541573444964139	9492376623917486974923202
2931016394761975263190347	5198267398125617994391348
7632198126531809327186321	9511972558779880288252979
2933458058294405155197296	531792940316231219758372
7712154432211912882310511	96024134246191871255264
3075514410490975920315348	5384358126771794128356947
7858918664240262356610010	9631217114906129219461111
8149436716871371161932035	315769310532511284321993
311147498522793452860017	5439211712248901995423441
7898156786763212963178679	99081898510275335981319
3145621587936120118458701	5610379826092838192760458
814759101703757337848616	5913237476341764299813987
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 $\{0, 1, \dots, 90 \cdot 10^{25}\}$, 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.

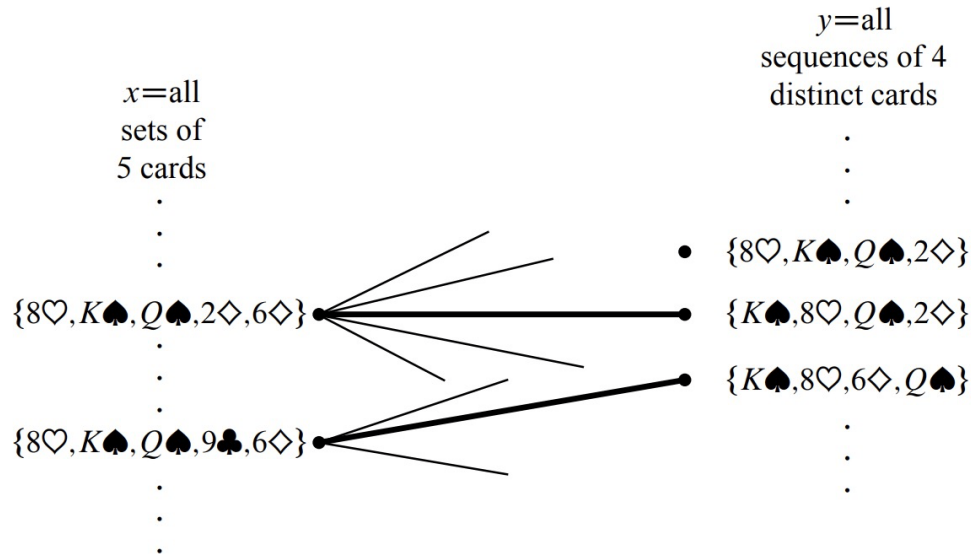


Figure 14.5 The bipartite graph where the nodes on the left correspond to sets of 5 cards and the nodes on the right correspond to *sequences* of 4 cards. There is an edge between a set and a sequence whenever all the cards in the sequence are contained in the set.

For example,

$$\{8♥, K♠, Q♠, 2♦, 6♦\} \quad (6.8.1)$$

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 $(8♥, K♠, Q♠, 2♦)$, $(K♠, 8♥, Q♠, 2♦)$, and $(K♠, 8♥, 6♦, Q♠)$.

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

$$\{8♥, K♠, Q♠, 9♣, 6♦\}, \quad (6.8.2)$$

then the Assistant reveals the corresponding sequence

$$(K♠, 8♥, 6♦, Q♠). \quad (6.8.3)$$

Using the matching, the Magician sees that the hand (6.8.2) is matched to the sequence (6.8.3), so he can name the one card in the corresponding set not already revealed, namely, the 9♣. 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 (6.8.1), it would be possible for the Assistant to reveal the same sequence (6.8.3), 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♣ or the 2♦.

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:

$10\heartsuit \ 9\diamondsuit \ 3\heartsuit \ Q\spadesuit \ J\diamondsuit$.

- 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 $10\heartsuit$.

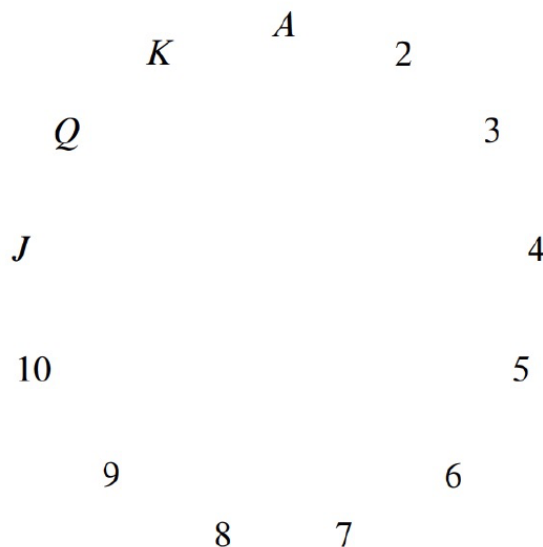


Figure 14.6 The 13 card ranks arranged in cyclic order.

- 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:

$A\clubsuit A\diamondsuit A\heartsuit A\spadesuit 2\clubsuit 2\diamondsuit 2\heartsuit 2\spadesuit \dots K\heartsuit K\spadesuit$

The order in which the last three cards are revealed communicates the number according to the following scheme:

$$\begin{aligned}
 (\text{small, medium, large}) &= 1 \\
 (\text{small, large, medium}) &= 2 \\
 (\text{medium, small, large}) &= 3 \\
 (\text{medium, large, small}) &= 4 \\
 (\text{large, small, medium}) &= 5 \\
 (\text{large, medium, small}) &= 6
 \end{aligned} \tag{6.8.4}$$

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!

