

Notes for Discrete Mathematics

These are notes I wrote up for my Discrete Math classes.

If you see anything wrong, please send me a note at heinold@msmary.edu. Last updated March 9, 2016.

Contents

1	Logic	4
1.1	Logical operators and truth tables	4
1.2	Some logical rules	5
1.3	Universal quantifiers	7
1.4	Conditionals	8
1.5	Some applications of logic	11
2	Sets	13
2.1	Common set notation and operations	13
2.2	Set identities	14
2.3	Cartesian product and power set	15
2.4	Notation	16
3	Functions and Relations	18
3.1	Well-defined functions	19
3.2	One-to-one functions	19
3.3	Onto functions	21
3.4	Bijections	21
3.5	Inverses	22
3.6	Preimages	22
3.7	A combined example	23
3.8	Relations	24
3.9	Logarithms	26
3.10	Some places where logarithms show up	27
3.11	Floors and ceilings	32
3.12	Modular arithmetic	33
4	Recursion and Induction	37
4.1	Recursion	37
4.2	Induction	42
5	Counting	46
5.1	The multiplication rule	46
5.2	Rearranging things	48
5.3	Subsets	49

5.4	Addition rule	51
5.5	Negation rule	52
5.6	Summary	53
6	Probability	55
6.1	The basic rule	55
6.2	The multiplication rule	56
6.3	Other rules	58
6.4	Binomial probabilities	59
6.5	Expected value	60
6.6	Bayes' theorem	61
7	Graphs	63
7.1	Terminology	64
7.2	Graph coloring	65
7.3	Eulerian tours	66
7.4	Hamiltonian cycles	67
7.5	Weighted graphs and minimum spanning trees	69

Chapter 1

Logic

1.1 Logical operators and truth tables

Let's start with a few common symbols:

Symbol	Common name
\wedge	AND
\vee	OR
\oplus	XOR
\sim	NOT

The expression $p \wedge q$ is true only when both p and q are true. The expression $p \vee q$ is true whenever one, the other, or both of p and q are true. In math the word “or” always means “one, the other, or both.” It is sometimes called the *inclusive or*. On the other hand, the *exclusive or*, $p \oplus q$, is true when one or the other, *but not both*, of p and q are true. Finally, the expression $\sim p$ is the opposite of p , true if p is false, and false if p is true.¹

We can use tables, called *truth tables*, to display the rules for these operations:

$\frac{p}{T}$	$\frac{q}{T}$	$\frac{p \wedge q}{T}$	$\frac{p}{T}$	$\frac{q}{T}$	$\frac{p \vee q}{T}$	$\frac{p}{T}$	$\frac{\sim p}{F}$	$\frac{p}{T}$	$\frac{q}{T}$	$\frac{p \oplus q}{F}$
$\frac{p}{T}$	$\frac{q}{F}$	$\frac{p \wedge q}{F}$	$\frac{p}{T}$	$\frac{q}{F}$	$\frac{p \vee q}{T}$	$\frac{p}{F}$	$\frac{\sim p}{T}$	$\frac{p}{T}$	$\frac{q}{F}$	$\frac{p \oplus q}{T}$
$\frac{p}{F}$	$\frac{q}{T}$	$\frac{p \wedge q}{F}$	$\frac{p}{F}$	$\frac{q}{T}$	$\frac{p \vee q}{T}$	$\frac{p}{F}$	$\frac{\sim p}{T}$	$\frac{p}{F}$	$\frac{q}{T}$	$\frac{p \oplus q}{T}$
$\frac{p}{F}$	$\frac{q}{F}$	$\frac{p \wedge q}{F}$	$\frac{p}{F}$	$\frac{q}{F}$	$\frac{p \vee q}{F}$			$\frac{p}{F}$	$\frac{q}{F}$	$\frac{p \oplus q}{F}$

Truth tables are sometimes useful for understanding the effects of complicated conditions. Here are a couple of examples.

1. Write the truth table for $\sim(\sim p \vee \sim q)$.

p	q	$\sim(\sim p \vee \sim q)$
T	T	$\sim(\sim T \vee \sim T) = \sim(F \vee F) = T$
T	F	$\sim(\sim T \vee \sim F) = \sim(F \vee T) = F$
F	T	$\sim(\sim F \vee \sim T) = \sim(T \vee F) = F$
F	F	$\sim(\sim F \vee \sim F) = \sim(T \vee T) = F$

From this, we see that $\sim(\sim p \vee \sim q)$ is the same as $p \wedge q$.

2. If we want to see if two logical expressions are equivalent, we can compare their truth tables. For instance, to see if $\sim(p \vee q)$ and $\sim p \vee \sim q$ are equivalent, we can make a truth table like below:

¹AND and OR are sometimes given the names *conjunction* and *disjunction*. You will also sometimes see $\neg p$ or \bar{p} for the NOT operation.

p	q	$\sim(p \vee q)$	$\sim p \vee \sim q$
T	T	$\sim(T \vee T) = F$	$\sim T \vee \sim T = F$
T	F	$\sim(T \vee F) = F$	$\sim T \vee \sim F = T$
F	T		
F	F		

We see that they do not produce the same results in the second row. Therefore, they are not equivalent and we don't have to keep going with the table.

3. We can make truth tables for expressions involving more variables. For instance, here is the truth table for $p \vee (q \wedge r)$:

p	q	r	$p \vee (q \wedge r)$
T	T	T	$T \vee (T \wedge T) = T$
T	T	F	$T \vee (T \wedge F) = T$
T	F	T	$T \vee (F \wedge T) = T$
T	F	F	$T \vee (F \wedge F) = T$
F	T	T	$F \vee (T \wedge T) = T$
F	T	F	$F \vee (T \wedge F) = F$
F	F	T	$F \vee (F \wedge T) = F$
F	F	F	$F \vee (F \wedge F) = F$

There are 8 rows, corresponding to all the possible combinations of true and false for the three variables.

4. One interesting use of truth tables is for solving liar-truth-teller riddles. In these riddles, there are two tribes, one whose members always tell the truth and one whose members always lie. We meet a couple of tribe members who tell us something, and we have to tell to which tribe each belongs. Suppose we meet two who say the following:

- A: Exactly one of us is lying.
 B: At least one of us is telling the truth.

We can use logic to reason out who belongs to which tribe, but let's use the following truth table instead:

A's tribe	B's tribe	A's statement	B's statement
T	T	F	T
T	F	T	T
F	T	T	T
F	F	F	F

In the first row, we suppose that A and B are both truth-tellers. So there are two truth-tellers and no liars (this is the T T on the left half of row 1). In that case, A's statement, that exactly one person is lying, is false, while B's statement, that at least one is telling the truth, is true (this gives the F T in the right half of row 1). The left half, T T, doesn't match the right half, F T, so this row doesn't give the solution. Essentially, we have A being a truth-teller but telling a lie.

To find the solution, we find the table row where the first two columns match the last two columns, which occurs in row 4, where we have both A and B being liars.

1.2 Some logical rules

Order of operations

In a lot of ways, AND, OR, and NOT behave like arithmetic operators, with AND being like multiplication, OR being like addition, and NOT being like taking the negative of something. One place where the logical operators

behave like arithmetic operations is in order of operations. The NOT operation is always done first, followed by AND, and OR after that.¹

In most programming languages, AND is done before OR, which can lead to bugs if we are not careful. For instance, suppose we want something to happen if x is 3 or 4 and y is 1. Here is how the condition would look:

```
if (x == 3 or x == 4) and y == 1
```

The parentheses are absolutely needed. Since AND has a higher precedence than OR, leaving off the parentheses makes the condition equivalent to $x == 3 \text{ or } (x == 4 \text{ and } y == 1)$, which is different (for instance, $x = 3$ and $y = 0$ would make this one true, but not the original). In general, for the sake of clarity it is good to use parentheses, even when they aren't required.

Tautologies and contradictions

A *tautology* is an expression that is always true, and a *contradiction* is an expression that is always false. A simple example of a tautology is $p \vee \sim p$, while a simple example of a contradiction is $p \wedge \sim p$. The final column of the truth table of a tautology must consist entirely of T's, and the final column for a contradiction must consist of all F's.

In everyday English, the word *tautology* is used to describe a statement that is clearly true, usually due to a repetition of words. Examples include "Math is fun because math is fun" or "If it's cold outside, then outside it is cold."

De Morgan's laws

Here are two logical rules, called De Morgan's laws, that show up a lot:

- $\sim(p \vee q) = \sim p \wedge \sim q$
- $\sim(p \wedge q) = \sim p \vee \sim q$

These rules can easily be verified with truth tables. Here is an example of each of the rules:

1. Statement: *A or B went to the party.*
Negation: *A didn't go to the party and B also didn't go.*
2. Statement: *A and B went to the party.*
Negation: *Either A didn't go to the party or B didn't go to the party.*

One use for these rules is in simplifying if statements in programs. For instance, suppose we have the following complicated condition (where $==$ and $!=$ stand for *equal to* and *not equal to*):

```
if not (A==2 or B == 2):
```

We can use De Morgan's laws to simplify it to the following:

```
if A != 2 and B != 2:
```

Forgetting De Morgan's laws is a really common mistake. Remember that when negating $p \vee q$ and $p \wedge q$, not only do p and q get negated, but the logical symbol changes as well.

Other laws

There are a number of other logical rules, many of which are commonsense.

¹Note that some texts treat AND and OR as having the same level of precedence.

1. *Commutative rules*: The order of p and q in an AND or OR statement does not matter.

$$p \vee q = q \vee p \quad p \wedge q = q \wedge p$$

2. *Associative rules*: If two ANDs or two ORs are strung together, it doesn't matter which we do first.

$$p \vee (q \vee r) = (p \vee q) \vee r \quad p \wedge (q \wedge r) = (p \wedge q) \wedge r$$

3. *Distributive laws*: The second rule in particular is reminiscent of the algebra rule $a(b + c) = ab + ac$.

$$p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r) \quad p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$$

4. Let \mathcal{T} be a tautology and let \mathcal{C} be a contradiction. Then, we have

$$p \vee \mathcal{T} = \mathcal{T} \quad p \wedge \mathcal{T} = p \quad p \vee \mathcal{C} = p \quad p \wedge \mathcal{C} = \mathcal{C}$$

The first rule, for example, comes from the fact that an OR statement is true whenever at least one of its parts is true, and since \mathcal{T} is true, we must have $p \vee \mathcal{T}$ always being true, i.e. being a tautology.

5. A few simple rules:

$$p \vee p = p \quad p \wedge p = p \quad \sim(\sim p) = p$$

6. Absorption rules:

$$p \vee (p \wedge q) = p \quad p \wedge (p \vee q) = p$$

It might not be obvious at first glance why these are true. Consider the first statement. Suppose p is true. Then $p \vee (p \wedge q)$ is true since an OR statement is true whenever one of its parts is true. Suppose, on the other hand, that p is false. Then $p \wedge q$ must also be false, regardless of what q is, and so both parts of $p \vee (p \wedge q)$ are false, making the whole thing false. Thus, when p is true, the expression is true, and when p is false, the expression is false. Thus it is equivalent to p .

If these rules seem a lot like algebra, it's because they form a type of algebra called *Boolean algebra*.¹

We can use these rules and De Morgan's laws to simplify complicated expressions. For example, we can simplify $\sim(p \vee (\sim q \wedge p))$ like below:

$$\sim(p \vee (\sim q \wedge p)) = \sim p \wedge \sim(\sim q \wedge p) = \sim p \wedge (\sim(\sim q) \vee \sim p) = \sim p \wedge (\sim p \vee q) = \sim p.$$

The first two equalities are De Morgan's laws. The third uses both the double negative rule and commutativity, and the last equality is one of the absorption rules.

1.3 Universal quantifiers

The concepts of *every* and *there is* are common enough in math to warrant their own symbols, \forall and \exists . The symbol \forall is commonly read as *for all* and \exists is commonly read as *there exists*, though both of these can be rendered into English in any number of ways. These symbols are called *universal quantifiers*.

Here are a few examples of a statement in plain English followed by an equivalent using universal quantifiers:

1. Every cloud has a silver lining
 \forall cloud \exists silver lining.

¹In fact, there's an entire branch of math called abstract algebra that is concerned with structures like this where we take something—like logic, symmetries of shapes, or roots of polynomials—define some arithmetic-like operations on it, and use those operations in an algebraic way to do interesting things.

- Given any positive number, you can find another number larger than it.
 $\forall x > 0, \exists n$ such that $n > x$.
- For any positive number, you can find another positive number less than 1 unit away from it.
 $\forall x > 0, \exists y > 0$ such that $|x - y| < 1$.
- Given any two numbers, the square root of the sum of their squares is nonnegative.
 $\forall x \forall y, \sqrt{x^2 + y^2} \geq 0$.
 $\forall x, y$ we have $\sqrt{x^2 + y^2} \geq 0$. (an informal version of the previous line)

The phrase *such that* shows up in some of these examples. It is a common phrase and is often abbreviated as *s.t.* or with the symbol \ni .

Negating universal quantifiers

Here are two rules for negating \forall and \exists :

- Statement:* $\forall p, q$ is true.
Negation: $\exists p$ such that $\sim q$.
- Statement:* $\exists p$ such that q .
Negation: $\forall p, \sim q$ is true.

Sometimes, you might see \nexists as the negation of \exists . Here are a few examples of negation:

- Statement:* $\forall x, x^2 > 0$.
Negation: $\exists x$ such that $x^2 \leq 0$.

The original statement says that the square of any number is positive. To negate that statement, we have to show that there is (\exists) some number whose square is not positive (like 0, since $0^2 = 0$).

- Statement:* $\exists x$ such that $x^2 = -1$.
Negation: $\forall x, x^2 \neq -1$.

In other words, the negation of saying that some number has a square equal to 1 is to say that every number's square is different from 1.

- Statement:* Every person is at least 6 feet tall and has red hair.
Negation: There is some person who is under 6 feet tall or who doesn't have red hair.
 Notice the subtle application of De Morgan's law here.

- Statement:* $\forall x \exists y$ such that $y^2 = x$.
Negation: $\exists x$ such that $\forall y, y^2 \neq x$.

Notice that we have to apply both negation rules here. In general, if you have a really complicated expression, you can negate it by repeated application of the two rules.

- Statement:* $\forall p \exists q$ such that $\forall r \forall s \exists t$ such that $t^2 = 0$.
Negation: $\exists p$ such that $\forall q \exists r$ and $\exists s$ such that $\forall t, t^2 \neq 0$.

1.4 Conditionals

A statement of the form *If p, then q* is called a *conditional*. We use the notation $p \rightarrow q$. An example is the statement *If it is snowing, then it is cold*. In that statement, p is *it is snowing*, and q is *it is cold*.

The *contrapositive* of $p \rightarrow q$ is $\sim q \rightarrow \sim p$. Here is an example:

Statement: *If it is snowing, then it is cold.*
Contrapositive: *If it is not cold, then it is not snowing.*

The contrapositive of any statement is logically equivalent to the original statement. This is often useful in proofs. Sometimes, working with the contrapositive is easier than working with the original.

The converse of $p \rightarrow q$ is $q \rightarrow p$. Here is an example:

Statement: *If it is snowing, then it is cold.*

Converse: *If it is cold, then it is snowing.*

We see from this example that the converse is not logically equivalent to the original statement. Snow implies cold, but cold does not necessarily imply snow. In summary, given $p \rightarrow q$ is true, here's what we have:

$\sim p \rightarrow ?$ (If it's not snowing, we can't conclude whether it's cold or not.)

$q \rightarrow ?$ (If it's cold, we can't conclude whether it's snowing or not.)

$\sim q \rightarrow \sim p$ (Contrapositive: If it's not cold, then we can conclude that it's not snowing.)

Here is a more mathematical example. Consider the following statement:

If x is even, then $f(x) > 0$.

We don't care about exactly what the function f is. We just want to use logic to know what we can and can't conclude from this statement. Consider the following questions:

1. Question: If x is odd, what can we conclude about $f(x)$?

Answer: Nothing. Given $p \rightarrow q$, we can't conclude anything from $\sim p$.

2. Question: If $f(x) = -2$, what can we conclude about x ?

Answer: We can conclude that x is odd via the contrapositive.

3. Question: If $f(x) = 5$, what can we conclude about x ?

Answer: Nothing. Given $p \rightarrow q$, we can't conclude anything from q .

If and only if statements

Sometimes it does happen a statement and its converse are both true, that both $p \rightarrow q$ and $q \rightarrow p$ are true. In that case, we write $p \leftrightarrow q$ and say p *if and only if* q . This is sometimes called a *biconditional*. In this case, p and q are logically equivalent.

Consider the following example:

An integer n is even if and only if $n + 1$ is odd.

This statement works both ways:

Statement: *If n is even, then $n + 1$ is odd.*

Converse: *If $n + 1$ is odd, then n is even.*

Both the statement and its converse are true. This is the exception, rather than the rule. Most of the time, the converse won't always be true, but in those cases where the converse is also true, we use the phrase *if and only if* to indicate that fact.

Necessary and sufficient conditions

Given $p \rightarrow q$, we say that p is *sufficient* for q and q is *necessary* for p .

In general, suppose we have a statement that tells us about X having some property. We have the following:

- A sufficient condition can be used to tell us that X has the property, but not that it doesn't have the property.
- A necessary condition can be used to tell us that X does not have the property, but it can't tell us that it does have the property.

- A necessary and sufficient condition (an if and only if statement) can tell us both things—whether X has the property or not.

For example, consider the following statement:

If the integer n ends in 4, then it is even.

Ending in 4 is a sufficient condition for being even. Whenever a number ends in 4, it will be even. However, ending in 4 is not a necessary condition for being even, since it is possible for a number to be even but not end in 4. We can use this condition to find even numbers but we can't use it to say something is not even.

Now consider this statement:

If x is a perfect square, then it ends in 0, 1, 4, 5, 6, or 9.

In other words, perfect squares *must* end in those digits. If a number doesn't end in one of those digits, then it's not a perfect square. So ending in one of those digits is a necessary condition for being a perfect square. A necessary condition like this is usually used in its contrapositive form; namely, if x ends in 2, 3, 7, or 8, then it is *not* a perfect square.

Finally, consider the following:

The integer n is divisible by 10 if and only if it ends in 0.