- [14.8: The Pigeonhole Principle](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

6.9: 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-major⁵ 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:

$$\begin{aligned} |M| &= 60 \\ |E| &= 200 \\ |P| &= 40. \end{aligned} \tag{6.9.1}$$

The size of a union of three sets is given by a more complicated Inclusion-Exclusion formula:

$$\begin{aligned} |S_1 \cup S_2 \cup S_3| &= |S_1| + |S_2| + |S_3| \\ &\quad - |S_1 \cap S_2| - |S_1 \cap S_3| - |S_2 \cap S_3| \\ &\quad + |S_1 \cap S_2 \cap S_3| \end{aligned} \tag{6.9.2}$$

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:

$$\begin{aligned} 4 &\text{ math - EECS double majors} \\ 3 &\text{ math - physics double majors} \\ 11 &\text{ EECS - physics double majors} \\ 2 &\text{ triple majors} \end{aligned} \tag{6.9.3}$$

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:

$$\begin{aligned}
 |M \cup E \cup P| &= |M| + |E| + |P| - |M \cap E| - |M \cap P| - |E \cap P| \\
 &\quad + |M \cap E \cap P| \\
 &= 60 + 200 + 40 - 6 - 5 - 13 + 2 \\
 &= 278
 \end{aligned}
 \tag{6.9.4}$$

Sequences with 42, 04, or 60

In how many permutations of the set $\{0, 1, 2, \dots, 9\}$ 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 $\{0, 1, 2, \dots, 9\}$ 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).

$$\begin{aligned}
 |S_1 \cup S_2 \cup \dots \cup S_n| &= \\
 &\text{the sum of the sizes of the individual sets} \\
 \text{minus} &\text{ the sizes of all two-way intersections} \\
 \text{plus} &\text{ the sizes of all three-way intersections} \\
 \text{minus} &\text{ the sizes of all four-way intersections} \\
 \text{plus} &\text{ the sizes of all five-way intersections, etc.}
 \end{aligned}
 \tag{6.9.5}$$

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

$$\sum_{i \leq i < j \leq n} |S_i \cap S_j|.$$

Similarly, the sum of the sizes of the three-way intersections is

$$\sum_{i \leq i < j < k \leq n} |S_i \cap S_j \cap S_k|.$$

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).

$$\begin{aligned} \left| \bigcup_{i=1}^n S_i \right| &= \sum_{i=1}^n |S_i| \\ &\quad - \sum_{i \leq i < j \leq n} |S_i \cap S_j| \\ &\quad + \sum_{i \leq i < j < k \leq n} |S_i \cap S_j \cap S_k| + \cdots \\ &\quad + (-1)^{n-1} \left| \bigcap_{i=1}^n S_i \right|. \end{aligned} \tag{6.9.6}$$

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).

$$\left| \bigcup_{i=1}^n S_i \right| = \sum_{\emptyset \neq I \subseteq \{1, \dots, n\}} (-1)^{|I|+1} \left| \bigcap_{i \in I} S_i \right| \tag{6.9.7}$$

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

$$\phi(n) = n \prod_{i=1}^m \left(1 - \frac{1}{p_i} \right). \tag{6.9.8}$$

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 :

$$C_a ::= \{k^{\text{int}}[0..n) \mid a|k\}.$$

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,

$$S = \bigcup_{i=1}^m C_{p_i}. \quad (6.9.9)$$

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_i} 's, namely,

$$\left| \bigcap_{j \in I} C_{p_j} \right| = \frac{n}{\prod_{j \in I} p_j}. \quad (6.9.10)$$

for any nonempty set $I^{\text{int}}[1..m]$. This lets us calculate:

$$\begin{aligned} |S| &= \left| \bigcup_{i=1}^m C_{p_i} \right| && \text{(by 14.9.3)} \\ &= \sum_{\emptyset \neq I \subseteq [1..m]} (-1)^{|I|+1} \left| \bigcap_{i \in I} C_{p_i} \right| && \text{(by Inclusion-Exclusion (14.9.1))} \\ &= \sum_{\emptyset \neq I \subseteq [1..m]} (-1)^{|I|+1} \frac{n}{\prod_{j \in I} p_j} && \text{(by 14.9.4)} \\ &= -n \sum_{\emptyset \neq I \subseteq [1..m]} \frac{1}{\prod_{j \in I} (-p_j)} \\ &= -n \left(\prod_{i=1}^m \left(1 - \frac{1}{p_i} \right) \right) + n, \end{aligned} \quad (6.9.11)$$

so

$$\phi(n) = n - |S| = n \prod_{i=1}^m \left(1 - \frac{1}{p_i} \right),$$

which proves (6.9.8).

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.

6.10: 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*).

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \tag{6.10.1}$$

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 (6.10.1). 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,

$$\begin{aligned} A &::= \{(1, X) \mid X \subseteq [2, n] \text{ AND } |X| = k-1\} \\ B &::= \{(0, Y) \mid Y \subseteq [2, n] \text{ AND } |Y| = k\} \end{aligned} \quad (6.10.2)$$

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 $f: (A \cup B) \rightarrow S$ will prove the identity (6.10.1). 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 6.10.2

$$\sum_{r=0}^n \binom{n}{r} \binom{2n}{n-r} = \binom{3n}{n}$$

Proof available at the end of this section.

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.

Answers

Theorem 6.10.2

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. ■

This page titled [6.10: Combinatorial Proofs](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

- [14.10: Combinatorial Proofs](#) by Eric Lehman, F. Thomson Leighton, and Albert R. Meyer is licensed [CC BY-NC-SA 4.0](#).

7: GRAPH THEORY

7.1: Introduction

7.2: Path and Cycles

7.1: Introduction

The graph theory is one of the branches of mathematics. It was originated all the way back to 1736 when the first paper in graph theory was published. It didn't gain a widespread interest until 1920s. One of the main reasons it became intense interested is its applicability in many diverse fields, including computer science.

Let's have a taste of graph theory.

Example

Following is the highway system in Wyoming. An road inspector need to travel all the roads to file a report on the road conditions. Now, the question is "Is it possible to start in Greybull, travel each of the roads exactly once, and return to Greybull?" We use graph theory to find the answer to this question, so we can see how graph theory works.

preview_fr29

A graph is normally drawn with dots and lines. In graph theory, we call those dots vertices and call those lines edges. The first thing that we need to do is to create a graph model of the highway system shown above. In the graph model, we consider cities are vertices and roads between cities are edges, the graph we created shown below:

preview_2l1k

From the graph we created, we can tell: If the inspector wants to start in Greybull, travel each of the roads exactly once, and return to Greybull, then he/she can't do it, it is impossible. Why? How do we draw such conclusion? When we look at the graph, if there exists any vertex with an odd number of edges coming in or going out, such as vertex "Shoshoni" or "Worland", then it is impossible to travel each road in the map exactly once.