Ending in 0 is both a necessary and a sufficient condition for being divisible by 10. In other words, if a number ends in 0, then you know it is divisible by 10, and, further, if a number is divisible by 10, then it must end in 0. Another way to think of this is that if a number ends in 0, it is divisible by 10, and if it doesn't end in 0, then it's not divisible by 10. An if and only if condition like this is useful in that it tells us two things: which numbers are divisible by 10 and which numbers are not divisible by 10.

An example

Suppose we have the following statement:

If x is prime, then $f(x)$ is even.

1. What is the contrapositive?
Answer: If $f(x)$ is odd, then x is not prime.
2. What is the converse?
Answer: If $f(x)$ is even, then x is prime.
3. Is x being prime a necessary or a sufficient condition for $f(x)$ being even?
Answer: It's sufficient. If x is prime, then we are guaranteed that $f(x)$ will be even. It is not a necessary condition, however, as there may be other ways for $f(x)$ to be even besides x being prime.
4. Is $f(x)$ being even a necessary or a sufficient condition for x being prime?
Answer: It is necessary. By the contrapositive, if $f(x)$ is not even, then x is not prime. So the only way x can be prime is if $f(x)$ is even.
5. If x is prime, what can we conclude about $f(x)$?
Answer: $f(x)$ must be even.
6. If x is not prime, what can we conclude about $f(x)$?
Answer: Nothing.
7. If $f(x)$ is even, what can we conclude about x ?
Answer: Nothing.
8. If $f(x)$ is odd, what can we conclude about x ?
Answer: x must not be prime. (This is the contrapositive.)

Conditionals and truth tables

A conditional has a truth table associated with it:

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

When we say that $p \rightarrow q$, this means that if p is true, then q must be true. This gives the first two lines of the truth table. The last two lines may be a bit puzzling. If p is not true, then why should $p \rightarrow q$ be true? In mathematics, we say that $p \rightarrow q$ is *vacuously true* in this case. For example, if we were to say “Every purple cow can fly,” that would technically be a true statement. It is technically true for *every purple cow*; it’s just that there aren’t any purple cows in existence. As another example, if I say that I’ve made a hole-in-one on every golf hole I’ve ever played, that would be a vacuously true statement since it is true for every one of the *zero* holes of golf I’ve played.

We can also see from the truth table that $p \rightarrow q$ is equivalent to $(p \wedge q) \vee \sim p$.

1.5 Some applications of logic

We have just covered the very basics of logic here, mostly just some notation and a few of the most important concepts. But there’s a whole lot more to logic. Logic forms an entire branch of mathematics, with many deep results. Here are a few applications of the logic we’ve covered here.

1. **Web searches** — When you do a web search with multiple terms, like *discrete mathematics*, by default search engines insert an (implicit) AND in there, as if the search were *discrete AND mathematics*, looking for all documents that have both those terms in there. However, if you want documents that contain at least one of those words, then you can insert an OR, like *discrete OR mathematics*. To exclude a term, a NOT operation, use a minus sign, like *discrete mathematics –logic* to find all documents that contain the words *discrete*, *mathematics*, but not the word *logic*. You can use parentheses to chain together more complicated searches. See your favorite search engine’s documentation for other things you can do. You might be surprised at the variety of options there are.
2. **Bitwise operations** — At the hardware level of a computer, everything is done in terms of bits and bytes. A bit is a value that can be 0 or 1, and a byte is a group of 8 bits. Computers don’t move individual bits around. They instead send whole bytes. Communicating with hardware often requires individual bits of a byte to be set to specific values. Logical operations like AND, OR, NOT, and XOR are used to set and extract those values. The trick is that 0 is equivalent to *false* and 1 is equivalent to *true*.

For example, if we have the byte 00101010, and we want to set the fourth bit to a 1, we can OR the byte with 0001000. The OR-ing is done bit-by-bit, and we get

```
  00101010
OR 00010000
-----
  00111010
```

OR-ing a bit with a 0 has no effect on it, while OR-ing a bit with 1 will set it to 1 if it is not already a 1.

As another example, XOR-ing a byte with 11111111 has the effect of flipping all the bits. Bitwise operations can be used to do all sorts of things, like read the value of a specific bit, flip a bit, or set a range of bits to something. They are fundamentally important when working at the hardware level or in assembly language.

3. **Formal mathematical proofs by computer** — Traditionally, mathematical proofs have been written out by hand in a somewhat formal, but also somewhat informal, manner, using a lot of plain English and omitting simple details that are easily filled in by the reader. Mathematical proofs could be written out in a completely formal way, using nothing but logical symbols and omitting no details, but that gets tedious very quickly

and it can be quite difficult to read and understand. Mathematical proofs are designed to convince the people reading them that a result is true, and a proof-writer often does all he/she can to make the ideas easily understood by the reader.

However, there are some benefits to using pure logical notation for proofs, especially when computers are involved. A mathematician who writes a proof out purely with logical notation can use a computer to check the result for subtle errors that a human might miss. There are also theorem-proving computer programs, where some basic definitions are inputted to the program using logical notation, and the program then actually does a search looking for and/or proving new results. Some new theorems that people have missed have been found this way. A web search for *automated theorem proving* will turn up more information.

4. **Satisfiability** — A satisfiability problem is one where you are given a logical expression, like $(p \vee q) \wedge (q \vee \sim r) \wedge (\sim p \vee r)$, and you want to know if there is any assignment of T 's and F 's to the variables that would make the statement true. In other words, we want to know whether the expression is a contradiction or not. For example, in the expression above, if we set p and r to F and q to T , then the overall statement is true.

Satisfiability problems are important in that many real-world and mathematical problems can be reduced to satisfiability problems.

Chapter 2

Sets

Roughly speaking, a *set* is a collection of objects. For example, $\{1, 2, 3, 4, 5\}$ is a set. There are many types of sets, like the set of all integers, the set of all prime numbers, the set of all lowercase letters, or the set of all cities in the U.S. with at least 100,000 people.

The order we write the elements doesn't matter: $\{1, 2\}$ and $\{2, 1\}$ are the same set. A set also has no repeats.

2.1 Common set notation and operations

Let A and B be sets. Here is some common notation:

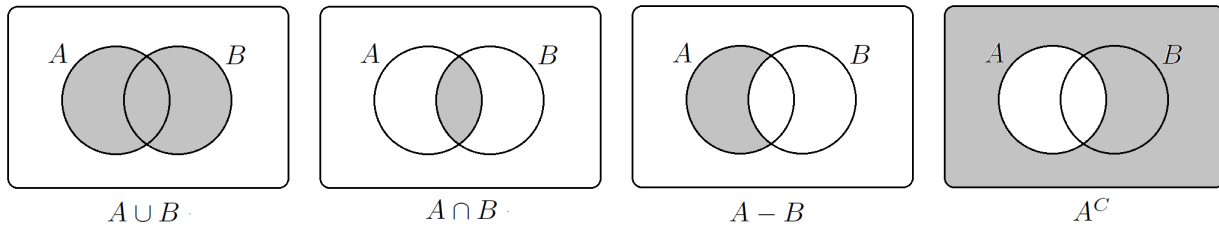
Notation	Explanation
$x \in A$	x is an element of the set A .
$A \subseteq B$	A is a <i>subset</i> of B . That is, everything in A is also in B .
$A \supseteq B$	A is a <i>superset</i> of B (B is a subset of A).
$ A $	The <i>size</i> or <i>cardinality</i> of A – the number of elements of the set.
$A \cup B$	The <i>union</i> of A and B – everything in either A , B , or both.
$A \cap B$	The <i>intersection</i> of A and B – everything in both A and B .
$A - B$	The <i>difference</i> of A and B – everything in A , but not in B .
A^C	The <i>complement</i> of A – everything that is not in A .
\emptyset	The <i>empty set</i> , a special set that contains no elements.

Notes: The term *everything* in the definition of the complement is a bit ambiguous. The complement is usually taken with respect to some universal set that will depend on the context. Also, a common way of saying A and B have nothing in common is $A \cap B = \emptyset$. In that case, we say that A and B are *disjoint*.

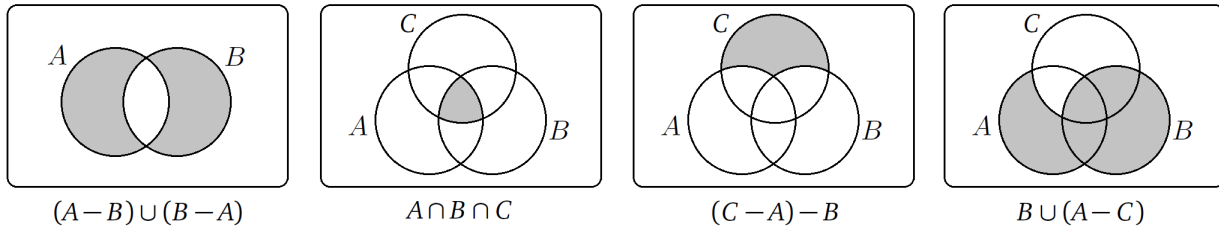
As an example, suppose $A = \{1, 2, 3, 4, 5\}$ and $B = \{4, 5, 6, 7\}$. We have

- $1 \in A$ and $8 \notin A$
- $\{1, 2, 3\} \subseteq A$ and $\{1, 2, 3, 4, 5, 6\} \supseteq A$
- $|A| = 5$ and $|B| = 4$
- $A \cup B = \{1, 2, 3, 4, 5, 6, 7\}$
- $A \cap B = \{4, 5\}$
- $A - B = \{1, 2, 3\}$ and $B - A = \{6, 7\}$
- If our universal set is all positive integers, then $A^C = \{6, 7, 8, 9, \dots\}$.
- $\{1, 2, 3\} \cap \{6, 7\} = \emptyset$.

Some of these operations can be illustrated with Venn diagrams, like below:



Here are a few more Venn diagram examples:



Notation in math is unfortunately not very standard. Here are a few common alternate notations you may run into:

- $A \setminus B$ for $A - B$
- \bar{A} or A' for A^C
- Some authors use $A \subset B$ to mean A is a subset of B that is not equal to B and $A \subseteq B$ to mean that A is a subset of B , possibly equal to B . Others use $A \subset B$ to mean that A is a subset of B , not making a distinction between whether A might possibly equal B .

2.2 Set identities

Here are some occasionally useful set identities.

1. *Commutative rules:* The order of the sets in a union or intersection doesn't matter.

$$A \cup B = B \cup A \quad A \cap B = B \cap A$$

2. *Associative rules:* If we have a union or intersection of three things, it doesn't matter which one we do first.

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

3. *Distributive laws:*

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

4. Let \mathcal{U} be the universal set. Then, we have the following:

$$A \cup \mathcal{U} = \mathcal{U} \quad A \cap \mathcal{U} = A \quad A \cup \emptyset = A \quad A \cap \emptyset = \emptyset$$

5. A few more rules:

$$A \cup A = A \quad A \cap A = A \quad (A^C)^C = A$$

6. Absorption rules

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

7. De Morgan's laws:

$$(A \cup B)^c = A^c \cap B^c \quad (A \cap B)^c = A^c \cup B^c$$

8. A useful rule for working with differences:

$$A - B = A \cap B^c$$

These identities are almost entirely analogous to those from Section 1.2 for \vee and \wedge , and that is no coincidence. This is one example of similar structures appearing in different mathematical settings.

We can use these identities to simplify expressions. For instance, we can simplify $((A^c \cup B^c) - A)^c$ like below:

$$((A^c \cup B^c) - A)^c = ((A^c \cup B^c) \cap A^c)^c = ((A^c \cup B^c)^c \cup (A^c)^c) = (A \cap B) \cup A = A.$$

The first equality uses the set difference rule and the second uses De Morgan's law. The third equality uses De Morgan again and the double complement rule. The last equality uses commutativity and the absorption rule.

Disproving set identities

To disprove a set identity, all that is necessary is to find a single counterexample. For example, suppose we want to disprove the following statement:

$$(A - B) \cup (B - A) = A \cup B.$$

If we let $A = \{1, 2\}$ and $B = \{2, 3\}$, then $(A - B) \cup (B - A) = \{1, 3\}$ but $A \cup B = \{1, 2, 3\}$. So the left and right sides of the identity are not equal, which means the identity is not true in general.

2.3 Cartesian product and power set

The *Cartesian product* $A \times B$ of sets A and B is the set consisting of all consists of all ordered pairs of the form (a, b) with $a \in A$ and $b \in B$. For example, if $A = \{1, 2, 3\}$ and $B = \{8, 9\}$, then

$$A \times B = \{(1, 8), (1, 9), (2, 8), (2, 9), (3, 8), (3, 9)\}.$$

The Cartesian product of three or more sets is defined in an analogous way. Cartesian products show up whenever we need a mathematical way to refer to pairs, triples, etc. of elements from a set. For instance, letting \mathbb{R} denote the set of all real numbers, $\mathbb{R} \times \mathbb{R}$ (or sometimes \mathbb{R}^2) denotes all ordered pairs of real numbers.¹

The Cartesian product shows up repeatedly in math and occasionally in computer science, particularly in databases.

The *power set* $\mathcal{P}(A)$ of a set A is the set of all subsets of A . For instance, if $A = \{1, 2, 3\}$, then

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

Notice that the elements of $\mathcal{P}(A)$ are themselves sets. It is possible to have sets whose elements are themselves sets. To take things further, a few elements of $\mathcal{P}(\mathcal{P}(A))$ includes the elements $\{\{1\}\}$ and $\{\{1, 2\}, \{3\}, \phi\}$, among 254 others.

In general, the size of $\mathcal{P}(A)$ is $2^{|A|}$. For instance, $A = \{1, 2, 3\}$ and we have $2^3 = 8$ elements in its power set.

¹This is sometimes called the Cartesian plane, a term familiar from high school algebra and named for Renè Descartes, who introduced the idea.

2.4 Notation

Sets are very important in math, and there are a number of different ways of describing sets mathematically. First, here are some important sets:

Symbol	Name	Description
\mathbb{N}	Natural numbers	1, 2, 3, 4, ...
\mathbb{Z}	Integers	0, ± 1 , ± 2 , ± 3 , ± 4 , ...
\mathbb{Q}	Rational numbers	Fractions (ratios of whole numbers, like 1/2 or 9/4)
\mathbb{R}	Real numbers	The entire number line

The sets \mathbb{Z}^+ , \mathbb{Q}^+ and \mathbb{R}^+ refer to the positive integers, rationals, and reals. Note also that some authors take 0 to be a natural number.

Notice that we have $\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}$. That is, every natural number is also an integer, every integer is also a rational number (for instance, 2 is the fraction 2/1), and every rational number is also a real number.

Set-builder notation

A common way of describing sets is *set-builder notation*. Here is an example:

$$\{x \in \mathbb{R} : x^2 = 1\}$$

We read this as “The set of all real numbers whose square is 1.” The colon is read as “such that.” Many authors use a vertical bar instead of a colon.

Here are some more examples:

Setbuilder notation	Actual values
$\{n \in \mathbb{N} : n \text{ is prime and ends in } 3\}$	$\{3, 13, 23, 43, 53, 73, 83, \dots\}$
$\{(a, 2a) : a \in \mathbb{N}\}$	$\{(1, 2), (2, 4), (3, 6), (4, 8), \dots\}$
$\{n \in \mathbb{N} : n = 2k \text{ for some } k \in \mathbb{N}\}$	$\{2, 4, 6, 8, \dots\}$
$\{n \in \mathbb{Z} : \exists k \in \mathbb{Z} \text{ s.t. } n = k^2\}$	$\{0, 1, 4, 9, 16, 25, 36, 49, \dots\}$
$\{(x, y, z) : x, y, z \in \mathbb{N} \text{ and } x^2 + y^2 = z^2\}$	$\{(3, 4, 5), (6, 8, 10), (5, 12, 13), (7, 24, 25), \dots\}$

Here a few more examples of set notation showing up in mathematical expressions:

Math	English
$\forall x \in \mathbb{R}^+ \exists n \in \mathbb{N} \text{ s.t. } \frac{1}{n} < x$	Given any positive real number, you can find a natural number whose reciprocal is less than it
$ \{(x, y) : x, y \in \mathbb{Z} \text{ and } 2y - x = 0\} = \infty$	There are infinitely integer solutions to $2y - x = 0$
$\{n \in \mathbb{Z} : n = 2k + 1 \text{ for some } k \in \mathbb{Z}\}$	The set of all odd integers
$\{x \in \mathbb{R} : x^2 \in [1, 4]\}$	All the real numbers whose squares are between 1 and 4 (inclusive)

Big unions and intersections

Sometimes, we have a union of several sets, like $A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \cup A_6 \cup A_7 \cup A_8$. We can write this with the shorthand $\bigcup_{n=1}^8 A_n$. If we have infinitely many sets $A_1 \cup A_2 \cup \dots$, we can write $\bigcup_{n=1}^{\infty} A_n$. A similar notation exists for intersections.

As an example, using $[a, b]$ to denote all the real numbers between a and b (including a and b), what is $\bigcap_{n=1}^{\infty} [-\frac{1}{n}, \frac{1}{n}]$?

Solution: The intersection consists of the elements that are in every one of those sets. There is only one such element, 0, so the intersection is $\{0\}$. No other real number is in the intersection as given any positive number a , we can find an integer n such that $\frac{1}{n} < a$.

Chapter 3

Functions and Relations

When many people think of functions, they think of formulas like $f(x) = x^2 + 1$ or $g(x) = \sin(x)$. The mathematical definition of a function is more general. We have two sets: a set X of inputs to the function and a set Y of outputs. A function is a rule that takes each element in X and assigns it an element in Y .

The set X specifies what all the possible inputs are. It is called the *domain*. The set Y describes what the outputs look like. It is called the *codomain*. We write $f : X \rightarrow Y$ to denote that f is a function from X to Y .

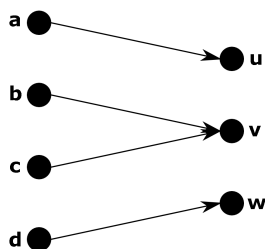
Not everything in the codomain will necessarily be “hit” by an element from the domain. The set of all elements that are hit is called the *range*. Formally, the range of $f : X \rightarrow Y$ is $\{y \in B : y = f(x) \text{ for some } x \in X\}$.