Why? Here is the logic:

Let's consider vertex "Shoshoni". There are three edges coming in or going out from this vertex. It is an odd number. Let's discuss, in this case, why it is impossible to travel each road in the map exactly once

The inspector comes in Shoshoni through one of three edges, he/she will need to leave Shoshoni from one of the other two edges to avoid traveling an edge twice. Regardless which edge he/she chooses to leave Shoshoni, he/she will not have a chance to travel the third edge unless he/she comes back to Shoshoni. But if the inspector comes back to Shoshoni, he/she has to get out because the inspector needs to go back to Greybull. However, the inspector can't get out Shoshoni unless he/she travels one the three edges twice.

Similar logic can be applied to other graphs with more vertices and more edges.

This example shows us how graph theory can be used to solve problems in this nature.

We won't become an expert of graph theory by taking one or two classes. We will begin the study of graph theory with an introduction of basic terms and terminologies.

In graph theory, a graph is considered as a set of vertices (nodes) V along with a set of edges (arcs) E . Each edge in E is represented by (v, w) where v and w are two vertices in V . If we use G to represent a graph, then we normally write $G=(V, E)$ which means G is graph defined by a set of vertices V and a set of edges E .

In above example, if we use G to represent the graph we created, then $G=(V,E)$, where $V=\{\text{Greybull, Sheridan, Buffalo, Gillette, Casper, Douglas, Muddy Gap, Lander, Shoshoni, Worland}\}$ and $E=\{e_1, e_2, \dots, e_{13}\}$, and those e_i ($i=1, \dots, 13$) are edges that connect vertices in the graph

Definition 1: Undirected Graph

Given a graph with a set of vertices V and a set of edges E . Each edge in the graph can be represented by (v, w) with v and w being elements of V . If (v, w) and (w, v) are considered as the same edge, then the graph is called an undirected graph. In this

case, we say (v, w) is the edge between v and w .

The graph we created in above example is an undirected graph. In this graph, cities are vertices and roads are edges between cities. Each edge is represented by (city 1 names, city 2 names), say (Greybull, Sheridan). Obviously, either (Greybull, Sheridan) or (Sheridan, Greybull) represents the same road between those two cities.

Definition 2: Directed Graph (Digraph)

Given a graph with a set of vertices V and a set of edges E . Each edge in the graph can be represented by (v, w) with v and w being elements of V . If (v, w) and (w, v) are considered as two different edges, then the graph is called a directed graph or a digraph. The (v, w) is called the edge from v to w and (w, v) is called the edge from w to v respectively.

Example

Following is a directed graph:

preview_2e7o

For a directed graph, each edge is directional which can either go into a vertex or go out of a vertex. In above sample directed graph, we can see edge e_1 and edge e_2 are the edges between v_2 and v_4 . They are two different edges. One of them, e_1 , is represented by (v_2, v_4) and the other one, e_2 , is represented by (v_4, v_2) . Here the order matters. Since e_1 and e_2 are associated with the same pair of vertices, we call such edges **parallel edges**.

In following graph, we can see some edges may go out and come into the same vertex, such edges are called **loops**. A graph contains no parallel edges nor loops is called a **simple graph**.

preview

Following terminologies may also be used in our discussion:

- If an edge e (directed or undirected) is represented by (v, w) , then we say e is an incident on v and w
- If an edge e (directed or undirected) is represented by (v, w) , then we say v and w are incident on e
- If an edge e (directed or undirected) is represented by (v, w) , then we say v and w are adjacent vertices

In our first example, edges in the graph are roads between cities. Sometimes, we may need to label the edges by the average travel times or distances of the roads. If a graph includes labels on the edges, then such graph is called weighted graph. The length of a path is defined by the sum of weights of the edges in the path for a weighted graph. For an unweighted graph, the length of a path is defined by the number of edges in the path.

Example

We label the edges in the graph we created in our first example by driving distance along the roads:

preview_3un8

The length of the path from Greybull, through Sheridan and Buffalo, to Gillette is the sum of the weights of the edges in the path, which is 150 mi. If we took the weights of edges out of the graph, then the graph becomes an unweighted graph. The path from Greybull, through Sheridan and Buffalo, to Gillette will be 3.

At the end of this lecture, we go over a few of definitions that we may use in the future discussion.

Definition 3

A graph is called a **complete graph** if there is an edge between every pair of distinct vertices.

Example

Following is a complete graph:

preview_om4p

Following is not a complete graph:

preview_qk79

Definition 4

Let V be a set of vertices and E be a set of edges, a graph G can be written as $G=(V, E)$. If there exist subsets V_1 and V_2 of V such that $V_1 \cap V_2 = \emptyset$ and $V_1 \cup V_2 = V$, which means V_1 and V_2 are not overlap and V_1 and V_2 consist of entire V , also, each edge in E connects one vertex in V_1 and one vertex in V_2 , then such graph is called a **bipartite** graph.

Example

Following graph is bipartite:

preview_o4e3

In this graph, $V_1 = \{v_1, v_2\}$ and $V_2 = \{v_3, v_4, v_5\}$. E consists of three edges. Each edge in E connects one vertex in V_1 and one vertex in V_2 .

Definition 5

The **complete bipartite graph** is a simple graph whose vertex set V is partitioned into subsets V_1 and V_2 in which the edge set E consists of **all** edges of the form (v_1, v_2) with $v_1 \in V_1$ and $v_2 \in V_2$

Example

Following is a complete bipartite graph:

preview_0bh2

7.2: Path and Cycles

In this section, we will continue to discuss graphs. We will begin this lecture by introducing some of the basic terms

Definition 1

A graph $G = (V, E)$ is **connected** if given any vertices v and w in V , there is a path from v to w .

We will use two examples to explain what kind of graph is considered as connected and what kind of graph is not.

Example

Following graph is connected:

preview

Following is an unconnected graph:

preview_0y3t

It is an unconnected graph because there is no path from vertex v_1 to vertex v_2 .

Definition 2

Let $G = (V, E)$ be a graph. We call $G' = (V', E')$ a **subgraph** of G if

- $V' \subseteq V$ and $E' \subseteq E$
- For every edge $e \in E'$, if e is incident on v' and w' , then $v', w' \in V'$

We can write $G' \subseteq G$.

Example

Given a graph as following:

preview_o9db

Here $G = (V, E)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{e_1, e_2, e_3, e_4, e_5\}$

Now, let $V' = \{v_1, v_2, v_3\}$ and $E' = \{e_1, e_4, e_5\}$, then $G' = (V', E')$ is subgraph of G . It is because

- V' is subset of V and E' is also a subset of E
- For any edge e' in E' , the two vertices connected by e' are also in V'

However, if we let $V' = \{v_1, v_2, v_3\}$ and $E' = \{e_1, e_2, e_5\}$, then $G' = (V', E')$ is not a subgraph of G because one of vertices connected by edge e_2 which is v_4 is not in V'

Definition 3

Let G be a graph and let v be a vertex in G . The subgraph G' of G containing of all the edges and vertices in G that contained in some path beginning at v is called the **component** of G containing v .

Example

Give a graph $G = (V, E)$ as following:

preview_jwaj

In this graph,

- The component of G containing v_3 is the subgraph $G' = (V_1, E_1)$ with $V_1 = \{v_1, v_2, v_3\}$ and $E_1 = \{e_1, e_2, e_3\}$
- The component of G containing v_4 is the subgraph $G' = (V_2, E_2)$ with $V_2 = \{v_4\}$ and $E_2 = \phi$
- The component of G containing v_5 is the subgraph $G' = (V_3, E_3)$ with $V_3 = \{v_5, v_6, v_7\}$ and $E_3 = \{e_4, e_5\}$

Definition 4

Let v and w be vertices in a graph G .

- A **simple path** from v to w is a path from v to w with no repeated vertices. No repeated vertices means a path that does not pass a vertex more than once.

Example

preview_ks5i

In this graph, the path $(v_2, e_1, v_1, e_4, v_3, e_5, v_2, e_2, v_4)$ is not a simple path because it passes v_2 twice. The path $(v_1, e_4, v_3, e_5, v_2)$ is a simple path.

- A **cycle** is a path of nonzero length from v to v with no repeated edges. No repeated edges means the path does not go through an edge more than once.

Example

In above graph, the path $(v_1, e_1, v_2, e_5, v_3, e_4, v_2)$ is a cycle

- A **simple cycle** is a cycle from v to v in which, except for the beginning and ending vertices that are both equal to v , there is no repeated vertices.

Example

In above graph, the path $(v_1, e_1, v_2, e_5, v_3, e_4, v_2)$ is a simple

Definition 5

A cycle in a graph G that includes all of the edges, and all of the vertices of G is called a **Euler cycle**.

The question if a graph has a Euler cycle is the same question as if there is a path in a graph that goes through all the vertices exactly once. Recall the example we discussed in the previous lecture, [Lecture 3.5](#), in which we consider the question that if it is possible for the road inspector to begin from city Greybull and travel all the road exactly once and come back to Greybull. This question is basically asking if there is a Euler cycle in the graph, we used for modeling the highway system in that example.

We have discussed the question and drew a conclusion that it is not possible for the road inspector to do so. Such conclusion is also supported by following two theorems:

Theorem 1

If a graph G has a Euler cycle, the G is connected, and every vertex has an even degree. The degree of a vertex is defined by the number of edges coming into or going out of the vertex. When we say a vertex has an even degree, we mean the number of edges coming into or going out the vertex is an even number.

Equivalently, if not all the vertices have an even degree, then the graph will not have a Euler cycle.

Theorem 2

If G is a connected graph and every vertex has an even degree, then G has a Euler cycle, which means it is possible to begin from a vertex and come back to the same vertex by passing all the edges exactly once (no repeated edges)

Example

In following graph, is it possible to begin from v_1 and come back to v_1 by passing all the edges exactly once?

preview_ygds

Based on Theorem 2, the answer is yes.

Example

In following graph, is it possible to begin from v_1 and come back to v_1 by passing all the edges exactly once?

preview_q8nw

Based on Theorem 1, the answer is no.

Theorem 3

If G is a graph with m edges and n vertices, then the sum of degrees of all the vertices is equal to $2m$, which is always an even number.

Example

Given a graph as following:

preview_f00c

- The degree of v_1 is 2
- The degree of v_2 is 3
- The degree of v_3 is 3
- The degree of v_4 is 2

The sum of the degrees of all the vertices is 10

The number of the edges is 5

Thus, we can see, in this graph, the sum of the degrees of all the vertices is equal to 2 times the number of the edges

The significance of this theorem is to allow us to quickly calculate the number of sums of the degrees of all the vertices by just counting the number of edges in a graph.

Theorem 4

A graph has a path with no repeated edges from v to w ($v \neq w$) containing all the edges and vertices if and only if it is connected and v and w are the only vertices having odd degree.

Example

Given a graph as following:

preview_nhb6

Based on the Theorem 4, we can determine that there is a path from v_3 to v_2 that passes all the edges exactly once (no repeated edges) because v_3 and v_2 are the only two vertices that have odd number of degrees. Such path is ($v_3, e_3, v_4, e_2, v_2, e_5, v_3, e_4, v_1, e_1, v_2$)

Please notice, such path is not unique, there are other paths that can also passes all the edges exactly once. Also, such paths allow repeated vertices.

Following is the last theorem that we would like to discuss in this lecture:

Theorem 5

If a graph G contains a cycle from v to v , then G must also contain a simple cycle from v to v .

A cycle may allow repeated vertices, but a simple cycle won't. This theorem basically says if there is a cycle from v to v which may contains repeated vertices, we can always find a simple cycle from v to v without repeated vertices except the vertex v .

Example

In following graph:

preview_2ddb

Based on Theorem 2, we know, in this graph, we have a Euler cycle. But a Euler cycle is not necessarily a simple cycle. In this graph, the Euler cycle is $(v_1, v_2, v_3, v_4, v_5, v_2, v_4, v_6, v_5, v_3, v_1)$, here we omit the edge between two vertices for the sake of simplicity. Obviously, it is not a simple cycle because it contains repeated vertices such as v_2, v_3 and v_5 . Based on Theorem 5, we must have a simple cycle from v_1 to v_1 . Actually, we have more than one simple cycle from v_1 to v_1 . For instance, (v_1, v_2, v_3, v_1) is a simple cycle from v_1 to v_1 .