Here are some examples:

1. A familiar function like $f(x) = x^2$ can be thought of as a function $f : \mathbb{R} \rightarrow \mathbb{R}$. It is a rule that takes a real number x and assigns it another real number x^2 . Both the inputs and outputs (domain and codomain) are real numbers. The range is the interval $[0, \infty]$.
2. Let P be the set of all people in the world, and define $f : P \rightarrow \mathbb{Z}$ such that $f(p)$ is the age of p . We are assigning each person an integer, which is their age. The inputs are people and the outputs are integers. The range is the set $\{0, 1, 2, \dots, 115\}$ (where 115 is the age of the oldest person alive as I write this).
3. Many things we don't think of as functions can be written as functions. For instance, ordinary addition can be thought of as a function. The operation $2 + 7 = 9$ corresponds to taking the input $(2, 7)$ and returning the output 9. The domain is $\mathbb{R} \times \mathbb{R}$, which is all ordered pairs of real numbers. The codomain and range are both \mathbb{R} .

We will often use the informal terms “hit,” “map,” or “send” to describe the action of a function. We can think of things in the domain being mapped or sent over into the codomain and some (but not necessarily all) of the things in the codomain as being hit by things from the domain.

We will use diagrams like the one below to illustrate concepts involving functions on small sets.



This diagram indicates a function from the set $\{a, b, c, d\}$ to the set $\{u, v, w\}$ with $f(a) = u$, $f(b) = f(c) = v$ and $f(d) = w$.

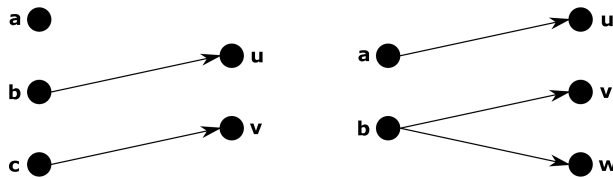
We will also use the following notation in some example problems: Let A be a set of characters, like $A = \{a, b, c\}$. The set A^* denotes the set of all *words* from A , that is all the strings of characters we can make from the characters of A . Some example words from $A = \{a, b, c\}$ include $abaabcca$, $abba$, a , and $\{\}$, which is the empty word.

3.1 Well-defined functions

Mathematicians often use the term *well-defined* to mean that a function has a clear and unambiguous definition. In particular, for a function to be well-defined, every element in the domain must be mapped to one and only one thing. That is,

1. Everything in the domain must get sent to something.
2. Nothing in the domain can get sent to more than one thing.

The figure below illustrates where the two possibilities can go wrong:



Neither is a well-defined function. On the left $f(a)$ is undefined, and on the right, $f(b)$ is ambiguously defined.

Here are a few examples of functions that are not well-defined.

1. Let f be a function from the set of all people to itself, such that $f(p)$ is person p 's favorite author. This function is not well-defined because not everyone has a favorite author. Also, some people might have two favorite authors.

For a function to be well-defined, it must be completely clear how f acts on every element of the domain. Here we have two problems. There are some values of p for which $f(p)$ has no possible value and other values of p for which it is not clear which of two or more values $f(p)$ should be.

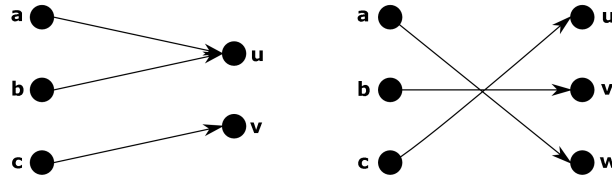
2. Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be given by $f(x) = 1/x$. Then f is not well-defined because $f(0)$ is undefined. We could make this into a well-defined function by changing the domain from \mathbb{R} to $\mathbb{R} - \{0\}$.
3. Let $A = \{a, b, c\}$ and define $f : A^* \rightarrow \mathbb{Z}$ such that $f(w)$ is the location of the first a in the word w . This is not well-defined because not every word has an a . For instance, $f(bbb)$ is undefined.
4. Let $A = \{a, b, c\}$ and define $f : A^* \rightarrow \mathbb{Z}$ such that $f(w)$ is the location of an a in the word w . This is not well-defined because we are not precise enough. If w has multiple a 's, then any of their locations could be possibilities for $f(w)$. This is not allowed. We must have only one possibility for each element of the domain.

These examples may seem simple, but in higher math, when you are working with some complicated and abstract structures, it often happens that when you define a function, it is not clear at first that it is in fact a well-defined function, and there is a lot of work that goes into showing that it really is well-defined.

3.2 One-to-one functions

It is possible for an output to be hit by multiple inputs. For instance, given $f(x) = x^2$, the output 9 is hit by two inputs, 3 and -3. If this doesn't happen, if every output is hit by no more than one input, we call the function *one-to-one*.

On the left below is a function that is not one-to-one as u is hit by two inputs. The function on the right is one-to-one.



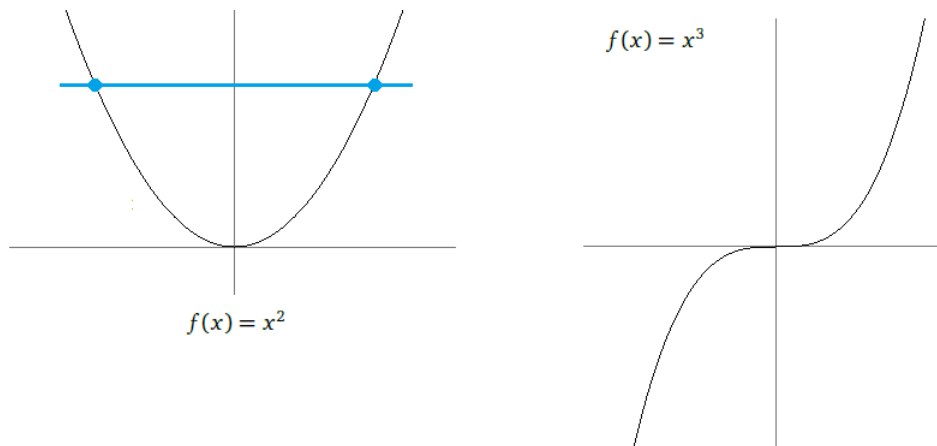
The formal definition of $f : X \rightarrow Y$ being one-to-one is that whenever $f(x_1) = f(x_2)$, we have $x_1 = x_2$. In other words, the only way two elements of the domain, x_1 and x_2 , can be sent to the same place is if x_1 and x_2 are the same element. This definition is useful for formally proving functions are one-to-one. In particular, it can give us something to work with algebraically.

Here are a few examples of functions that are and aren't one-to-one.

1. The function $f : \mathbb{R} \rightarrow \mathbb{R}$ given by $f(x) = x^2$ is not one-to-one because $f(-2)$ and $f(2)$ both get mapped to 4. In other words, we have an element in the codomain that is hit twice.

In general, for functions from \mathbb{R} to \mathbb{R} , the horizontal line test can be used to test if the function is one-to-one. Graph the function and if any horizontal line hits the function in more than one place, then it is not one-to-one.¹

On the other hand, $f(x) = x^3$ is one-to-one. Notice that no horizontal line hits the function in more than one place. See the figure below:



2. Let P denote the set of all people, and define $f : P \rightarrow \mathbb{Z}$ such that $f(p)$ is p 's age in years (as a whole number). Then f is not one-to-one since there are many people with the same age. For instance, the output 21 corresponds to millions of possible inputs.
3. Let S denote the set of all students at a certain college and define $f : S \rightarrow \mathbb{Z}$ such that $f(s)$ is the student ID number of s . This is one-to-one, and in fact, it would be bad if it weren't as then two students would have the same ID number.
4. Let $A = \{a, b, c\}$ and define $f : A^* \rightarrow A^*$ such that $f(w)$ is obtained from w by removing all the c 's from w . For instance $f(cabcacc) = aba$. This function is not one-to-one because, for instance, $f(ac) = a$ and $f(acc) = a$. In fact, everything in the range will be hit infinitely often, so this is very much not one-to-one.
5. On the other hand, let $A = \{a, b, c\}$ and define $f : A^* \rightarrow A^*$ such that $f(w)$ is obtained from w by reversing the letters of w . This function is one-to-one.

¹Contrast this to the vertical line test, where you draw a vertical line and see if it hits the graph in more than one place. Failing the vertical line test means the function is not well-defined as there would be something in the domain being mapped to multiple things.

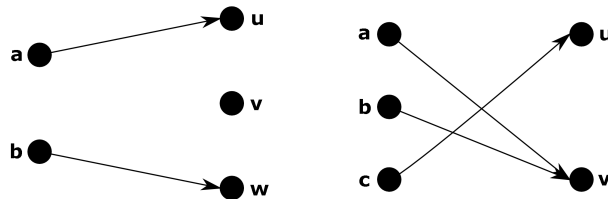
6. One important place that the concept of one-to-one functions is important is in hash functions. When you log into a site with a password, your password isn't (or shouldn't be) stored in plain text. Instead the password is run through something called a hash function that scrambles it into a string like 2fj34Ferijd324H. That hash value is stored on the server, and when you log on, your password is run through the hash function and compared to the hash value on the server. The scrambled value is designed so that it is difficult to reconstruct the password from it so that if the server is compromised, the attacker won't be able to get someone's password and potentially use it on other sites.

But an artifact of the hashing process is that the hash function is not one-to-one. In other words, it is possible (though unlikely) that two passwords could lead to the same hash value. This is called a collision. This is important in digital signatures. Hash functions are used for creating digital signatures of documents so that people can be sure they are getting an authentic copy of a document. But if an attacker finds a collision, they can use that to create two copies of a document that have the same hash value. Since the real and the fake document will both have the same hash value, the digital signature algorithm will treat both as being authentic.

3.3 Onto functions

For many functions, not everything in the codomain will necessarily be hit by something from the domain. The codomain specifies the type of output (like real numbers, integers, etc.), but not every possible real number, integer, etc. will be necessarily be hit. If everything in the codomain is hit, then we call the function *onto*.

On the left below is a function that is not onto. In particular, v is not hit by any inputs. The function on the right is onto.



Formally, $f : X \rightarrow Y$ is called onto if for every $y \in Y$, we can write $y = f(x)$ for some $x \in X$. In other words, the range and codomain are equal.

Here are a few more examples:

1. The function $f : \mathbb{R} \rightarrow \mathbb{R}$ given by $f(x) = x^2$ is not onto because there are negative values in the codomain that are not hit. On the other hand, $f(x) = x^3$ is onto.
2. Let P denote the set of all people, and define $f : P \rightarrow \mathbb{Z}$ such that $f(p)$ is p 's age in years (as a whole number). Then f is not onto since there many ages are not attained (like 200 or any negative age).
3. Let $A = \{a, b, c\}$ and define $f : A^* \rightarrow A^*$ such that $f(w)$ is obtained from w by removing all the c 's from w . For instance $f(cabcacc) = aba$. This function is not onto because anything containing a c will not be hit.
4. On the other hand, let $A = \{a, b, c\}$ and define $f : A^* \rightarrow A^*$ such that $f(w)$ is obtained from w by reversing the letters of w . This function is onto.

Note that if a function is onto, then its range and codomain are equal. You can always change a function into an onto function by removing all the things that are not hit from the codomain.

3.4 Bijections

A function that is both one-to-one and onto is called a *bijection*.¹ Here are a few examples of bijections:

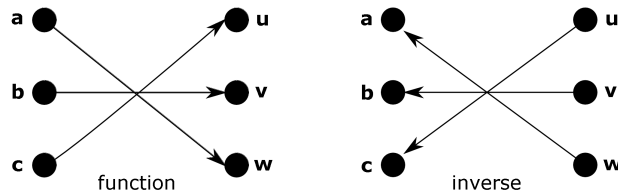
¹A few alternate terms you might see are *injection* for one-to-one, *surjection* for onto, and *one-to-one correspondence* for bijection.

1. Let $E = \{\dots, -4, -2, 0, 2, 4, \dots\}$ denote the set of all even integers. Then $f : \mathbb{Z} \rightarrow E$ be given by $f(z) = 2z$ is a bijection.
2. Let $A = \{1, 2, 3, 4, 5\}$ and $B = \{11, 12, 13, 14, 15\}$ then $f : A \rightarrow B$ defined by $f(x) = x + 5$ is a bijection. An interesting and useful result is that the only way there can be a bijection between two finite sets is if they have the same number of elements.
3. Let $A = \{0, 1, 2, 3, 4\}$, then $f : A \rightarrow A$ defined by $f(x) = 4 - x$ is a bijection. Note that any bijection from a set to itself is just a rearrangement (a permutation) of its elements.

3.5 Inverses

If a function is a bijection, then it has an *inverse*. The inverse of a function $f : X \rightarrow Y$ is a function $f^{-1} : Y \rightarrow X$ such that if $y = f(x)$, then $f^{-1}(y) = x$. In other words, f^{-1} undoes the effects of f .

One way to think of an inverse is as reversing the arrows, like in the figure below.



Here are a few examples of inverses:

1. Let $f(x) = 2x + 1$. We get $f(x)$ by multiplying x by 2 and then adding 1. The inverse is the reverse of this process, namely we first subtract 1, and then divide by 2. In other words, $f^{-1}(x) = (x - 1)/2$.
2. Let $A = \{a, b, c, \dots, z\}$ be our usual 26-letter alphabet, let $B = \{1, 2, 3, \dots, 26\}$, and define $f : A \rightarrow B$ such that $f(a)$ is the position of the letter a in the alphabet. Then f^{-1} gives the letter at a given position. For instance, $f^{-1}(26) = z$.
3. Let $A = \{a, b, c\}$ and define $f : A^* \rightarrow A^*$ such that $f(w)$ is obtained from w by changing any a 's to b 's and b 's to a 's. For instance, $f(abcaa) = bacbb$. Then f is its own inverse. That is, f^{-1} is the same as f .

Note that a function must be a bijection in order to have an inverse. Thinking of the inverse process as reversing the arrows, we see that if f is not onto, then f^{-1} would not be well-defined as there would be something in the domain of f^{-1} that has nothing to map to. And if f is not one-to-one, then there are points where f^{-1} would be ambiguously defined.

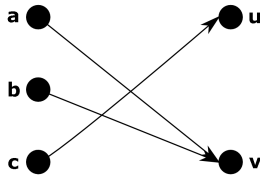
3.6 Preimages

A related notion to inverse functions is the *preimage* or *inverse image*. Given $f : X \rightarrow Y$, the preimage of a set $A \subseteq Y$ is $\{x \in X : f(x) \in A\}$. In other words, it is all the inputs that have an output in A .

The preimage of A is denoted $f^{-1}(A)$. It is a little unfortunate that a similar notation is used for both preimages and inverse functions. One way to tell them apart is that the preimage is always applied to a set.

Here are a few examples:

1. In the figure below, $f^{-1}(\{v\}) = \{a, b\}$ and $f^{-1}(\{u, v\}) = \{a, b, c\}$.

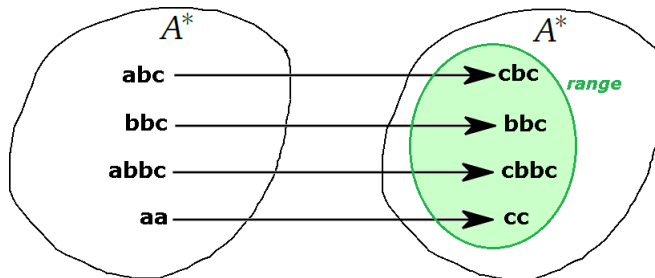


- Let $f(x) = x^2$. Then $f^{-1}(\{4\}) = \{-2, 2\}$ and $f^{-1}([4, 9]) = [-3, -2] \cup [2, 3]$.
- Let P be the set of all people alive today. Define $f : P \rightarrow \mathbb{Z}$ such that $f(p)$ is the age of p in years. Then $f^{-1}(\{40\})$ is the set of all 40-year-olds and $f^{-1}(\{13, 14, 15, 16, 17, 18, 19\})$ is the set of all teenagers.
- Let $A = \{a, b\}$ and define $f : A^* \rightarrow \mathbb{Z}$ such that $f(w)$ is the length of the word w . Then $f^{-1}(\{2\})$ consists of all the words of length 2, namely $\{aa, ab, ba, bb\}$. Similarly, $f^{-1}(\{0, 1, 2\})$ consists of all the words of length 2 or less, namely $\{aa, ab, ba, bb, a, b, \{\}\}$.

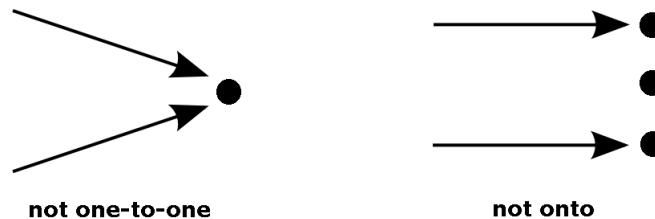
3.7 A combined example

Here is an example of all the concepts. Let $A = \{a, b, c\}$ and suppose $f : A^* \rightarrow A^*$ is defined such that $f(w)$ is obtained from w by replacing each a of w with a c .

- Well-defined** – This is a well-defined function. Every input has an output and it is clear that there is only one possible output for a given input.
- Domain and codomain** – The domain and codomain are given in the expression $f : A^* \rightarrow A^*$. Both are A^* .
- Understanding the function** – To get a handle on how the function behaves, try a few examples, like below.



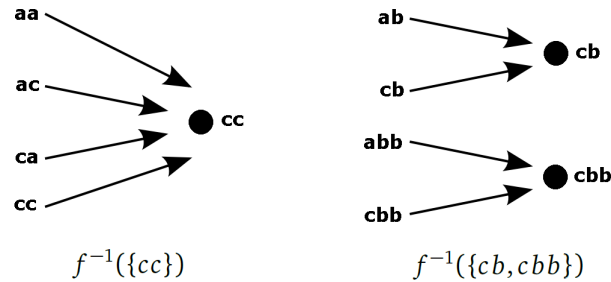
- Range** – From these examples, we see that the range is all words that contain only b 's and c 's.
- One-to-one** – To determine if the function is one-to-one, we have to see if we ever have a situation like the one shown below on the left, where two inputs go to the same output. This does happen for our function as both $f(aa) = cc$ and $f(cc) = cc$. So it is not one-to-one.



- Onto** – To determine if the function is onto, we have to see if we have a situation like the one shown above on the right, where something in the codomain is not hit by any inputs. This also happens for our function as anything containing an a is not hit. Thus, the function is not onto.
- Inverse** – This function is neither one-to-one nor onto, so it has no inverse.

8. **Preimages** – Suppose we want to know the preimage $f^{-1}(\{cc\})$. In other words, we want to find all of the inputs that give us cc as an output. There are four such inputs, aa , ac , ca , and cc . Thus $f^{-1}(\{cc\}) = \{aa, ac, ca, cc\}$.

As another example, suppose we want the preimage $f^{-1}(\{cb, cbb\})$. This is all the inputs that give us either cb or cbb . They are ab , cb , abb , and cbb . See the figure below.



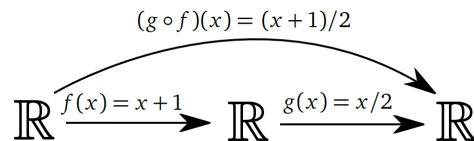
On the other hand, the preimage $f^{-1}(\{a\}) = \emptyset$, since there are no inputs that give us an output of a .

Composition of functions

Given two functions $f : X \rightarrow Y$ and $g : Y \rightarrow Z$, the *composition* of f and g , denoted $g \circ f$ is the function obtained by first applying f and then applying g . In particular, for any $x \in X$, we have $(g \circ f)(x) = g(f(x))$.

In other words, a composition is a combination of two functions. Here are a few examples:

1. Let $f(x) = x + 1$ and $g(x) = x/2$. Then $(g \circ f)(x) = g(f(x)) = (x + 1)/2$. We get $(x + 1)/2$ from x by first adding 1 and then dividing by 2. The action of f is to add 1 and the action of g is to divide by 2, and both actions are combined in $g \circ f$. See the figure below.



2. Let $A = \{a, b, c\}$ and define $f : A^* \rightarrow A^*$ such that $f(w)$ is obtained from w by removing every a from w . Define $g : A^* \rightarrow \mathbb{Z}$ such that $g(w)$ is the length of w . Then $(g \circ f)(w)$ is obtained by first removing every a from w and taking the length of the remaining letters. In other words, $(g \circ f)(w)$ returns the number of b 's and c 's in w .
3. Let $A = \{a, b, c, d, e, f\}$ and define $f : A^* \rightarrow A^*$ and $g : A^* \rightarrow A^*$ such that $f(w)$ rotates the letters of w to the right, with the last one wrapping around to the front, while $g(w)$ reverses the letters of w . For example, consider $(g \circ f)(abcdef)$, which is $g(f(abcdef))$. We have $f(abcdefg) = fabcde$ and $g(fabcde) = edcbaf$. So we see that $g \circ f$ is a function that reverses all the letters of a word except the last, which it leaves fixed.

Notice also that both f and g are bijections and hence have inverses. The inverse of f rotates letters to the left, while g is its own inverse. The inverse $(g \circ f)^{-1}$ is given by $f^{-1}(g^{-1}(x))$. Note that the order is reversed from $g \circ f$. When we do $g \circ f$, we first do f and then do g . So to undo the effect, we first have to undo g and then undo f .

3.8 Relations

A function is a rule that takes a set of inputs and assigns a single output to each one. For a function to be well-defined, each input needs an output and cannot have more than one output. If we remove these restrictions, then we have the concept of a *relation*.

With a function f , we use the notation $f(x) = y$ to say that x is mapped to y . With a relation R , we use the notation $x R y$ to indicate that x is related to y . It is possible to have x related to multiple things, like $x R y_1$ and $x R y_2$.

Many discrete math books spend a lot of time on relations. We will just cover the basics here. Here are a few examples of relations:

1. Let P be the set of all people alive right now and consider the relation R between P and \mathbb{Z} such that $p \in P$ is related to $y \in \mathbb{Z}$ provided p is/was in college during year y .

We can see why this is not a function — some people don't go to college, and most people go to college for more than one year.

However, this is a relation. It just ties together some elements from P with some elements from \mathbb{Z} . For instance, basketball player Michael Jordan (MJ) was in college from 1981 to 1984, so we have $MJ R 1981$, $MJ R 1982$, $MJ R 1983$, and $MJ R 1984$.

2. As another example consider the relation R between \mathbb{N} and itself such that $n R d$ provided d is a divisor of n . For instance, $4 R 1$, $4 R 2$, and $4 R 4$ since the divisors of 4 are 1, 2, and 4. Again, this is not a function since each integer has several divisors.

The formal definition of a relation is between two sets X and Y is that it is subset of the Cartesian product $X \times Y$. The elements, (x, y) of the subset are all the things for which $x R y$.

In math, probably the most important type of relation is an *equivalence relation*. An equivalence relation from a set X to itself is a relation \sim that satisfies the following three properties:

1. (Reflexive property) For all $x \in X$, we have $x R x$. That is, everything is related to itself.
2. (Symmetric property) Whenever $x R y$, we also have $y R x$. That is, if x is related to y , then y is related to x .
3. (Transitive property) Whenever $x R y$ and $y R z$, we also have $x R z$. That is, if x is related to y which is in turn related to z , then x is related to z .

Here are a few examples:

1. Let $A = \{a, b, c\}$ and let R be the relation on A^* , where $w_1 R w_2$ provided w_1 starts with the same letter as w_2 . This is reflexive, since a word starts with the same letter as itself. It is symmetric, if w_1 starts with the same letter as w_2 , then w_2 starts with the same letter as w_1 . It is also transitive, since if w_1 starts with the same letter as w_2 and w_2 starts with the same letter as w_3 , then w_1 starts with the same letter as w_3 . Thus this relation is an equivalence relation.
2. Let R be the relation on the set of all people ever, where $x R y$ provided y a parent of x . This relation is not reflexive, since a person cannot be their own parent. It is not symmetric, since if x is a parent of y , then y cannot be a parent of x (i.e., your mom is your parent, but you aren't a parent of your mom). It is not transitive, since if y is the parent of x and z is a parent of y , then it doesn't follow that z is a parent of x (z is a grandparent of x , but not a parent). This relation satisfies none of the properties, so it is not an equivalence relation.
3. Let R be the relation on \mathbb{N} where $n R d$ provided d is a divisor of n . This relation is reflexive because any integer is a divisor of itself. It is not symmetric because, for example, 4 is divisible by 2 but 2 is not divisible by 4. It is transitive, since if a is a divisor of b and b is a divisor of c , then a is a divisor of c . This relation is not an equivalence relation because it is not symmetric.

Equivalence relations often correspond to notions of equality or sameness. The first example above, for instance is about words having the same first letter. Other examples of equivalence relations include equality of numbers, people being the same age, numbers leaving the same remainder, or words having the same length.

3.9 Logarithms

Consider the problem of finding an x such that $2^x = 47$. We know x should be between 5 and 6, since $2^5 = 32$ and $2^6 = 64$, but what is the exact value? It turns out to be about 5.5546. The way to find this is to use logarithms.

In general, $\log_b(a)$ answers the question “What power do we have to raise b to in order to get a ?” That is, it solves the equation $b^x = a$. For example, $\log_2 16$ is 4, since $2^4 = 16$ and $\log_5 25 = 2$, since $5^2 = 25$. We also have $\log_{10}(.01) = -2$, since $10^{-2} = .01$.

The most common bases (values of b) are 10, e , and 2.

1. The base 10 logarithm, (\log_{10}), is common key on many calculators. Base 10 logs used to be important for doing calculations before calculators. Slide rules are (were) mechanical devices built around \log_{10} for doing calculations.
2. The base 2 logarithm, \log_2 , is especially useful in computer science, since so many things in computer science are based around the number 2.
3. The base e logarithm, or *natural logarithm*, is usually denoted by \ln or just \log . It is fundamentally important in calculus and much of higher math.

For computing logarithms to various bases, the following formula is useful, since most scientific calculators have a natural log key:

$$\log_b(a) = \frac{\ln(a)}{\ln(b)}.$$

For example, $\log_2(47) = \frac{\ln(47)}{\ln(2)} \approx 5.5546$.

Properties of logs

Here are some occasionally useful properties of logarithms (\log here represents a logarithm to any base):

1. $\log(ab) = \log(a) + \log(b)$
2. $\log(a/b) = \log(a) - \log(b)$
3. $\log(b^a) = a \log(b)$
4. $b^{\log_b(a)} = a$ and $\log_b(b^a) = a$
5. The solution of $b^x = a$ is $x = \log_b(a)$ or $x = \frac{\ln(a)}{\ln(b)}$.

For example, the first property tells us that $\log(3) + \log(4) = \log(12)$. The first and second properties are useful for combining and breaking up logs. See examples 5 and 6 in the next section for a practical example.

The third property is particularly useful. A simple example is $\log(2^4) = 4 \log(2)$. Several examples in the next section rely on this property.

The fourth and fifth properties are useful when solving equations with logarithms. The fourth property can be interpreted as saying that logarithms and exponentials are inverses of each other.¹

A firm understanding of the definition along with these properties is most of what you need in order to be successful with logarithms.

¹It might be helpful to think of it this way: The inverse of addition is subtraction. The inverse of multiplication is division. And the inverse of exponentiation is taking logarithms.

3.10 Some places where logarithms show up

1. **Solving equations** — Any time we have an equation where the variable we want to solve for is in the exponent, a logarithm might be handy. For instance, to solve $2^x = 39$, we get $x = \log_2(39)$ or $x = \ln(39)/\ln(2)$, which is about 5.2854.
2. **Population** — A population of bacteria, animals, or people can grow exponentially if left unchecked. For example, suppose we start with 100 bacteria, and the colony doubles in size every hour. How many will there be after 3 hours? We have $100 \rightarrow 200 \rightarrow 400 \rightarrow 800$, so there will be 800 bacteria. We can think of 800 as $100 \cdot 2^3$. Now suppose we want to know how long it will take to get to 20,000 bacteria. To get this, we solve $100 \cdot 2^x = 20000$. First divide by 100 and then take the logarithm to get $x = \log_2(200)$, or about 6.6 hours.
3. **Interest** — Suppose we put \$100 in the bank at 2% interest, compounded once per year. After one year, we will have \$102. After two years, we will have \$104.04. The interest builds on itself (i.e., it *compounds*). For the second year, we have the 2% of 100 plus the 2% on the \$2 interest from the first year. The interest continues to build on itself exponentially. We will have $100 \cdot 1.02$ after one year. After two years, it's $100 \cdot 1.02 \cdot 1.02$ or $100(1.02)^2$. After n years we will have $100(1.02)^n$.

In general, if P is our principal amount, r is the interest rate, and n is the number of years, then the amount after n years will be

$$P(1 + r)^n.$$

Here's where logarithms come in: Suppose we want to know how long it will take for our \$100 investment to double. We set up the equation $100(1.02)^n = 200$. We can solve this to get $n = \log_{1.02}(2)$ or $n = \ln(2)/\ln(1.02)$, which is about 35 years.

If the interest is compounded more often, say k times a year, then the formula becomes

$$P \left(1 + \frac{r}{k} \right)^{kn}.$$

Suppose we have \$1000 on a credit card with APR 18%. Interest on credit cards is compounded daily. How long will it take to grow to \$10,000?

We solve $1000(1 + .18/365)^{365n} = 10000$. Start by dividing through by 1000 to get $(1 + .18/365)^{365n} = 10$. If we take the natural logarithm of both sides, we get $\ln(1 + .18/365)^{365n} = \ln(10)$. Use one of the log properties to bring 365n down front, and solve for n to get

$$n = \frac{\ln(10)}{365 \ln(1 + .18/365)} = 12.8 \text{ years.}$$

4. **Digits in a number** — Suppose we want to know how many digits a big number like 3^{25000} has. That number is too big for most calculators. However, logs help us here. Numbers between 10 and 100 (10^1 and 10^2) have 2 digits, numbers between 100 and 1000 (10^2 and 10^3) have 3 digits. In general, numbers between 10^n and 10^{n+1} have $n + 1$ digits. Thus to tell how many digits a number n has, we want to solve $10^x = n$ and round up. In other words, we compute $x = \log_{10}(n)$ and round up to get the number of digits of n .

For our example $n = 3^{25000}$, we have $\log_{10}(3^{25000}) = 25000 \log_{10}(3)$, which gives us 11,929 digits.

5. **Calculations before calculators** — Here is an example of some of the log properties in action. In the old days before calculators, logarithms were used to help with multiplication. For instance, let's use logs to help us do 87×213 .

Start with $\log_{10}(87 \times 213)$. Write 87 and 213 in scientific notation, so we have $\log_{10}(8.7 \times 10^1 \times 2.13 \times 10^2)$, which we can write as $\log(8.7 \times 2.13 \times 10^3)$. Break it up into

$$\log_{10}(8.7) + \log_{10}(2.13) + \log_{10}(10^3).$$

The last term simplifies to 3. For the first two terms, in the old days we would look them up in a table of logarithms to get .9395 and .3284. Add up all three values to get 4.2679. This is the key idea of the logarithm approach: it turns multiplication into addition.

To undo the logarithm, we raise 10 to the 4.2679 power. We have $10^{4.2679} = 10^4 10^{.2679}$. We would then look up .2679 backwards in a log table to get 1.8531. Thus we have our answer, 1.8531×10^4 or 18,531.

6. **Logs and multiplying small numbers** — There’s a type of email spam filter called a Bayesian spam filter. The way it works is you first train the filter with a lot of real emails and spam emails. For each word or phrase, you end up with frequencies: the word appears in such-and-such percentage of spam emails and such-and-such percentage of real emails, and you flip those around so that if you see that word or phrase in an email, you have a certain probability that the email is real and a certain probability it is spam. For instance, if you see the phrase “free trial” in an email, there is a high probability it is spam, while if you see the word phrase “discrete math” there is a much lower probability the email is spam.

The spam filter works by considering all the words and phrases in the email, multiplying their probabilities all together, and comparing them to some preset spam level. The multiplication is necessary because of the rules of probability. Each of the probabilities is a value less than 1, and multiplying a bunch of those numbers together can give a very small result. For instance, if we have 1000 words, each with probability .1, the result is $.1^{1000}$ or 10^{-1000} . Most computers and calculators can’t easily handle numbers much less than about 10^{-300} , so the result of $.1^{1000}$ will be reported as 0.

The solution is to use logarithms. If the probabilities are p_1, p_2, \dots, p_n , instead of computing $p_1 p_2 \dots p_n$, we compute $\ln(p_1 p_2 \dots p_n)$, which we can rewrite as $\ln(p_1) + \ln(p_2) + \dots + \ln(p_n)$. Adding a bunch of small numbers will not cause the same problem as multiplying them. For instance, whereas $.1 \times .1 \times \dots \times .1$ (1000 times) gives 0 on most computers, $\ln(.1) + \ln(.1) + \dots + \ln(.1)$ gives -2302.

In general, using logs is a useful technique when you have to multiply a bunch of really small numbers.

7. **Finding a formula** — Consider the following table of values from an experiment.

x	y
1	0
10	12
100	24
1000	36
10000	48

Notice how the x values are going up by a multiplicative factor of 10, while the y values are going up by a constant (additive) factor of 12. A situation like this is indicative of a logarithmic relationship between x and y . Suppose we add another column to the table:

x	$\log_{10}(x)$	y
1	0	0
10	1	12
100	2	24
1000	3	36
10000	4	48

From this we see the relationship more clearly: $y = 12 \log_{10}(x)$.

In general, whenever a multiplicative change in the input corresponds to an additive change in the output, there is a logarithm involved.

8. **Binary search** — One of the most important algorithms in computer science is the *binary search*. It is used to find a specific item from an ordered list of items. For instance, suppose we are looking to see if the name “Gauss” appears in an alphabetical list of 1,000,000 names. Start by looking at the middle element; maybe it turns out to be “Mandelbrot.” We know therefore that “Gauss” must be in the first half of the list. We then

look at the middle element of the first half; maybe it's "Euler." Thus "Gauss" must be in the right half of the first half, i.e. in the range from element 250,000 to element 500,000. We then continue this process over and over, until we either find "Gauss" or run out of things to check. How many steps could this possibly take?

In other words, how many times can we cut the list in half until we can't go any further? Starting with 1,000,000 names to check, by the second step there are 500,000 to check, then 250,000, etc. So we are trying to solve $1000000 \times 1/2 \times 1/2 \cdots \times 1/2 = 1$, i.e., $1000000(1/2)^x = 1$, or $2^x = 1000000$. The solution is $x = \log_2(1000000)$, which is about 20 times.

So even with a list of 1,000,000 names, it takes no more than 20 steps to determine if a given name is in the list. If we had one billion names, it would take $\log_2(10^9) = 30$ steps. With one trillion names, it would take $\log_2(10^{12}) = 40$ steps. In general, with 10^n names, it would take $\log_2(10^n) = n \log_2(10) \approx 3.3n$ steps. This is a hallmark of logarithmic growth—a multiplicative increase in the input corresponds to a linear increase in the output.

In practical terms, this means that searching a sorted list of any reasonable size is (at least theoretically) a very quick thing to do. In general, logarithmic algorithms are highly desirable. Compare an algorithm that takes $\log_2(n)$ steps versus one that takes n steps or n^2 steps. At $n = 1,000,000$, the $\log_2(n)$ algorithm takes 20 steps, while the others take 1 million and 1 trillion steps, respectively.

9. **Logarithms and human senses** — Many of our senses operate on a logarithmic scale, including brightness, sound, and touch. In other words, a multiplicative change in the strength of the signal corresponds to a linear change in perception. For instance, we measure loudness with the Decibel scale. A sound of 110 decibels is not 10% louder than a sound of 100 decibels, but rather it is 10 times louder.

Research by Stanislas Dehaene on babies and on Amazon tribesman with no formal education appears to show that our natural number sense is logarithmic. The radio show *Radiolab* did a nice piece on this (season 6, episode 5). In it Dehaene describes how he asked the tribesmen what number was halfway between 1 and 9. Most people would say 5, but the tribesmen said 3. An answer of 5 comes from thinking of numbers linearly (or additively) as $1 + 4 = 5$ and $5 + 4 = 9$. The tribesmen's answer of 3 comes from thinking logarithmically (or multiplicatively) as $1 \times 3 = 3$ and $3 \times 3 = 9$. It is possible that our number sense is naturally logarithmic because much of our sensory experience is logarithmic.

10. **Benford's law** — One of the most remarkable facts about numbers is Benford's law, which states, roughly, that for data values spread across several orders of magnitude, there are more numbers that start with a 1 or 2 than start with 8 or 9. This runs counter to most people's intuition, where we would expect the starting digits to all be equally likely.