8: TREES

8.1: Introduction and Huffman Codes

8.2: Terms and Characterizations of Trees

8.3: Binary Trees and Binary Search Trees

8.1: Introduction and Huffman Codes

Trees form one of the most widely used subclasses of graphs. In other words, A tree is a specific type of graphs. Such graphs are widely used in computer science for organizing and relating data in a database. In this lecture, we will focus our discussions on trees and their applications.

Before we give the definition of a tree, we first review following terms we have used in previous lecture:

- A **vertex** in a graph is a dot and an **edge** is a line between two vertices.
- A graph is a set of vertices and a set of edges.
- If there are more than one distinctive edge between two vertices, then those edges are called **parallel edges**.
- If an edge comes in and goes out the same vertex, then such edge is called a **loop**.
- If a graph contains no parallel edges and no loops, then it is called a **simple graph**.
- If we start at a vertex v_0 , travel along then edge to vertex v_1 , travel along another edge to vertex v_2 , and so on, an eventually arrive at vertex v_n , we call the complete tour a **path** from v_0 to v_n
- A **simple path** is a path that contains no repeated vertices.

Definition 1

A **tree** T is a simple graph satisfying the following: If v and w are vertices in T , there is a unique simple path from v to w . A rooted tree is a tree in which a particular vertex is designated as the root. A vertex in a tree is also called a **node**.

Example

Following graph is a tree:

preview_3tx1

If we designate vertex v_2 as the root, then the tree can be drawn as following:

preview

Notice the tree is drawn with the root placed on the top, so it is actually an "upside down" tree.

The **level** of a vertex is defined by the length of the path from the root to the vertex. For example, the level of the root is 0 and the levels of vertices v_3 and v_6 are 1 and the levels of vertices v_1 , v_4 , and v_5 are 2.

The **height** of a tree is defined by the maximum level number that occurs. The height of the tree above is 2.

A tree is often used to represent a collection of objects that are hierarchically related such as officers of the administrative system of a school or family members of a family.

Following graph is not a tree:

preview_0u96

because we can reach v_5 from v_1 through two different paths: (v_1, v_4, v_5) and (v_1, v_2, v_5)

Trees are widely used in computer science. Here is one of the well-known applications of tree used in computer coding: **Huffman Codes**.

Example

Suppose we want to use 0s and 1s to code following string of characters: BCAADDDCCACACAC, the following article well explains how Huffman Codes work:

<https://www.programiz.com/dsa/huffman-coding#working>

Please study above article carefully. Now, we will work on another coding problem with the method discussed in the article.

Question

Assume a string of characters contains characters A, B, C, D, E and the frequencies of each character in the string are summarized as following:

Character Frequency

=====

A 2

B 3

C 7

D 8

E 12

Step 1: Create a tree by using the first two numbers in the frequency list as following:

preview_6vv2

and update the frequency list as following:

2, 3, 7, 8, 12 -> 2+3, 7, 8, 12 -> 5, 7, 8, 12

Step 2: Repeat step 1 with the updated frequency list, which is using the first two numbers in the list to create a tree as following:

preview_rmgi

and update the frequency list as following:

5, 7, 8, 12 -> 5+7, 8, 12 -> 12, 8, 12 -> 8, 12, 12

Step 3: Repeat step 1 with the updated frequency list, which is using the first two numbers in the list to create a tree as following:

preview_kjin

and update the frequency list as following:

8, 12, 12 -> 8+12, 12 -> 20, 12 -> 12, 20

Step 4: Use the two numbers in the updated frequency list to create a tree as following:

preview_coqh

Step 5: Construct a tree with the trees created in each step above beginning with the tree created in step 4: replacing 12 by the tree created in step 2 and replace 20 by the tree created in step 3, and then replace 5 by the tree created in step 1

preview_45qw

Step 6: Remove the number 32, 12, 20, and 5 from the tree (not the vertices containing those numbers)

preview_jrzw

Step 7: Replace the frequencies by the characters accordingly:

preview_woq1

Final Step: At each node, put 0 on the left edge and put 1 on the right edge as following:

preview_wcxi

This is the tree we use for encoding and decoding the characters in a string with Huffman Codes.

To encode the character in the string, we traverse the tree from the root to the character and record the 0s or 1s as the Huffman code. For example, the letter A in the string will be coded as 0000 and letter D will be coded as 10 and so on. When we send a string to a client machine, we send the Huffman codes the characters in the string along with the tree. Then, the receiver will use the tree to decode the string. For example, the code 001000010 will be decoded as BAD. How the decoding works? Start from the left side of the code and begin with the root of the tree, if it is 0 in the code, go left on the tree. If it is 1, go right on the tree until we reach a specific character in the tree. That is why the first character is decoded as B because we go left, left, and then right to reach the vertex that contains B. Then go back to the root of the tree and continue for the rest of the 0s and 1s in the code.

8.2: Terms and Characterizations of Trees

In this section, we will continue the discussion on trees. First, we will use following tree to introduce some of the terms used in the discussion

preview_iauc

- Top-down traverse the tree, if there is an edge from a vertex v to vertex w , the w is called a **child** of v . For example, vertex B and C are the children of A. D and E are the children of B and so on.
- If w is a child of v , the v is called the **parent** of w . For example, C is the parent of F and G
- If w and v have the same parent, then w and v are **siblings**.
- If a vertex v has no children, then v is called a **terminal vertex** or a **leaf**. For example, H, I, E, F, G are terminal vertices.
- In the sample tree above, the vertices B, C, D, E, F, G, H, I are called the **descendent** of the vertex A and A is called an **ancestor** of B, C, D, E, F, G, H, I. Similarly, B is an ancestor of D, E, H, I, and D, E, H, I are the descendent of B
- If a vertex is not a terminal vertex, then it is called an **internal or branch vertex**. For example, vertex B, C, D are internal vertices.
- If v is a vertex, then the tree formed by v together with all its descendent is called a **subtree**.
- A tree may only have a root without any edges and other vertex.
- A tree that does not have any vertex is still a tree, it is called an **empty tree**.