Here is an example to help explain this. Bank account amounts are spread across several orders of magnitude. Some people only have \$20 in the bank, while others have thousands or even millions. It is not too hard to for a bank account to go from \$80 to \$90 (i.e., to change the first digit from an 8 to a 9). That's just a 12.5% increase. But to go from \$100 to \$200 (i.e., change the first digit from a 1 to a 2) requires doubling your money (or a 100% increase). To go from \$800 to \$900 again requires a 12.5% increase, while to go from \$1000 to \$2000 again requires doubling your money. That is why numbers starting with 1 or 2 are so much more likely than numbers starting with higher digits.

It turns out that the probability that a number starts with digit d is given by $\log_{10}(1 + \frac{1}{d})$. Here is a table for all the values of d .

digit	probability (as %)
1	30.1
2	17.6
3	12.5
4	9.7
5	7.9
6	6.7
7	5.8
8	5.1
9	4.6

There is a 30.1% probability of starting with a 1, while only a 4.6% probability of starting with a 9. Starting with 1 is as likely as starting with 5, 6, 7, 8, or 9, combined.

Benford's law has been applied to molecular weights, baseball stats, census data, revenue figures, stock prices, street addresses, lengths of rivers, and many more things. It has been used in forensic accounting, which involves looking through accounting records for evidence of fraud. When people make up data (like expenses), they don't take into account Benford's law and are more likely to try and spread the starting digits around evenly.

One thing to remember is that Benford's law does not apply to all data sets. It works best if the data is spread across several orders of magnitude. For data that is constrained in a narrow range (like test scores, human heights, etc.), Benford's law might not apply.

The aforementioned *Radiolab* episode also has a segment about Benford's law.

11. **Logs and dice rolls** — Here is a somewhat surprising place that logarithms show up: Suppose we have five dice that we keep rolling until all of them are the same, saving dice between rolls that are the same, sort of like in the game *Yahtzee*.¹ How many rolls will it take on average before all five dice are the same? It turns out to take around 11 rolls.

Now what if we started with 10 dice or 100 dice, or even 1000 dice? How many rolls should we expect it to take until all the rolls come out the same? Here is a table of values from a computer simulation:

Number of dice	Number of rolls
10	15.3
100	28.6
1000	41.5
10000	54.2

Even with thousands of dice, the number of rolls is still relatively small. We see that a 10 times multiplicative increase in the number of dice corresponds to an additive increase in the number of rolls needed. This indicates that a logarithm is involved.

Why is this? At each roll, we expect roughly $1/6$ of the dice to be ones, $1/6$ to be twos, etc. So at each step, roughly $1/6$ of the dice will be the same as the one we are holding. We will take those dice out, leaving $5/6$ of the previous total left to roll. So if we start with D dice, after n rolls, there will be roughly $D(5/6)^n$ dice left. We want to know how many steps it takes until we are down to just one die, which means we are solving $D(5/6)^n = 1$. We solve this for n to get $n = \log_{6/5}(D)$.

Increasing D by a factor of 10 corresponds to an increase of

$$\log_{6/5}(10D) - \log_{6/5}(D) = \log_{6/5}(10) \approx 12.6 \text{ rolls,}$$

which corresponds to the constant increase of about 13 seen in the table above.

12. **Appearances of logs in math and science** — There are a number of places in math and science where logarithms show up.

- (a) The Richter scale is logarithmic. A magnitude 8 earthquake is not 33% stronger than a magnitude 6, but rather it is 1000 times stronger. The magnitude is related to the logarithm of the earthquake's power.
- (b) The Prime Number Theorem states that the number of primes less than n is roughly $\frac{n}{\ln(n)}$. In other words, the proportion of numbers less than n that are prime is roughly $\frac{1}{\ln(n)}$. For instance, the formula predicts that roughly 14.5% of numbers less than 1000 are prime and roughly 7.2% of numbers less than 1,000,000 are prime.
- (c) Hick's Law — The amount of time it takes to choose from n choices is proportional to $\log_2(n + 1)$.

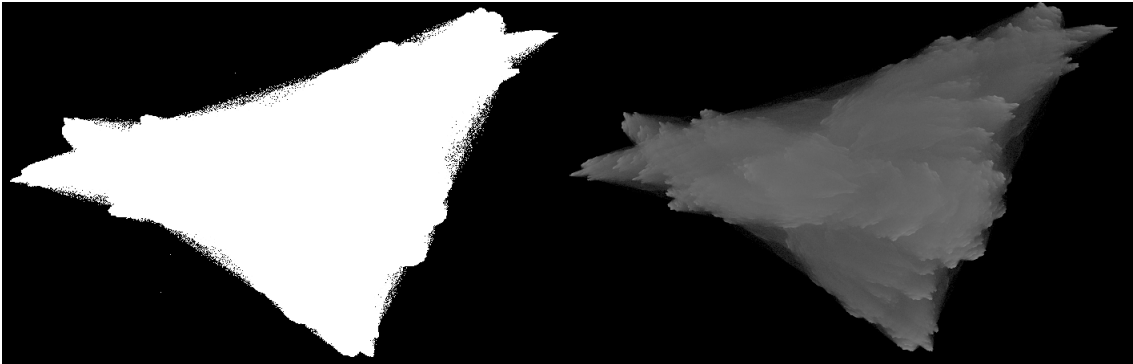
¹For instance, suppose our first roll gives us two 1s, a 3, a 4, a 6. We keep the two 1s off to the side and reroll the other three dice. Suppose we get a 1 and two 5s. We put the 1 off to the side and reroll the other two dice. We keep this up until we get five 1s.

- (d) Fitt's law — The time to point to something (with a finger or a mouse, for instance) is proportional to $\log_2\left(1 + \frac{D}{W}\right)$, where D is the distance to the target and W is the size of the target.
- (e) Sturges' rule — When choosing how many bins to break data up into when drawing a histogram with n data values, Sturges' rule says to break it up into equally sized bins of width $1 + \log_2(n)$.
- (f) In the Mercator map projection, which is the usual way the continents from our spherical planet are drawn on a flat map, the relationship between map distance D and latitude ϕ is given by $D = a \ln(\tan(\frac{\phi}{2} + 45^\circ))$. For instance, you might have noticed on many maps that Greenland and Alaska appear much bigger than they are. The formula above bears that out.
- (g) If you are randomly choosing items from a set of size N , the number of items you have to choose before there is a probability p of a repeat is roughly

$$\sqrt{2N \ln\left(\frac{1}{1-p}\right)}.$$

For example, if we have a library of 5000 songs from which we randomly select songs, how many songs would we have to listen to before we have a 25% chance of hearing the same song twice? Plugging $N = 5000$ into the formula gives just 53 songs.

13. **Iterated function systems** — The figures below show two different plots of a something called an iterated function system.



We won't try to give a definition of what an iterated function system is, but the important thing for this discussion is it is a plot of thousands of points, some of which are hit lots of times. The figure above on the left colors a point white if it is hit once and black if it is never hit. The figure on the right colors it a shade of gray according to how many times it is hit, giving a much more interesting picture.

It turns out that some points are hit just a few times while others are hit thousands of times. A linear coloring, say where we a point hit twice as often is colored twice as bright won't work because there is too great a range of values. With some points hit 10,000 times, others hit 1000 times, still others hit 100 or 10 times, any linear map of the following form will not be effective:

$$\text{brightness} = K(\# \text{ times hit}).$$

However, something of the following form works quite well:

$$\text{brightness} = K \log(\# \text{ times hit})$$

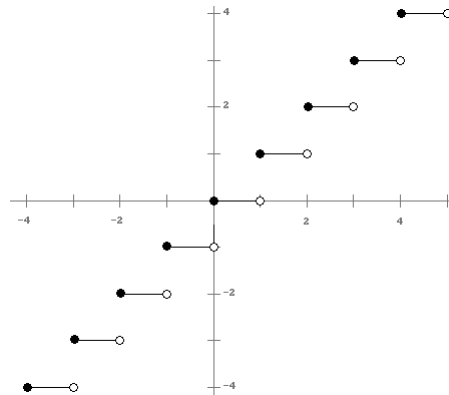
The logarithm reduces a range like 10 to 1,000,000 down to a range from 1 to 6, which can then be scaled to give a nice range of brightness.

3.11 Floors and ceilings

The *floor* of a real number x , denoted $\lfloor x \rfloor$, is the greatest integer less than or equal to x . For instance, we have

$$\begin{aligned}\lfloor 3.14 \rfloor &= 3 \\ \lfloor 4.99 \rfloor &= 4 \\ \lfloor 6 \rfloor &= 6 \\ \lfloor -3.14 \rfloor &= -4.\end{aligned}$$

For positive numbers, the floor just drops the decimal part. That rule doesn't quite work for negatives. Here is the graph of the floor function:



The floor function is also sometimes called the *greatest integer function*. Some authors use the notations $\lfloor x \rfloor$ or $[[x]]$ for the floor.

There is the related notion of the *ceiling* of x , denoted $\lceil x \rceil$. It is the least integer greater than or equal to x . For instance,

$$\begin{aligned}\lceil 3.14 \rceil &= 4 \\ \lceil 4.99 \rceil &= 5 \\ \lceil 6 \rceil &= 6 \\ \lceil -3.14 \rceil &= -3.\end{aligned}$$

In general, the floor always rounds down, and the ceiling always rounds up.

The floor and ceiling functions are important in math and programming and are built into most programming languages, usually in the math library. In addition, many programming languages use *integer division*. For example in Java, if you do $7/4$, the result is actually 1, not 1.75. To Java, 7 and 4 are both integers, so the result of the operation must be an integer as well. Java drops the decimal part so that x/y in Java for integers x and y is actually $\lfloor x/y \rfloor$. In Python 3.0 and later, $7/4$ will produce 1.75, and $7//4$ will do integer division to produce 1.

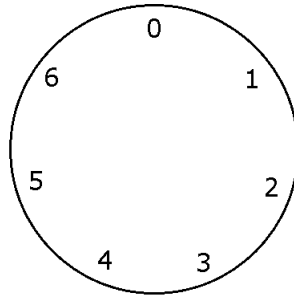
Here are a few examples of floors and ceilings:

1. Suppose on a quiz, you get 1 point for every 3 correct answers. Your total score is given by $\lfloor x/3 \rfloor$, where x is the number of correct answers.
2. Suppose we want a function describing how many leap years there have been since the year 2000. Leap years this century are 2000, 2004, 2008, 2012, \dots , coming every four years. If y is the current year, then the number of leap years is $\lceil (y - 2000)/4 \rceil$.
3. Floors are often useful for describing unusual patterns. For instance, for $n = 0, 1, 2, \dots$, the expression $\lfloor 2n/3 \rfloor$ generates the pattern 0, 0, 1, 2, 2, 3, 4, 4, 5, 6, 6, \dots . The *Online Encyclopedia of Integer Sequences* (OEIS) is useful for things like this, if you have a sequence of values and you want to find a formula for it. The OEIS also has information about places that the sequence shows up.

3.12 Modular arithmetic

Modular arithmetic is a kind of “wrap around” arithmetic, like arithmetic with time. In 24-hour time, after 23:59, we wrap back around to the start 00:00. After the 7th day of the week (Saturday), we wrap back around to the start (Sunday). After the 365th or 366th day of the year, we wrap back around to the first day of the year. Many things in math and real-life are cyclical and a special kind of math, known as modular arithmetic, is used to model these situations.

Let’s look at some examples of arithmetic modulo (mod) 7, where we use the integers 0 through 6. We can think of the integers as arranged on a circle, like below:



We have the following:

1. 7 is the same as 0, 8 is the same as 1, 9 is the same as 2, etc.
2. In general, any multiple of 7 is the same as 0, any number of the form $7k + 1$ is the same as 1, any number of the form $7k + 2$ is the same as 2, etc.
3. $4 + 5$ is the same as 2. Adding 5 corresponds to moving around the circle 5 units clockwise.
4. $4 - 5$ is the same as 6. Subtracting 5 corresponds to moving 5 units counterclockwise.
5. $4 + 21$ is the same as 4. Adding 21 corresponds to going around the circle 3 times and ending up where you started.

In general, the notation $a \bmod b$ refers to the value a reduces to when working modulo b . It is the remainder when a is divided by b . One way to compute $a \bmod b$ is to find the closest multiple of b less than a and subtract it from a . For instance, to find $68 \bmod 7$, the closest multiple of 7 less than 68 is 63, and $68 - 63 = 5$, so $68 \bmod 7 = 5$.

We can represent this process with an equation as

$$a \bmod b = a - \left\lfloor \frac{a}{b} \right\rfloor b.$$

For example, to compute $422103 \bmod 101$, we divide $422103/101$ and drop the fractional part to get 4179. Then $422103 - 4179 \cdot 101 = 24$ is our answer.

This procedure applies to negatives as well. For instance, to compute $-31 \bmod 5$, the closest multiple of 5 less than or equal to -31 is -35, which is 4 away from -31, so $-31 \bmod 5 = 4$, or $-31 \equiv 4 \pmod{5}$.

As one further example, suppose we want $179 \bmod 18$. 179 is one less than 180, a multiple of 18, so it leaves a remainder of $18 - 1 = 17$. So $179 \bmod 18 = 17$.

Here are some examples of modular arithmetic:

1. **Programming** — Most programming languages have the modulo operation built in, usually using either the operator `%` or `mod`.

2. **Last digits** — To get the last digit of an integer, mod it by 10. For instance, the last digit of 456 is $456 \bmod 10 = 6$. The closest multiple of 10 to 456 is 450, and subtracting that away just leaves the last digit.

To get the last two digits, mod by 100, to get the last three, mod by 1000, and in general, to get the last n digits mod by 10^n . This technique is sometimes useful in computer programs if you need the last digits of a number. It is also useful in number theory.

3. **Checking for divisibility** — The modulo operation is used to test for divisibility, especially in computer programs. In general, n is divisible by k if and only if $n \bmod k = 0$ (in other words, if n/k leaves a remainder of 0). For example, n is even (divisible by 2) whenever $n \bmod 2$ is 0 and odd whenever $n \bmod 2$ is 1.

4. **Conversions** — Suppose we need to convert 210 seconds to minutes and seconds. The answer, 3 minutes and 30 seconds, can be gotten using the formulas below. If S is an amount in seconds, then we can write it as m minutes and s seconds by

$$m = \left\lfloor \frac{S}{60} \right\rfloor$$

$$s = S \bmod 60.$$

Similar ideas can be used to convert a height in inches to feet and inches or a weight in ounces to pounds and ounces.

5. **Converting between 2d and 1d arrays** — A similar type of conversion to the minutes and seconds conversion is useful for converting between two ways of representing a grid. We can represent the cells of the grid using a two-dimensional row/column coordinate system or one-dimensionally, giving each cell an index typewriter-style, as shown in the figure below.

(0,0)	0	(0,1)	1	(0,2)	2	(0,3)	3	(0,4)	4	(0,5)	5	(0,6)	6
(1,0)	7	(1,1)	8	(1,2)	9	(1,3)	10	(1,4)	11	(1,5)	12	(1,6)	13
(2,0)	14	(2,1)	15	(2,2)	16	(2,3)	17	(2,4)	18	(2,5)	19	(2,6)	20
(3,0)	21	(3,1)	22	(3,2)	23	(3,3)	24	(3,4)	25	(3,5)	26	(3,6)	27

It is useful to be able to convert between the two ways of representing the grid, especially in computer programming. A two-dimensional array is the natural way to work with this grid, but sometimes it is more convenient to work with a one-dimensional array. Here are the conversions between the two-dimensional representation (r, c) and a one-dimensional index i :

$$r = \left\lfloor \frac{i}{7} \right\rfloor \qquad i = 6r + c$$

$$c = i \bmod 7.$$

In terms of programming syntax, if our one-dimensional array is a and our two-dimensional array is b , to go from $b[r][c]$ to $a[i]$ we use $i = 6*r + c$ and to go from $b[r][c]$ to $a[i]$ use $r = i / 7$ and $c = i \% 7$.¹ This works for a grid with 7 columns. For a different number of columns, just replace the sevens with the number of columns.

6. **A wrap-around calculation** — One other place in programming that mods show up is if you need things to “wrap around.” For instance, suppose you have a 5-player game where the players go in order as if they were sitting around a table, with the first player going again after the last player is done. Assume the players are numbered 0 through 4. If p denotes the current player, then $(p + 1) \bmod 5$ can be used to do this wrapping around.

For example, if the current player is #2, then $(p + 1) \bmod 5$ will be $3 \bmod 5 = 3$. However, if the current player is #4, then $(p + 1) \bmod 5$ will be $5 \bmod 5 = 0$.²

7. **Another wrap-around calculation** — Suppose we have a list of 365 average temperatures, one for each day of the year, and we want to know when the largest 30-day change in temperatures happens.

¹In Python 3.0 and later, replace $r = i / 7$ with $r = i // 7$.

²This assumes the players are named player 0 through player 4. We would have to add 1 when printing out the name of the player to make the names go from player 1 to player 5.

In a computer program, if our list is called a , we might try to have an index i run through all the possible days and compute the absolute value of $a[i+30]-a[i]$.

The problem is that since the index i runs from 0 to 365, what happens if the largest change occurs between December 22 and January 21 (from index 355 to index 20)? Mods give us an easy way to solve this problem. We just compute the absolute value of $a[(i+30) \% 365] - a[i]$. Modding by 365 will wrap the indices around so that, for instance, 30 days after index 355 is 385, and $385 \bmod 365 = 20$ (i.e., 30 days after December 22 is January 21).

8. **Congruences** — Mathematicians use the notation $a \equiv c \pmod{b}$ to denote the modular relationship. We read the notation as “ a is congruent to b modulo (or mod) n .” When we say this, we mean that both a and c leave the same remainder when divided by b , or equivalently that $a - c$ is divisible by b . For example, we have $13 \equiv 6 \pmod{7}$ and $27 \equiv 13 \pmod{7}$.

Working with congruences turns out to be a lot like working with ordinary equations, and an entire algebra can be developed around it. This turns out to be very important in higher math.

9. **Calendar calculations** — Modular arithmetic can be used to find the day of the week of any date. For example, here is how to compute the date of Christmas is any given year Y :

$$a = \left\lfloor \frac{Y}{100} \right\rfloor \bmod 4 \qquad b = Y \bmod 100 \qquad c = \left\lfloor \frac{b}{4} \right\rfloor \bmod 7$$

Compute $b + c - 2a + 1 \bmod 7$. A 0 corresponds to Sunday, 1 to Monday, etc.

For example, if $y = 1977$, then $a = 19 \bmod 4 = 3$, $b = 77$, and $c = \lfloor 77/4 \rfloor \bmod 7 = 5$. Then we get $b + c - 2a + 1 = 0$, so Christmas was on a Sunday in 1977.

A more general process can be used to find the day of the week of any date in history. See *Secrets of Mental Math* by Art Benjamin and Michael Shermer.

10. **Check digits** — Many codes, like UPC codes, ISBN numbers of books, and credit card numbers use something called a check digit to catch errors from mistyped or mis-scanned codes.

For example, 10-digit ISBN numbers use the following scheme: Let x_1, x_2, \dots, x_{10} be the digits and compute $\left(\sum_{i=1}^9 i \cdot x_i\right) \bmod 11$. The result should equal the check digit, x_{10} . If not, then there was an error with the number.

Suppose we have the ISBN 0140275363. Start by computing $(1 \cdot 0 + 2 \cdot 1 + 3 \cdot 4 + 4 \cdot 0 + 5 \cdot 2 + 6 \cdot 7 + 7 \cdot 5 + 8 \cdot 3 + 9 \cdot 6) \bmod 11$. One shortcut we can use is to reduce each individual term mod 11. Doing this gives $0 + 2 + 1 + 0 + 10 + 9 + 2 + 2 + 10 = 36$, and $36 \bmod 11 = 3$, which matches with the last digit of the ISBN.

This check-digit scheme is able to catch single-digit errors and transposition errors (where two digits are accidentally swapped). However, if there are multiple errors, this scheme will usually, but not always catch them, as their effects might cancel out.

11. **Cryptography** Modular arithmetic is a critical part of modern cryptography, including the RSA algorithm and Diffie-Hellman key exchange. Both of these algorithms involve raising numbers to large powers and modding the results by a large prime. The security of Diffie-Hellman, for example, rests on the fact that it appears to be very difficult to solve the equation $a^x \bmod p = b$ for x , with a , b , and p known, if p is a very large prime (on the order of several hundred digits long).

12. **Generating random numbers** — One other important application of modular arithmetic is in generating random numbers. Random numbers are important in many aspects of computer programming. For instance, in a video game we might want some aspects of randomness so the game isn't always the same. Random numbers are also important in computer simulations, where a computer is used to simulate something like climate, traffic, or the spread of a disease.

One approach to generating random numbers is to use some physical process, like noise in a circuit, atmospheric processes, or radioactive decay. The main problem with this is that it can be slow. Instead, most computers use a mathematical process to generate random numbers. One of the most widely used approaches is the *linear congruential generator* (LCG), which uses modular arithmetic.

The basic idea is simple: choose values for a , b , and m in the equation $(ax + b) \bmod m$, choose a starting value for x (called the *seed*), and keep plugging the newly generated values of x into the formula. For example, suppose we use $a = 2$, $b = 7$, $m = 11$, and start with $x = 1$. Our formula is $(2x + 7) \bmod 11$.

Plugging $x = 1$ into the formula gives $(2 \cdot 1 + 7) \bmod 11 = 9$. Then plugging $x = 9$ into the formula gives $(2 \cdot 9 + 7) \bmod 11 = 3$. We then plug $x = 3$ into the formula and continue, the next few numbers generated being 2, 0, 7, 10, 5.

Choosing $m = 11$ means that our sequence of values will start repeating no later than the 11th term. Having our random numbers repeat this quickly is obviously bad, so in practice, much larger values of m are used. Care also has to be taken in choosing a and b in order for the numbers to be as random as possible. The C programming language uses an LCG with $a = 1,103,515,245$, $b = 12345$, and $m = 2^{32}$. Java uses $a = 25,214,903,917$, $b = 11$, and $m = 2^{48}$. Though LCGs are still used in a number of important programming languages, there are better approaches available now. For instance, Python uses something called the Mersenne Twister.

It is worth noting that we are using a purely mathematical approach to generate the random numbers, so they are not truly random in that anyone that knows the formula and parameters can also generate those numbers. In fact, random numbers generated by a process like this are called *pseudorandom numbers* in that they appear random, but really aren't. Pseudorandom numbers are fine for most applications, but one critical exception is cryptography. In that case, you don't want people being able to figure out the random numbers you used, so you have to be careful how you generate your numbers.

Chapter 4

Recursion and Induction

Recursion and induction are two related parts of math that involve building up new things from previous cases. Recursion is particularly useful as a way to design algorithms in computer science, while induction is one of the most important mathematical techniques for proving things. Both topics are also ones that people tend to find difficult. But it is possible to understand them with some practice.

4.1 Recursion

One of the most famous sequences of numbers in mathematics is the Fibonacci sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... The first two Fibonacci numbers are 1, and each term thereafter is the sum of the two preceding. We use the notation F_1, F_2, F_3, \dots to denote the Fibonacci numbers. Using subscripts in this way is very common in math.

The rules for the Fibonacci sequence can be written as

$$F_1 = 1, \quad F_2 = 1, \quad F_{n+1} = F_n + F_{n-1} \text{ for } n \geq 2.$$

The equation $F_{n+1} = F_n + F_{n-1}$ says that to get the next Fibonacci number, F_{n+1} , we add the two previous Fibonacci numbers, $F_n + F_{n-1}$. For example, plugging in $n = 2$ gives $F_3 = F_2 + F_1$ and plugging in $n = 3$ gives $F_4 = F_3 + F_2$.¹

A sequence like the Fibonacci numbers, where new terms are gotten from previous ones, is called *recursive* (or a *recurrence relation*). There are many things in mathematics that are defined recursively. Sometimes, a recursive definition is much easier to give than a non-recursive definition. For instance, the Fibonacci numbers have a non-recursive definition that is somewhat more complicated than the recursive one.² Here are some examples of recursive sequences:

1. **Powers** — Consider the sequence defined by $x_1 = 1$ and $x_{n+1} = 2x_n$ for $n \geq 1$. The first few terms of this sequence are 1, 2, 4, 8, 16, ... We see that these are just powers of two, so a non-recursive definition is $x_n = 2^n$.
2. **Finding a formula** — Suppose we are given the sequence 4, 8, 12, 16, 20, ... Can we find a recursive definition? To do so, we look at the relation between the terms. We see that each term is 4 more than the previous term, so our definition is $x_1 = 4$ and $x_{n+1} = 4 + x_n$ for $n \geq 1$. Note that it is important to have a definition for the initial term x_1 , as we need to know where the sequence starts.
3. **Interest** — Suppose we have a \$1000 account that gets 3% interest, compounded once a year, and we draw \$100 from it each year. We can create a recursive sequence x_n to describe the amount of money in the account after n years. We start with \$1000 in the account, so we have $x_0 = 1000$. Then we have $x_{n+1} = 1.03x_n - 100$ for $n \geq 0$. That is, the amount of money in the account next year is gotten by adding the current amount plus the interest and subtracting off \$100.

¹The definition given above is not the only way to do things. We could also use $F_{n+2} = F_{n+1} + F_n$ for $n \geq 1$ or $F_n = F_{n-1} + F_{n-2}$ for $n \geq 3$.

²It is called Binet's formula and is $F_n = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n \sqrt{5}}$.

4. **Factorials** — The *factorial* of a nonnegative integer n , denoted $n!$, is the product $n(n-1)(n-2)\cdots 3\cdot 2\cdot 1$. For example, $4! = 4\cdot 3\cdot 2\cdot 1 = 24$ and $5! = 5\cdot 4\cdot 3\cdot 2\cdot 1 = 120$. For reasons that will become clear a little later, $0!$ is defined to be 1.

We can describe factorials by the recursive sequence $f_0 = 1$ and $f_{n+1} = (n+1)f_n$. For instance, with $n = 5$, this says $f_5 = 5f_4$ or $5! = 5\cdot 4!$. We can see this because $4!$ is contained inside of $5! = 5\cdot(4\cdot 3\cdot 2\cdot 1)$.

5. **Permutations** — In math, it is often useful to know how many ways there are to rearrange things. For example, we can rearrange the string 123 as $123, 132, 213, 231, 312$, and 321 , six ways in total. Rearrangements are also called *permutations*. To find a general recursive formula for permutations, consider how we would go from permutations of 123 to permutations of 1234 . Starting with 123 , we can insert a 4 in any of four positions to get $4123, 1423, 1243, 1234$. Then with 213 , we insert a 4 to get $4213, 2413, 2143, 2134$. For each of the six permutations of 123 , we can insert a 4 in any of four locations, so there are $6 \times 4 = 24$ permutations of 1234 .

To go from permutations of 1234 to 12345 , there are five places to insert a 5 in each of the twenty-four permutations of 1234 , giving us $5 \times 24 = 120$ permutations of 12345 . In general, if P_n is the number of permutations of $12\cdots n$ then $P_{n+1} = (n+1)P_n$. This is the same as the factorial recursion we saw above. We also have $P_1 = 1$ as there is one way to rearrange the string 1 (which is to leave it alone), and $P_0 = 1$ since there is technically one way to rearrange the empty string (which is to leave it alone).¹ So we have $P_n = n!$.

6. **Pascal's triangle** — One of the most well known objects in mathematics is Pascal's triangle; the first few rows are shown below:

				1			
				1	1		
			1	2	1		
		1	3	3	1		
	1	4	6	4	1		
1	5	10	10	5	1		
1	6	15	20	15	6	1	

Each entry is the sum of the two directly above it. For instance, the 20 in the bottom row is the sum of the two 10s directly above it. We can describe Pascal's triangle recursively. Let $C_{r,c}$ denote the entry in row r , column c , where we start counting at 0. The top entry of the triangle is $C_{0,0}$. The two entries below it are $C_{1,0}$ and $C_{1,1}$. In general, the outside edges are given by $C_{r,0} = 1$ and $C_{r,r} = 1$ for any $r \geq 0$. Since each entry is the sum of the two above it, we have $C_{r,c} = C_{r-1,c-1} + C_{r-1,c}$ for any $r \geq 1$ and $1 \leq c \leq r-1$.²

7. **Number of subsets** — In this example we will find a recursive description of the number of k -element subsets of an n -element set. For brevity, we will use the shorthand 123 for a set like $\{1, 2, 3\}$. We can find a recursive description by looking at a small example.

The three-element subsets of 12345 are $123, 124, 134, 234, 125, 135, 235, 145, 245, 345$. Break these up into the ones that contain 5 ($125, 135, 235, 145, 245, 345$) and the ones that don't ($123, 124, 134, 234$).

Notice that the three-element subsets of 12345 that don't contain a 5 are the three-element subsets of 1234 . The three-element subsets of 12345 that do contain a 5 are the two-element subsets of 1234 with a 5 added on. So if we let $x_{n,k}$ denote the number of k -element subsets of $\{1, 2, \dots, n\}$, then we have just shown that $x_{5,4} = x_{5,3} + x_{5,2}$.

In general, this will always work. We get the k -element subsets of $\{1, 2, \dots, n\}$ from the k -element subsets of $\{1, 2, \dots, n-1\}$ along with the $k-1$ -element subsets of $\{1, 2, \dots, n-1\}$ with an n added. So we have

$$x_{n,k} = x_{n-1,k} + x_{n-1,k-1}.$$

Also, there is only one 0-element subset of a set (the empty set) and only one subset the same size as the set (the set itself), so $x_{n,0} = 1$ and $x_{n,n} = 1$ for every $n \geq 0$.

This recurrence is the same as the one for Pascal's triangle. So the entries in Pascal's triangle give the number of subsets of a set. For instance, to get the number of 3-element subsets of $\{1, 2, 3, 4, 5, 6\}$ we go to row 6, column 3 (starting counting at 0) to get 20.

¹This is where we get $0! = 1$.

²We will see later on that there is a nonrecursive definition for $C_{r-1,c-1}$. It is usually denoted $\binom{r}{c}$ and is given by $\frac{r!}{c!(r-c)!}$.

8. **Increasing sequences** — In this example, we will find a recursive answer to how many strings of integers start with 1, end with n and are strictly increasing. First, let's look at some examples to understand the problem.

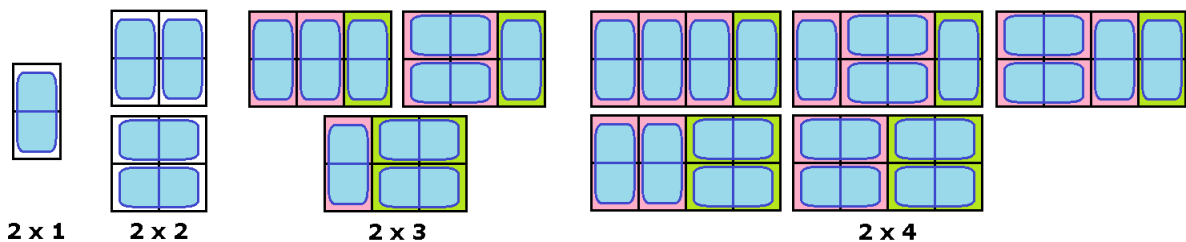
$n = 1: 1$
 $n = 2: 12$
 $n = 3: 13, 123$
 $n = 4: 14, 124, 134, 1234$
 $n = 5: 15, 125, 135, 145, 1235, 1245, 1345, 12345$

Looking at how we get the strings for $n = 5$, they come from each of the strings for $n = 1, 2, 3$, and 4 , with a 5 added to the end. That is, $x_5 = x_4 + x_3 + x_2 + x_1$. This works in general, so if x_n denotes the number of strings we are looking for, then

$$x_n = x_{n-1} + x_{n-2} + \cdots + x_1.$$

Noticing that $x_1 = 1$, $x_2 = 1$, $x_3 = 2$, $x_4 = 4$, and $x_5 = 8$, it seems that for $n \geq 2$ we have $x_n = 2^{n-2}$. That turns out to be the case and can be shown using the geometric series formula.

9. **Tiling a grid with dominoes** — For this example, we want the number of ways to tile a $2 \times n$ grid with 2×2 dominoes. We want to completely fill the grid with dominoes. We will use t_n to denote the number of ways to tile the $2 \times n$ grid with dominoes. Here are all possibilities up to 2×4 .



For $n = 3$ and beyond, we can compute t_n from t_{n-1} and t_{n-2} as follows: All the tilings for the $2 \times n$ case come from a tiling of the $2 \times (n-1)$ grid with the 2×1 tiling at the end or from a tiling of the $2 \times (n-2)$ grid with either of the two 2×2 tilings at the end. Thus $t_{n+1} = t_n + t_{n-1}$. We see this is the same as the Fibonacci recurrence. We end up with $t_n = F_{n-1}$.

10. **Derangements** — A *derangement* of a sequence is a rearrangement of the elements such that no element stays fixed. For instance, 21453 is a derangement of 12345 , but 32541 is not because 2 stays fixed. Each element needs to end up in a different location. Let d_n denote the number of derangements of $12 \cdots n$. We have $d_1 = 0$. Shown below are all the derangements of $12 \cdots n$ for $n = 2, 3$, and 4 :

$n = 2: 21$
 $n = 3: 312, 231$
 $n = 4: 2143, 2341, 2413, 3412, 3142, 3421, 4321, 4123, 4312$

Let's consider how we would find derangements of 12345 . Consider those that start with a 2. They are either of the form $21 _ _ _$ or of the form $2 _ _ _ _$ where the second entry is not a 1. In the first case, the last three digits must be a derangement of three items, and there are d_3 of those. In the second case, we know that 1 can't be in the second position, so the last four digits must also be a derangement, and there are d_4 of those.

The other starting values for our derangements are 3, 4, and 5, and a similar argument applies to them. So in total there are $4(d_3 + d_4)$ derangements of 12345 . In general, we have $d_{n+1} = n(d_{n-1} + d_n)$.

An interesting question to consider what percentage of permutations are derangements. As $n \rightarrow \infty$, the percentage approaches $1/e$, perhaps an unexpected occurrence of Euler's constant e .¹

11. **The towers of Hanoi** — The Towers of Hanoi is a classic problem. There are three pegs, and the one on the left has several rings of varying sizes on it. The goal is to move all the rings from that peg to the peg on the right. Only one ring may be moved at a time, and a larger ring can never be placed on a smaller ring. See the figure below.

¹A related interesting fact is $d_n = \lfloor n!/e \rfloor$ for $n \geq 1$.

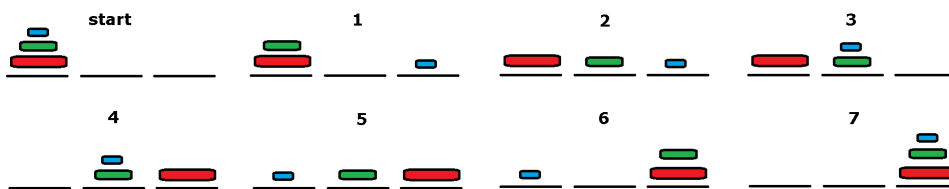


At this point, it would be good to stop reading and get out four objects of varying sizes and try the problem. Rings and pegs are not needed—any four objects of different sizes will do. The minimum number of moves needed is 15. It's also worth trying with just three objects. In that case, the minimum number of moves is 7.

We will solve the problem using recursion. The key idea is that the solution to the problem builds on the solution with one ring less. To start, here is the optimal solution with two rings:



The solution takes three moves and is pretty obvious. Shown below is the optimal seven-move solution with three rings.



Looking carefully, we can see that the solution for two rings is used to solve the three-ring problem. Steps 1, 2, and 3 are the two-ring solution for moving two rings from the left to the center. Steps 5, 6, and 7 are the two-ring solution for moving two rings from the center to the right. And overall, the three-ring solution itself is structured like the two-ring solution. Let's compare the solutions.

The two-ring solution can be described thusly:

Move the top ring to the center, move the bottom ring to the right, move the top ring to the right.

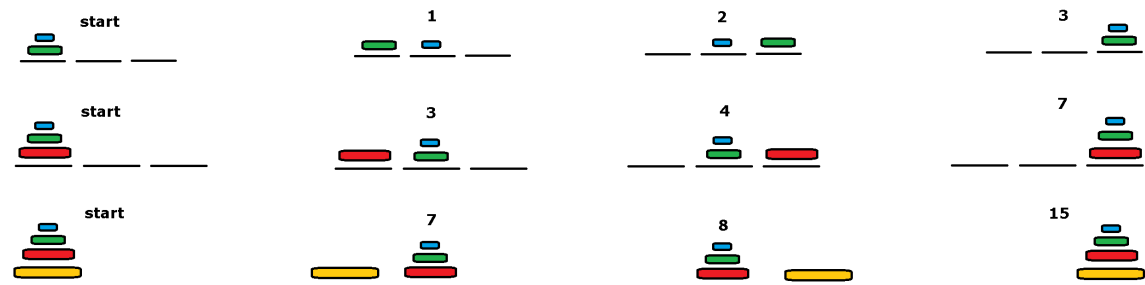
The three-ring solution can be described in a similar way:

Move the top *two* rings to the center, move the bottom ring to the right, move the top *two* rings to the right.