Let G be a graph and v be a vertex of the graph. A cycle from v to v is a path that begin at v and come back to v containing no repeated edges. If a graph contains no cycles, then it is called an acyclic graph. We have following theorem which is very useful in our discussion:

Theorem 1

A graph is a tree if and only if the graph is connected and acyclic.

This theorem can help us to quickly identify if a graph is a tree.

There are many different types of trees.

Definition 1

A tree T is a **spanning tree** of a graph G if T is a subgraph of G that contains all of the vertices of G

If a graph has a spanning tree, then it means we can start from a vertex and reach each of other vertices with a unique simple path.

How do we supposed to know if a graph contains a spanning tree? Following theorem will help:

Theorem 2

A graph G has a spanning tree if and only if G is connected.

A graph is connected if there is a path between any two vertices in the graph. The theorem 2 tells us as long as the graph is a connected single piece, then it will have a spanning tree.

Once we have determined a graph has a spanning tree, the next question is how to find such spanning tree. In other words, how to find a subgraph that starts at a vertex and reach each other vertices in the graph with a unique simple path.

There are many different ways of finding a spanning tree in a connected graph. Here we discuss so called "Breadth-First Search for a Spanning Tree".

Given a graph as following:

preview

This is a connected graph. Now we will walk through the process of finding a spanning tree.

Step 1: Select an ordering of the vertices, say a, b, c, d, e and select the first vertex a and label it as the root the spanning tree T

Step 2: Add all the vertices that can be reached with a single edge without producing a cycle. In this step, we have T built up as following:

preview_0hwm

Step 2: Add all the vertices that can be reached from d by a single edge without producing a cycle to the tree T as the children of b. At this step the T is built as following:

preview_uodx

If the tree T has already contained all the vertices in the graph, then it is the spanning tree we are looking for and stop. Otherwise, continue the process from vertex b and so on until the tree contains all the vertices in the graph.

Above process of finding a spanning tree from a connected graph can be written as an algorithm and coded with C++ or other programming languages.

8.3: Binary Trees and Binary Search Trees

Binary and Binary Search Trees

Trees we discuss in Graph Theory are often used in Computer Science for solving many programming problems. Binary trees as a subdivision of general trees are often used in creating a structure for data storage.

A tree is a directed graph which means the edges in a tree all have directions from one vertex to another vertex. If an edge between vertices m and n is pointed from m to n , then we can call m is the **parent** of n and n is one of **child** of m .

Definition

A Binary tree is a rooted tree in which every vertex has at most two children.

Example

The trees we created for Huffman Codes are binary trees:

preview_wtk3

Definition 1

A **full binary tree** is a binary tree in which each vertex has either two children or zero children.

The tree in the previous example is a full binary tree.

One of important facts about a full binary tree is

Theorem 1

If a full binary tree T has n internal vertices, then T has $n+1$ terminal vertices and $2n+1$ total vertices.

In the previous example, we have 4 internal vertices (the vertices that have at least one child), then we can see the tree has 5 terminal vertices (the one that don't have any children)

It won't be sufficient to convince others an assertion stated in a theorem by just presenting a few of examples. We will need a formal proof by using basic facts and logic.

Here we present a formal proof of above theorem to let us see the logic and the way we use in proving a theorem in graph theory.

Proof

Since the T has n internal vertices and T is a full binary tree, T will have $2n$ vertices that are children. Since there is only one vertex in T that is not a child of any other vertices, which is the root, the total number of vertices in T is $2n+1$. Now, subtract the number of internal vertices from the total number of vertices, $(2n+1)-n$, so T has $n+1$ terminal vertices.

From this point on, we will refer a vertex in a binary tree as a "node" and put the children below their parent with one on the left and one on the right. Refer the one on the left as "left child" and the one on the right as "right child"

preview_4cy6

Definition 2

A **binary search tree (BST)** is a binary tree T in which data are associated with the nodes. The data are arranged so that, for each node v in T , each data item in the left subtree of v is less than the data item in v , and each data item in the right subtree of v is greater than the data item in v .

Example

preview

In this example, the graph is binary tree with set of integers associated with each node. We can see that all the number on the left subtree of a node v are all less than the number in v and all the numbers in the right subtree are all greater than the number in v for any node v in the tree, thus this is a binary search tree.

Example

Construct a binary search tree for storing following sequence of integers: 32, 10, 38, 15, 19, 8, 9, 2, 11, 42, 35

We create such binary search tree in following process:

Step 1: create a node and place the first number, 32, in the sequence into the node and use this node as the root of BST

Step 2: Set the next number in the sequence as a child of 32. In this case, it is the left child because 10 is less than 32 and all the numbers that are less than 32 go to left subtree of 32 based on the BST definition.

Step 3: Take the third number, 38, in the sequence and begin at the number in the root, 32. If the number in the sequence is less than 32, go to the left, otherwise, go to the right. If we reach a node, compare the number in the sequence with the number in the node, then either go to the left or go to the right depending on whether the number in the sequence is less than or greater than the number in the node. Keep on going until no edge to follow. In this case, 38 is greater than 32, so we go to right, but there is no edge on this direction which means 32 has not right child. Now, we set 38 as right child of 32

preview_6igl

Step 3: Take the next number in the sequence, 15 and repeat Step 2: begin with 32, go the left. Reach 10, compare and then go to the right. No more edges, set 15 as the right child of 10:

preview_lz9p

Step 4: Take the next number in the sequence, 19 and repeat step 2.

preview_94pf

continue until we place all the numbers in the sequence. In this case, the binary search tree is shown as following:

preview_yaa8

Tree Traversals

Traversing a tree normally means visiting the objects that each node represents in that tree

There are many different ways of traversing a tree

1. Breadth-first Search

Visit all the nodes in the same level before visit the nodes in the next level beginning at level 0 (root)

Using such method to visit the tree in above example, we are getting a sequence of numbers as following: 32, 10, 38, 8, 15, 35, 42, 2, 9, 11, 19

2. Depth-first Search

Set the items in a tree in a certain order $v_1, v_2, \dots, v_n$. Visit the root first, and then visit the closest node with the smallest position number. From the second visited node, do the same thing: visit the closest node with the smallest position number. Repeat this process until reach a terminal node. After a terminal node is visited, visit its siblings. When all the members in a family (children and parent) are visited, visit the siblings of the parent of this family.