And the same can be said about the four-ring solution:

Move the top *three* rings to the center, move the bottom ring to the right, move the top *three* rings to the right.

And to move the top three rings, we use the three-ring solution. Shown below are parts of the two-, three-, and four-ring solutions, lined up, showing how they are similar.

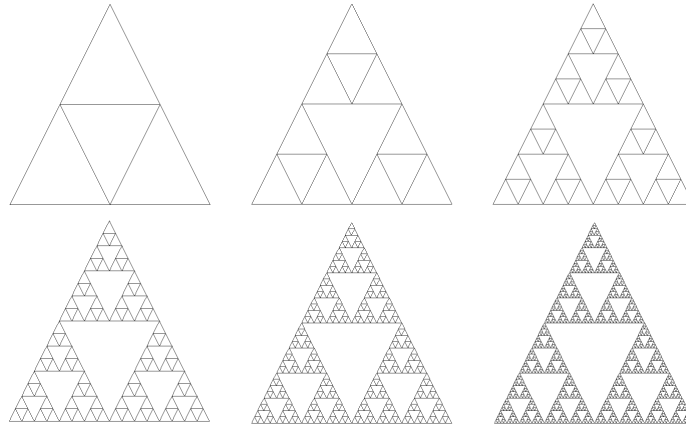


The general solution is a similar combination of the $n - 1$ solution and the two-ring solution. We use the $n - 1$ solution to move the top $n - 1$ rings to the center, then move the bottom ring to the right, and finally use the $n - 1$ solution to move the top three rings to the right.

It's not too hard to count the number of moves needed in general. The solution for two rings requires 3 moves. The solution for three rings requires two applications of the two-ring solution plus one more move,

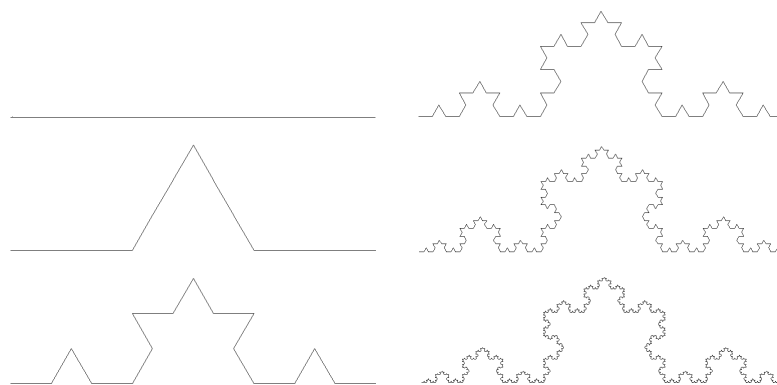
for a total of $2 \cdot 3 + 1 = 7$ moves. Similarly, the solution for four rings requires two applications of the three-ring solution plus one additional move, for $2 \cdot 7 + 1 = 15$ moves. In general, if h_n is the number of moves in the optimal solution with n rings, then we have $h_n = 2h_{n-1} + 1$. And in general, $h_n = 2^n - 1$ moves. This is exponential growth. Already, the solution for 10 pegs requires $2^{10} - 1 = 1023$ moves. This is a lot, but doable by hand in under an hour. The solution for 20 pegs requires over a million moves, and the solution for 30 pegs requires over a billion moves. The solution for 100 pegs requires roughly 10^{30} moves, which just might be able to be completed if every computer in existence worked round the clock for millions of years on the problem.

12. **Fractals** — The *Sierpinski triangle* is an object formed as follows: Start with an equilateral triangle and cut out a triangle from the middle such that it creates three equal triangles like in first part of the figure below. Then for each of those three triangle do the same thing. This gives the top middle object in the figure below. Then for each of the nine (upright) triangles created from the previous step, do the same thing. Keep repeating this process.



The same procedure that we apply at the start is repeatedly applied to all the smaller triangles. So we see that the Sierpinski triangle is defined recursively. Many interesting mathematical objects are defined recursively.

Another example is the *Koch curve*. Start with a line segment. Take the middle third of the segment and add a 60° kink to it. This creates four smaller equal length line segments. Add 60° kinks to them in the same way to create 16 smaller line segments. Keep repeating this process. The result is of the first six steps is shown below.



The recursion here is that the same process that was applied at the first step is applied at each step thereafter, just on a smaller scale. If we could continue this process indefinitely, the result an infinitely complex object called a *fractal*. Fractals have the property that if you look at any part of them, those parts look like the whole object. This is called *self-similarity*.

Techniques like this can be used to generate a many interesting objects. In particular, they are used to generate realistic-looking landscapes for movies and video games. A web search for *fractals* and *fractal landscapes* will turn up a variety of stunning images.

Many real-life objects have fractal-like characteristics, including plants, landscapes, lightning, ice crystals, and stock market behavior.

From the examples above, we see that recursion is all about building something up from smaller cases of itself. Recursion is particularly useful in counting problems, in certain processes like the fractal examples above, and in designing computer algorithms.

4.2 Induction

The principle of mathematical induction is recursion's stuffier, more formal cousin. It is a technique for proving things that builds up a proof by using previous cases to develop the next case. Here is an example of an induction proof. We will explain what exactly it is doing in a minute. For now, just read through the proof.

Prove that $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$ **for** $n \geq 1$.

Base case: The base case here is $n = 1$. In that case the left side of the equation is 1 (just a single term being added), while the right side is $1(1 + 1)/2 = 1$. So the equation works for $n = 1$.

Induction step: Assume that $1 + 2 + \cdots + n = \frac{n(n+1)}{2}$. We must show that $1 + 2 + \cdots + (n + 1) = \frac{(n+1)(n+2)}{2}$. We have

$$1 + 2 + \cdots + (n + 1) = (1 + 2 + \cdots + n) + (n + 1) = \frac{n(n + 1)}{2} + (n + 1) = (n + 1) \left(\frac{n}{2} + 1 \right) = \frac{(n + 1)(n + 2)}{2}.$$

Thus the result is true by induction. □

This might seem a little bizarre. After all, it seems like we are assuming the very thing we are trying to prove in the induction step. The idea of the induction is we have a base case to get started and then, just like recursion, we show how to get from one step to the next. In this case, we show how to go from step n to step $n + 1$.

The base case shows that the formula works for $n = 1$. The induction step tells us that if the formula works for n , then it also works for $n + 1$. Thus, since the formula works for $n = 1$ it must work for $n = 2$. Since it works for $n = 2$, it must also work for $n = 3$. Since it works for $n = 3$, it must also work for $n = 4$. This continues forever, showing that the formula works for any $n \geq 1$.

People often describe an induction proof as a line of dominoes, each one knocking the next down. The base case corresponds to flicking the first domino to get the cascade started and the induction step tells us that if one domino falls, it will knock the next one down.

Induction proofs will usually work as follows:

1. *Base case* — Show that the result works for the smallest value of n . This value is usually given in the problem or is the smallest value that makes sense. The base case is often (but not always) quick and simple to show.
2. *Induction step* — Assume the result holds for n . Show it works for $n + 1$. A common mistake is to forget to use what you are assuming. But that is the key to induction proofs—you have to show how to get from the smaller case to the bigger case, so you *must* use your assumption here.

Here are some more examples of induction.

1. **Prove that** $n! > 2^n$ **for** $n > 3$.

Base case: The smallest value to check, according to the statement, is $n = 4$. We have $4! > 2^4$ since $24 > 16$.

Induction step: Assume $n! > 2^n$. We have to show that $(n + 1)! > 2^{n+1}$. We have

$$(n + 1)! = (n + 1)n! > (n + 1)2^n > (1 + 1)2^n = 2^{n+1}.$$

Thus the result is true by induction. \square

The approach used above is break $(n + 1)!$ down into $(n + 1)n!$. The idea is to get an $n!$ into things so that we can apply the induction hypothesis, $n! > 2^n$. After using the induction hypothesis, we are at $(n + 1)! > (n + 1)2^n$, but our goal is to get $(n + 1)! > 2^{n+1}$. Working backwards from 2^{n+1} , we can break it down into $2 \cdot 2^n$. So if we could just show $(n + 1)2^n > 2 \cdot 2^n$, we would be done. The key to showing this is that since $n > 3$ we have $n > 1$ and so $(n + 1) > (1 + 1) = 2$.

2. **Prove that $8^n - 5^n$ is divisible by 3 for $n \geq 0$.**

Base case: For $n = 0$, $8^0 - 5^0 = 0$ is divisible by 3.

Induction step. Assume $8^n - 5^n$ is divisible by 3. We must show $8^{n+1} - 5^{n+1}$ is divisible by 3. We have

$$8^{n+1} - 5^{n+1} = 8 \cdot 8^n - 5 \cdot 5^n = (5 + 3) \cdot 8^n - 5 \cdot 5^n = 5(8^n - 5^n) + 3 \cdot 8^n.$$

Since we know $8^n - 5^n$ is divisible by 3, and $3 \cdot 8^n$ is clearly divisible by 3, the entire expression is divisible by 3. Thus the result is true by induction. \square

The key to the approach above is we need to find a way to work $8^n - 5^n$ into things. To do that, the sneaky algebra trick of breaking 8 into 5 + 3 is used.¹

3. **Prove that $\frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \dots + \frac{1}{(2n-1)(2n+1)} = \frac{n}{2n+1}$ for $n \geq 1$.**

Base case: For $n = 1$, the left hand side is $\frac{1}{1 \cdot 3} = \frac{1}{3}$, while the right side is $\frac{1}{2 \cdot 1 + 1} = \frac{1}{3}$.

Induction step: Assume $\frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \dots + \frac{1}{(2n-1)(2n+1)} = \frac{n}{2n+1}$. We must show $\frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \dots + \frac{1}{(2n+1)(2n+3)} = \frac{n+1}{2n+3}$. The sum from our assumption is hiding in the bigger sum, so the bigger sum can be rewritten using the induction hypothesis as $\frac{n}{2n+1} + \frac{1}{(2n+1)(2n+3)}$. We can then simplify this into

$$\frac{n}{2n+1} + \frac{1}{(2n+1)(2n+3)} = \frac{2n^2 + 3n + 1}{(2n+1)(2n+3)} = \frac{(2n+1)(n+1)}{(2n+1)(2n+3)} = \frac{n+1}{2n+3}.$$

4. **Let F_1, F_2, \dots denote the Fibonacci numbers. Prove that $F_1 + F_2 + \dots + F_n = F_{n+2} - 1$.**

Base case: The base case is $n = 1$. We have the left side equal to F_1 , which is 1 and the right side equal to $F_3 - 1$, which is also 1.

Induction step: Assume the result holds for n . We need to show it holds for $n + 1$, namely that $F_1 + F_2 + \dots + F_{n+1} = F_{n+3} - 1$. We have

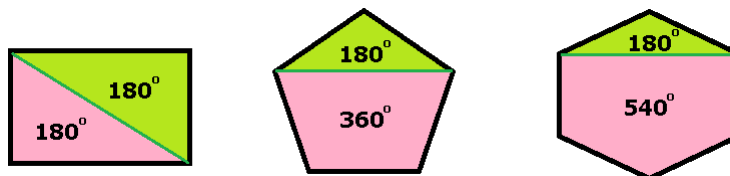
$$F_1 + F_2 + \dots + F_{n+1} = (F_1 + F_2 + \dots + F_n) + F_{n+1} = F_{n+2} - 1 + F_{n+1} = F_{n+3} - 1.$$

The last step comes from the recursive formula for the Fibonacci, that $F_{n+3} = F_{n+2} + F_{n+1}$. \square

5. **Prove that the sum of the angles in an n -gon is $180(n - 2)$ degrees.**

Base case: The smallest polygon is a triangle ($n = 3$) and we know the sum of the angles in a triangle is 180° . This fits with the formula $180(n - 2)$.

Induction step: Assume that the sum of the angles in an n -gon is $180(n - 2)$ degrees. We must show that the sum of the angles in an $n + 1$ -gon is $180(n - 1)$ degrees. To get an idea how the general argument works, let's look at a few small cases:



We can break up a quadrilateral (4-gon) into two triangles. We can break up a pentagon (5-gon) into a quadrilateral and a triangle. We can break up a hexagon (6-gon) into a pentagon and a triangle. In general, we can break up an $n + 1$ -gon into an n -gon and a triangle. The sum of the angles in the n -gon is $180(n - 2)$ degrees by our hypothesis. To that we add 180° from the triangle to get $180(n - 2) + 180 = 180(n - 1)$, which is what we wanted to show. \square

Not all induction proofs are as algebraic as this example shows.

¹An alternate approach would be to insert $(5 \cdot 8^n - 5 \cdot 8^n)$ into the expression (just adding 0), and rearrange terms.

6. **Prove that the equation $x^2 + y^2 = z^{2n}$ has a solution where x , y , and z are positive integers.**

Base case: The equation $x^2 + y^2 = z^2$ has plenty of solutions in positive integers. For instance, we have $3^2 + 4^2 = 5^2$.¹

Induction step: Assume we have an integer solution (x_0, y_0, z_0) to $x^2 + y^2 = z^{2n}$. We must find an integer solution to $x^2 + y^2 = z^{2(n+1)}$. We have $x_0^2 + y_0^2 = z_0^{2n}$. Multiplying this equation through by z_0^2 gives $x_0^2 z_0^2 + y_0^2 z_0^2 = z_0^{2n} z_0^2$, which we can rewrite as $(x_0 z_0)^2 + (y_0 z_0)^2 = z_0^{2(n+1)}$. So $(x_0 z_0, y_0 z_0, z_0)$ is an integer solution. \square

As an example, starting with $3^2 + 4^2 = 5^2$, if we multiply through by 5^2 , we get $(3 \cdot 5)^2 + (4 \cdot 5)^2 = 5^4$, or $15^2 + 20^2 = 5^4$. If we multiply through by 5^2 again, we get $(15 \cdot 5)^2 + (20 \cdot 5)^2 = 5^6$, or $75^2 + 100^2 = 5^6$. We see that we can keep this process up. Induction is what allows us to turn this recursive process of generating new solutions into a formal proof.

7. **Recall De Morgan's law states that $(A \cup B)^C = A^C \cap B^C$. Prove that it works for any number of sets, namely that $(A_1 \cup A_2 \cup \dots \cup A_n)^C = A_1^C \cap A_2^C \cap \dots \cap A_n^C$.**

Base case: The base case $n = 2$ is De Morgan's law, which we take as given.

Induction step: Suppose that $(A_1 \cup A_2 \cup \dots \cup A_n)^C = A_1^C \cap A_2^C \cap \dots \cap A_n^C$. We need to show that $(A_1 \cup A_2 \cup \dots \cup A_{n+1})^C = A_1^C \cap A_2^C \cap \dots \cap A_{n+1}^C$. We have

$$(A_1 \cup A_2 \cup \dots \cup A_{n+1})^C = ((A_1 \cup A_2 \cup \dots \cup A_n) \cup A_{n+1})^C.$$

We can apply De Morgan's law to this, thinking of the stuff inside the parentheses as a union of two sets, to get

$$(A_1 \cup A_2 \cup \dots \cup A_n)^C \cap A_{n+1}^C.$$

We can use the induction hypothesis to rewrite this as

$$(A_1^C \cap A_2^C \cap \dots \cap A_n^C) \cap A_{n+1}^C,$$

which is equivalent (using the associative property) to what we want. \square

This sort of argument of extending a result from two objects to arbitrarily many is very common in mathematics. Once you are comfortable with it, it is almost automatic to do and many authors will just say that a result follows easily by induction, relying on the reader to supply the routine details.

8. **Prove that there are $n(n+1)/2$ two-element subsets of $\{1, 2, \dots, n\}$.**

Base case: The base case is $n = 2$. There is one two-element subset of $\{1, 2\}$ and the formula $n(n-1)/2$ also gives 1.

Induction step: Suppose that there are $n(n-1)/2$ two-element subsets of $\{1, 2, \dots, n\}$. We need to show that there are $(n+1)n/2$ two-element subsets of $\{1, 2, \dots, n+1\}$. The key is that the two-element subsets of $\{1, 2, \dots, n+1\}$ consist of the two-element subsets of $\{1, 2, \dots, n\}$ along with all the one-element subsets of $\{1, 2, \dots, n\}$ unioned with $\{n+1\}$. There are $n(n-1)/2$ of the former (by the induction hypothesis) and n of the latter, giving us a total of $n(n-1)/2 + n = n(1 + (n-1)/2) = (n+1)n/2$ subsets, as desired. \square

This is the same breakdown we did earlier when finding a recursive formula for the number of subsets. As a concrete example, here is how we go from $n = 4$ to $n = 5$. The subsets of $\{1, 2, 3, 4, 5\}$, written in abbreviated form, are 12, 13, 23, 14, 24, 34, and 15, 25, 35, 45. There are $4(4-1)/2$ in the first group and 4 in the second group.

Strong induction

When doing induction proofs, it is sometimes useful to rely on more than just one previous case. For instance, instead of showing that case n implies case $n+1$, we can use more previous cases, like $n-1$, $n-2$, etc. This is analogous to something like the Fibonacci formula, $F_{n+1} = F_n + F_{n-1}$, where the next value depends on multiple previous values.

¹This triple $(3, 4, 5)$, is an example of a *Pythagorean triples*. It corresponds to a triangle whose sides are all integers. There are many other examples, like $(5, 12, 13)$, $(7, 24, 25)$, and multiples of them.

In fact, we can rely on any previous cases. Doing so is sometimes called using *strong induction*. It can actually be shown that strong induction is equivalent to ordinary induction. That is, at least in theory, anything that can be proved by strong induction could also be proved by regular induction. It's just that the approach using strong induction might be easier.

Here are some examples:

1. **The Fundamental Theorem of Arithmetic states that every integer $n \geq 2$ can be written as a product of prime numbers. Prove it using strong induction.**¹

Base case: We can write $n = 2$ trivially as a product of primes, namely as a product of one term (just itself).