Here, we demonstrate how to use Depth-first Search method to traverse the binary search tree we created in above example. In this binary search tree, we have 11 nodes which represent a set of 11 integers. The Depth-first Search depends on the way we arrange those integers. Here we simply arrange them in an ascending order: 2, 8, 9, 10, 11, 15, 19, 32, 35, 38, 42 and use a position value i with $i = 1, 2, 3, \dots, 11$ to represent the positions of each integer in the sequence. For example, the position

value of the integer 2 is 1 and the position value of 19 is 7. And we also name each node in this binary search tree with the number it represents. For example, we will name the root as node 32 and call its left child node 10 and so on.

Now, let's get started.

step 1: visit the root node 32, then from 32 visit next closest node and choose the one with the smallest position value. In this case, it is node 10

step 2: from node 10, visit closest node with the smallest position again. Continue this process until we reach a terminal node, in this case, it is node 2

step 3: After visiting this terminal node, visit its siblings if it has siblings.

step 4: After everyone in this family is visited, visit the siblings of the parent. In this case, it is node 15.

step 5: Now, from node 15, repeat the Depth-first Search.

step 6: After all the members of node 15's family are visited, all the members in node 10's family are also visited, now visit the siblings of node 10, which is node 38

With Depth-first Search from each node, we will eventually visit all the node in the tree and get a sequence of numbers as following: 32, 10, 8, 2, 9, 15, 11, 19, 38, 35, 42

It may seem very complicated and very confusing, actually, it is not. It is just difficult to explain it with the text. Let's watch the following video demo:

[preview_f55h](#)

3. Preorder Visit

This method visits the parents before visiting their children. Children are visited right after the parent is visited and children are visited from left to right

If we preorder visit the binary search tree we created in above example, we will get a sequence as following:

32, 10, 8, 2, 9, 15, 11, 19, 38, 35, 42

For this binary search tree, Preorder Visit and Depth-First Search produce the same result. But, in general, they won't.

4. Inorder Visit

This method visits a binary tree as following: visit a node after all the nodes in its left subtree are visited and before any node in its right subtree is visited.

If we inorder visit the binary search tree we created in above example, we will get following sequence:

2, 8, 9, 10, 11, 15, 19, 32, 35, 38, 42

5. Postorder Visit

This method visits a node after all the descendent of this node are visited. Siblings are visited from left to right.

If postorder visit the binary search tree we created in above example, we will get following sequence:

2, 9, 8, 11, 19, 15, 10, 35, 42, 38, 32

We can write an algorithm for creating a binary search tree or any type of binary tree. We can also write algorithms for those traverse methods and code them with different computer languages. We can use tree structures in many programming applications and more.

Studying computer science is a long journey, it will never end.

INDEX

B

binary relations

[3.2: Binary Relations](#)

C

cardinality

[3.4: Cardinality and Power Set](#)

E

equivalence

[1.9: Equivalence and Validity](#)

F

functions

[3.1: Functions](#)

I

induction

[4.1: Introduction to Mathematical Induction](#)

P

Pascal's triangle identity

[6.10: Combinatorial Proofs](#)

S

set

[2.1: Sets and Set Notations](#)

T

truth tables

[1.8: Propositional Logic in Computer Programs](#)

V

validity

[1.9: Equivalence and Validity](#)

GLOSSARY

S

Sample Word 1 | Sample Definition 1

This page titled [Glossary](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by [Jin He](#).

DETAILED LICENSING

Overview

Title: Discrete Mathematics for Computer Science (Jin He)

Webpages: 57

Applicable Restrictions: Noncommercial

All licenses found:

- [CC BY-NC-SA 4.0](#): 64.9% (37 pages)
- [Undeclared](#): 35.1% (20 pages)

By Page

- Discrete Mathematics for Computer Science (Jin He) - [CC BY-NC-SA 4.0](#)
 - Front Matter - [CC BY-NC-SA 4.0](#)
 - Discrete Mathematics for Computer Science - [CC BY-NC-SA 4.0](#)
 - InfoPage - [CC BY-NC-SA 4.0](#)
 - Table of Contents - [Undeclared](#)
 - Licensing - [CC BY-NC-SA 4.0](#)
 - 1: Introduction to Propositional Logic - [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: Proving an Implication - [CC BY-NC-SA 4.0](#)
 - 1.4: Proving an "If and Only If" - [CC BY-NC-SA 4.0](#)
 - 1.5: Proof by Cases - [CC BY-NC-SA 4.0](#)
 - 1.6: Proof by Contradiction - [CC BY-NC-SA 4.0](#)
 - 1.7: Propositions from Propositions - [CC BY-NC-SA 4.0](#)
 - 1.8: Propositional Logic in Computer Programs - [CC BY-NC-SA 4.0](#)
 - 1.9: Equivalence and Validity - [CC BY-NC-SA 4.0](#)
 - 1.10: The Algebra of Propositions - [CC BY-NC-SA 4.0](#)
 - 1.11: Predicate Formulas - [CC BY-NC-SA 4.0](#)
 - 2: Sets and Set Operations - [Undeclared](#)
 - 2.1: Sets and Set Notations - [CC BY-NC-SA 4.0](#)
 - 2.2: Set Operations - [Undeclared](#)
 - 2.3: Set Difference and Set Complement - [Undeclared](#)
 - 3: Functions and Binary Relations - [Undeclared](#)
 - 3.1: Functions - [CC BY-NC-SA 4.0](#)
 - 3.2: Binary Relations - [CC BY-NC-SA 4.0](#)
 - 3.3: Reverse Relations and Images - [Undeclared](#)
 - 3.4: Cardinality and Power Set - [CC BY-NC-SA 4.0](#)
 - 4: Mathematical Induction - [Undeclared](#)
 - 4.1: Introduction to Mathematical Induction - [CC BY-NC-SA 4.0](#)
 - 4.2: Proof by Using Mathematical Induction - [Undeclared](#)
 - 4.3: Loop Invariant - [Undeclared](#)
 - 5: Algorithms - [Undeclared](#)
 - 5.1: Introduction to Algorithm - [Undeclared](#)
 - 5.2: More Examples of Algorithms - [Undeclared](#)
 - 6: Cardinality Rules - [CC BY-NC-SA 4.0](#)
 - 6.1: Counting One Thing by Counting Another - [CC BY-NC-SA 4.0](#)
 - 6.2: Counting Sequences - [CC BY-NC-SA 4.0](#)
 - 6.3: The Generalized Product Rule - [CC BY-NC-SA 4.0](#)
 - 6.4: The Division Rule - [CC BY-NC-SA 4.0](#)
 - 6.5: Counting Subsets - [CC BY-NC-SA 4.0](#)
 - 6.6: Sequences with Repetitions - [CC BY-NC-SA 4.0](#)
 - 6.7: Counting Practice - Poker Hands - [CC BY-NC-SA 4.0](#)
 - 6.8: The Pigeonhole Principle - [CC BY-NC-SA 4.0](#)
 - 6.9: Inclusion-Exclusion - [CC BY-NC-SA 4.0](#)
 - 6.10: Combinatorial Proofs - [CC BY-NC-SA 4.0](#)
 - 7: Graph Theory - [Undeclared](#)
 - 7.1: Introduction - [Undeclared](#)
 - 7.2: Path and Cycles - [Undeclared](#)
 - 8: Trees - [Undeclared](#)
 - 8.1: Introduction and Huffman Codes - [Undeclared](#)
 - 8.2: Terms and Characterizations of Trees - [Undeclared](#)
 - 8.3: Binary Trees and Binary Search Trees - [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 - [CC BY-NC-SA 4.0](#)

- Detailed Licensing - *Undeclared*

This page titled [Discrete Mathematics for Computer Science \(Jin He\)](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by Jin He.

DETAILED LICENSING

Overview

Title: Discrete Mathematics for Computer Science (Jin He)

Webpages: 57

Applicable Restrictions: Noncommercial

All licenses found:

- [CC BY-NC-SA 4.0](#): 64.9% (37 pages)
- [Undeclared](#): 35.1% (20 pages)

By Page

- Discrete Mathematics for Computer Science (Jin He) - [CC BY-NC-SA 4.0](#)
 - Front Matter - [CC BY-NC-SA 4.0](#)
 - Discrete Mathematics for Computer Science - [CC BY-NC-SA 4.0](#)
 - InfoPage - [CC BY-NC-SA 4.0](#)
 - Table of Contents - [Undeclared](#)
 - Licensing - [CC BY-NC-SA 4.0](#)
 - 1: Introduction to Propositional Logic - [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: Proving an Implication - [CC BY-NC-SA 4.0](#)
 - 1.4: Proving an "If and Only If" - [CC BY-NC-SA 4.0](#)
 - 1.5: Proof by Cases - [CC BY-NC-SA 4.0](#)
 - 1.6: Proof by Contradiction - [CC BY-NC-SA 4.0](#)
 - 1.7: Propositions from Propositions - [CC BY-NC-SA 4.0](#)
 - 1.8: Propositional Logic in Computer Programs - [CC BY-NC-SA 4.0](#)
 - 1.9: Equivalence and Validity - [CC BY-NC-SA 4.0](#)
 - 1.10: The Algebra of Propositions - [CC BY-NC-SA 4.0](#)
 - 1.11: Predicate Formulas - [CC BY-NC-SA 4.0](#)
 - 2: Sets and Set Operations - [Undeclared](#)
 - 2.1: Sets and Set Notations - [CC BY-NC-SA 4.0](#)
 - 2.2: Set Operations - [Undeclared](#)
 - 2.3: Set Difference and Set Complement - [Undeclared](#)
 - 3: Functions and Binary Relations - [Undeclared](#)
 - 3.1: Functions - [CC BY-NC-SA 4.0](#)
 - 3.2: Binary Relations - [CC BY-NC-SA 4.0](#)
 - 3.3: Reverse Relations and Images - [Undeclared](#)
 - 3.4: Cardinality and Power Set - [CC BY-NC-SA 4.0](#)
 - 4: Mathematical Induction - [Undeclared](#)
 - 4.1: Introduction to Mathematical Induction - [CC BY-NC-SA 4.0](#)
 - 4.2: Proof by Using Mathematical Induction - [Undeclared](#)
 - 4.3: Loop Invariant - [Undeclared](#)
 - 5: Algorithms - [Undeclared](#)
 - 5.1: Introduction to Algorithm - [Undeclared](#)
 - 5.2: More Examples of Algorithms - [Undeclared](#)
 - 6: Cardinality Rules - [CC BY-NC-SA 4.0](#)
 - 6.1: Counting One Thing by Counting Another - [CC BY-NC-SA 4.0](#)
 - 6.2: Counting Sequences - [CC BY-NC-SA 4.0](#)
 - 6.3: The Generalized Product Rule - [CC BY-NC-SA 4.0](#)
 - 6.4: The Division Rule - [CC BY-NC-SA 4.0](#)
 - 6.5: Counting Subsets - [CC BY-NC-SA 4.0](#)
 - 6.6: Sequences with Repetitions - [CC BY-NC-SA 4.0](#)
 - 6.7: Counting Practice - Poker Hands - [CC BY-NC-SA 4.0](#)
 - 6.8: The Pigeonhole Principle - [CC BY-NC-SA 4.0](#)
 - 6.9: Inclusion-Exclusion - [CC BY-NC-SA 4.0](#)
 - 6.10: Combinatorial Proofs - [CC BY-NC-SA 4.0](#)
 - 7: Graph Theory - [Undeclared](#)
 - 7.1: Introduction - [Undeclared](#)
 - 7.2: Path and Cycles - [Undeclared](#)
 - 8: Trees - [Undeclared](#)
 - 8.1: Introduction and Huffman Codes - [Undeclared](#)
 - 8.2: Terms and Characterizations of Trees - [Undeclared](#)
 - 8.3: Binary Trees and Binary Search Trees - [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 - [CC BY-NC-SA 4.0](#)

- Detailed Licensing - *Undeclared*

This page titled [Discrete Mathematics for Computer Science \(Jin He\)](#) is shared under a [CC BY-NC-SA 4.0](#) license and was authored, remixed, and/or curated by Jin He.