Induction step: Assume all integers $2, 3, \dots, n$ can be written as products of primes. We must show that $n + 1$ can be written as a product of primes. If $n + 1$ is itself prime, then it is trivially a product of primes and we're done. If not, then we can factor it into ab , with a and b both in the range from 2 through n . By the induction hypothesis, both a and b can be written as products of primes, so $n + 1 = ab$ can be as well. \square

Notice how the strong induction works. We assume that the result holds for *all* integers from the base case through n and use that to prove the result holds for $n + 1$. For instance, if $n + 1 = 18$, then we break 18 up into 3 and 6 and rely on the fact that 3 and 6 are products of primes.

2. **Let F_1, F_2, \dots denote the Fibonacci numbers. Prove that $F_n < 2^n$ for $n \geq 1$.**

Base case: We have $F_1 = 1$ and $2^1 = 2$, so the result works for $n = 1$. We also have $F_2 = 2$ and $2^2 = 4$, so the result holds for $n = 2$.

Induction step: Assume the result holds for $k = 2, 3, \dots, n$. We need to show it holds for $n + 1$, namely that $F_{n+1} < 2^{n+1}$. We have

$$F_{n+1} = F_n + F_{n-1} < 2^n + 2^{n-1} = 2^{n-1}(2 + 1) < 2^{n-1}(2 + 2) = 2^{n+1}. \quad \square$$

In the work above, strong induction is needed since we use more than one previous case. Note also that since the induction step requires going back to two previous cases, we need two base cases to get us started.

Some final thoughts on induction

The key to induction, just like recursion, is building up from smaller cases. The following algebraic breakdowns are often useful for doing this:

- $2^{n+1} = 2 \cdot 2^n$
- $(n + 1)! = (n + 1)n!$
- $F_{n+1} = F_n + F_{n-1}$
- $a_1 + a_2 + \dots + a_{n+1} = (a_1 + a_2 + \dots + a_n) + a_{n+1}$
- $a_1 a_2 \dots a_{n+1} = (a_1 a_2 \dots a_n) \cdot a_{n+1}$

The last two rules are meant to apply to any sequence. And of course variants on these are also useful, like $2^{n+2} = 2^2 \cdot 2^n$ or $(n + 1)! = (n + 1)n(n - 1)!$.

One thing to be careful about in induction proofs is to not forget the base case. It is rare, but still possible for the induction step to work but have the result be false because the base case doesn't work. For instance, suppose we have the sequence given by $a_1 = 2$ and $a_{n+1} = a_n + \frac{1}{2^n}$ for $n \geq 1$, and suppose we want to prove that $a_n \geq 10$. Skipping the base case, we would assume that $a_n \geq 10$ and show that $a_{n+1} \geq 10$. Since $a_{n+1} = a_n + \frac{1}{2^n}$ is bigger than a_n , we must have $a_{n+1} \geq 10$. The problem is that the sequence never gets to 10. If it did get to 10, then the proof above would show that all terms after that point are greater than 10, but there is no basis, no place to start. In short, always remember the base case.

¹The theorem also states that the factorization is unique, but we won't consider that here.

Chapter 5

Counting

5.1 The multiplication rule

Consider the following question:

An ice cream parlor has types of cones, 35 flavors, and 5 types of toppings. Assuming you get one cone, one scoop of ice cream, and one topping, how many orders are possible?

The answer is $3 \times 35 \times 5 = 525$ orders. For each of the 3 cones, there are 35 different flavors of ice cream, so there are $3 \times 35 = 105$ possible cone/ice cream combinations. Then for each of those 105 combinations, there are 5 different toppings, leading to $105 \times 5 = 525$ possible orders.

Here is another question:

How many possible strings of 2 capital letters are there?

The answer is $26 \times 26 = 576$. The possible strings include AA, AB, ..., AZ then BA, BB, ..., BZ, all the way down to ZA, ZB, ..., ZZ. That is, 26 possible strings starting with A, 26 with B, etc., and 26 possible starting values. So there are 26×26 or 26^2 possibilities.

One visual technique that helps with a problems like this is to draw some slots for the possibilities:

$$\begin{array}{cc} \overline{A-Z} & \overline{A-Z} \\ \hline 26 & 26 \end{array}$$

If instead of 2 capital letters, we have 6 capital letters, there would be 26^6 possibilities. We might draw the slots for the letters like below:

$$\overline{26} \quad \overline{26} \quad \overline{26} \quad \overline{26} \quad \overline{26} \quad \overline{26}$$

These examples lead us to the following useful rule:

Multiplication Rule: If there are x possibilities for one thing and y possibilities for another, then there are $x \times y$ ways of combining both possibilities.

It's a simple rule, but it is an important building block in more sophisticated counting problems. Here are a few more examples:

1. A password can consist of capital and lowercase letters, the digits 0 through 9, and any of 32 different special characters. How many 6-character passwords are possible?

There are $26+26+10+32$ different possibilities for each character, or 94 possibilities in total. With 6 characters, there are thus $94^6 = 689,869,781,056$. The slot diagram is shown below:

$$\overline{94} \overline{94} \overline{94} \overline{94} \overline{94} \overline{94}$$

Is this a lot of passwords? Not really. Good password-cracking programs that make use of a computer's graphics processing unit can scan through tens of billions of passwords per second. So they would be able to crack a 6-character password in under a minute.

- The old system for phone numbers was as follows: All possible phone numbers were of the form ABC-DEF-GHIJ, where A, D, and F could be any digits from 2 through 9, B could be either a 0 or a 1, and the others could be any digit. How many phone numbers were possible under this system?

The slot diagram helps make things clear:

$$\begin{array}{cccccccccccc} \overline{A} & \overline{B} & \overline{C} & \overline{D} & \overline{E} & \overline{F} & \overline{G} & \overline{H} & \overline{I} & \overline{J} \\ \hline 8 & 2 & 10 & 8 & 10 & 8 & 10 & 10 & 10 & 10 \end{array}$$

So there are $8 \times 2 \times 10 \times 8 \times 10 \times 8 \times 10^4 = 1,024,000,000$ phone numbers. If 50 million new phone numbers were issued each year (after reusing old ones), we can see that we would run out of numbers after about 20 years.

- How many times does the print statement below get executed?

```
for i = 1 to 1000
  for j = 1 to 200
    for k = 1 to 50
      print "hi"
```

The answer is $1000 \times 200 \times 50 = 10,000,000$ times.

- A website has a customizable brochure, where there are 66 possible checkboxes. Checking or unchecking one of them determines if certain information will be or not be included in the brochure. How many brochures are possible?

There are two possibilities for each checkbox (checked or unchecked), and 66 possible checkboxes. See the figure below:



So there are $2^{66} \approx 7.3 \times 10^{19}$ possible brochures.

- A combination lock consists of a code of 3 numbers, where each number can run from 1 through 39. How long would it take to break into the lock just by trying all the possibilities?

There are $39^3 = 59,319$ possibilities in total. Assuming 10 seconds to check a combination, we are looking at about 164 hours.

Multiplication rule with no repeats

We know that there are 26^6 6-character strings of capital letters. What if repeats are not allowed? In that case, while there are still 26 possibilities for the first character, there are only 25 possibilities for the second letter since we can't reuse the first letter. There are 24 possibilities for the next letter, 23 for the next, etc. See the diagram below:

$$\overline{26} \overline{25} \overline{24} \overline{23} \overline{22} \overline{21}$$

There are thus $26 \times 25 \times 24 \times 23 \times 22 \times 21 = 165,765,600$ possibilities in total.

This type of argument can be used in other situations when counting things without repetition. In particular, it is used to count how many ways there are to rearrange things.

5.2 Rearranging things

Suppose we want to know how many ways there are to rearrange the letters ABCDEFG. There are 7 possibilities for what to make the first letter, then once that is used, there are 6 possibilities for the second, 5 for the third, etc. See the figure below:

$$\overline{7} \quad \overline{6} \quad \overline{5} \quad \overline{4} \quad \overline{3} \quad \overline{2} \quad \overline{1}$$

So there are 7! possibilities in total. In general we have to following:

There are $n!$ ways to rearrange a set of n different objects. Such a rearrangement is called a *permutation*.

See Section 4.1 for a different approach to permutations. Here are a couple of examples:

1. You and 4 friends play a game where order matters. To be fair, you want to play all the possible orderings of the game. How many such orderings are there?

There are 5 people, so there are $5! = 120$ ways of rearranging them.

2. You want to take a trip through 6 cities. There is a flight from each city to each other one. How many possible trip orders could you take?

There are $6! = 720$ orders in which you could visit the cities.

3. A rather silly sorting algorithm, called Bogosort, involves randomly rearranging the elements in an array and checking to see if they are in order. It stops when the array is sorted. How many steps would it take to sort an array of 1000 elements?

While it could in theory take only one step, on average it will take around $1000!$ steps, since there are $1000!$ ways to rearrange the elements of the array and only one of those corresponds to a sorted array.¹

Rearrangements with repeats

Suppose we want to know how many ways there are to rearrange a collection of items with repeats. For example, how many ways are there to rearrange the letters of the string AAB?

Suppose for a moment that the A's were different, say aAB instead of AAB. Then the 6 possible rearrangements would be as follows:

aAB, AaB,
aBA, ABa,
BAa, BaA

If we turn the lowercase a back into an uppercase A, then the first row's rearrangements both correspond to AAB, the second row's correspond to ABA, and the third row's correspond to BAA. For each of these three classes, there are $2!$ ways to rearrange the A's if they were different. From this we get that there are $\frac{6!}{2!}$ ways to rearrange AAB. In general, we have the following:

The number of rearrangements of a collection of n items where item 1 occurs m_1 times, item 2 occurs m_2 times, etc. is

$$\frac{n!}{m_1! \cdot m_2! \cdot \dots \cdot m_k!}$$

(where k is the number of distinct items).

¹Note that $1000!$ is a particularly large number, being about 2500 digits long.

I usually just think of this rule as the “Mississippi rule,” based on the first problem below.

1. How many ways are there to rearrange the letters in the word MISSISSIPPI?

There are 11 letters in MISSISSIPPI, consisting of 1 M, 4 I's, 4 S's, and 2 P's, so the number of rearrangements is

$$\frac{11!}{1! \cdot 4! \cdot 4! \cdot 2!} = 34,650.$$

2. Suppose we want to know how many possible ways 10 rolls of a die could occur in which we end up with exactly 4 ones, 2 twos, and 4 sixes.

This is equivalent to rearranging the string 1111226666. There are $\frac{10!}{4! \cdot 2! \cdot 4!} = 3150$ ways in total.

5.3 Subsets

Here is a question: How many subsets of $\{A, B, C, D, E, F\}$ are there in total?

To answer this, when we create a subset, we can think of going through the elements one at a time, choosing whether or not to include the element in the subset. For instance, for the subset $\{A, D, F\}$, we include A, leave out B and C, include D, leave out E, and include F. We can encode this as the string 100101, with a 1 meaning “include” and a 0 meaning “leave out,” like in the figure below:

A	B	C	D	E	F
1	0	0	1	0	1
A			D		F

Each subset corresponds to such a string of zeros and ones, and vice-versa. There are 2^6 such strings, and so there are 2^6 possible subsets.

In general, we have the following:

There are 2^n possible subsets of an n -element set.

Now let's consider a related question: How many 2-element subsets of $\{A, B, C, D, E, F\}$ are there?

As described above, two-element subsets of $\{A, B, C, D, E, F\}$ correspond to strings of zeros and ones with exactly 2 ones and 4 zeros. So we just want to know how many ways there are to rearrange the string 110000. There are $\frac{6!}{2!4!} = 15$ such ways.

A similar argument works in general to show that there are $\frac{n!}{k!(n-k)!}$ k -element subsets of an n -element subset. This expression is important enough to have its own notation, $\binom{n}{k}$. It is called a *binomial coefficient*. In general, we have

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots 1}.$$

The rightmost expression comes because a lot of terms cancel out from the factorials. It is usually the easiest one to use for computing binomial coefficients. In that expression, there will be the same number of terms in the numerator and denominator, with the terms in the denominator starting at k and working their way down, while the terms in the numerator start at n and work their way down.

To summarize, we have the following:

The number of k -element subsets of an n -element set is $\binom{n}{k}$.

As an example, the number of 3-element subsets of a 7-element set is

$$\binom{7}{3} = \frac{7!}{3! \cdot 4!} = \frac{7 \cdot 6 \cdot 5}{3 \cdot 2 \cdot 1} = 35.$$

As another example, the number of 4-element subsets of a 10-element set is

$$\binom{10}{4} = \frac{10!}{4! \cdot 6!} = \frac{10 \cdot 9 \cdot 8 \cdot 7}{4 \cdot 3 \cdot 2 \cdot 1} = 210.$$

See Section 4.1 for a different approach to finding the number of subsets. There are a lot of counting problems that involve binomial coefficients. Here are a few examples:

1. How many 5-card hands are possible from an ordinary deck of 52 cards?

There are $\binom{52}{5} \approx 2.6$ million hands.

2. How many ways are there to choose a group of 3 students from a class of 20?

There are $\binom{20}{3} = 1140$ possible groups.

In general, $\binom{n}{k}$ tells us how many ways there are to pick a group of k different things from a group of n things, where the order of the picked things doesn't matter.

A few rules for working with binomial coefficients

Here are a few quick rules for working with binomial coefficients:

1. $\binom{n}{0} = 1$ and $\binom{n}{n} = 1$.

That is, there is one 0-element subset of a set (the empty set) and one n -element subset of an n -element set (the set itself).

2. $\binom{n}{k} = \binom{n}{n-k}$

This is useful computationally. For instance, $\binom{20}{17}$ is the same as $\binom{20}{3}$.

3. Binomial coefficients correspond to entries in Pascal's triangle. Here are the first few rows.

$$\begin{array}{cccccccc}
 & & & & 1 & & & & \\
 & & & & 1 & & 1 & & \\
 & & & 1 & 2 & 1 & & & \\
 & & 1 & 3 & 3 & 1 & & & \\
 & 1 & 4 & 6 & 4 & 1 & & & \\
 1 & 5 & 10 & 10 & 5 & 1 & & & \\
 1 & 6 & 15 & 20 & 15 & 6 & 1 & &
 \end{array}$$

The top entry is $\binom{0}{0}$. The two entries below it are $\binom{1}{0}$ and $\binom{1}{1}$. In general, the entry in row n , column k (starting counting at 0) is $\binom{n}{k}$.

4. Binomial coefficients show up in the very useful binomial formula:

$$(x + y)^n = x^n + \binom{n}{1}x^{n-1}y + \binom{n}{2}x^{n-2}y^2 + \cdots + \binom{n}{n-1}xy^{n-1} + y^n.$$

The easy way to get the coefficients is to use Pascal's triangle. Here are a few examples of the formula:

$$(x + 1)^3 = x^3 + 3x^2 + 3x + 1$$

$$(x + 1)^4 = x^4 + 4x^3 + 6x^2 + 4x + 1$$

$$(x + y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4x^3y + y^4$$

Here is a little more complicated example:

$$\begin{aligned}(3x - y)^4 &= (3x)^4 + 4(3x)^3(-y) + 6(3x)^2(-y)^2 + 4(3x)(-y)^3 + (-y)^4 \\ &= 81x^4 - 108x^3y + 54x^2y^2 - 12xy^3 + y^4.\end{aligned}$$

5.4 Addition rule

Counting problems involving the word “or” usually involve some form of addition. Consider the following very simple question: How many cards in a standard deck are queens or kings? There are 4 queens and 4 kings, so there are $4 + 4 = 8$ in total. We have the following rule:

Addition Rule: If there are x possibilities for A , y possibilities for B , and no way that both things could simultaneously occur, then there are $x + y$ ways that A or B could occur.

Here are some examples:

1. How many 5-card hands are flushes, where all 5 cards have the same suit?

There are 13 cards in each of the 4 suits: diamonds, hearts, clubs, and spades. So there are $\binom{13}{5}$ hands that contain only hearts. Likewise there are $\binom{13}{5}$ hands that contain only hearts, and similarly for the other two suits. So overall, there are $\binom{13}{5} + \binom{13}{5} + \binom{13}{5} + \binom{13}{5}$ possible flush hands.

2. Problems containing the words “at least” or “at most” can often be done with the addition rule. For instance, how many 8-character strings of zeros and ones contain at least 6 zeros? The phrase “at least 6 zeros” here means our string could have 6, 7, or 8 zeros. So we add up the number of strings with exactly 6 zeros, the number with exactly 7, and the number with exactly 8.

To get how many contain exactly 6 zeros, think of it this way: There are 8 slots, and 6 of them must be zeros. There are $\binom{8}{6} = 28$ such ways to do this. Similarly, there are $\binom{8}{7} = 8$ strings with exactly 7 zeros. And there is 1 (or $\binom{8}{8}$) strings with 8 zeros.

So there are $28 + 8 + 1 = 37$ strings with at least 6 zeros.

The simple question we started this section out with asked how many cards in a standard deck are queens or kings. What if we change it to ask how many are queens or diamonds? There are 4 queens and 13 diamonds, but the answer is not 17. One of the cards in the deck is the queen of diamonds, which gets double-counted. So there are actually $4 + 13 - 1 = 16$ cards that are queens or diamonds. In general, we have the following rule:

General addition rule: If there are x possibilities for A , y possibilities for B , and z possibilities where both occur, then there are $x + y - z$ ways that A or B could occur.

An alternate approach to the above rule is to turn the question into an “or” question with no overlap. For instance, to count how many queens or diamonds are in a deck, we can approach it as counting how many cards are queens but not diamonds, how many are diamonds but not queens, and how many are both queens and diamonds. That gives us $12 + 3 + 1 = 16$ as our answer.

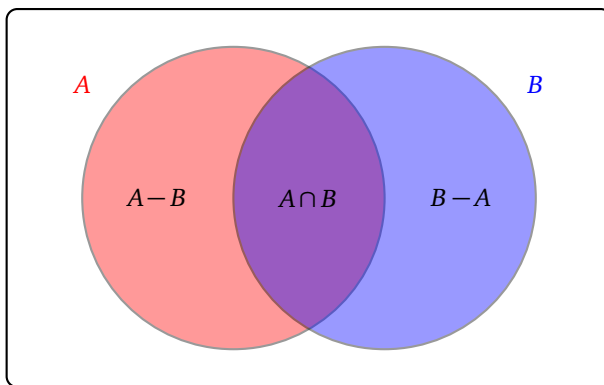
In terms of set notation, we could write the general addition rule as

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

And we could write this alternate approach as

$$|A \cup B| = |A - B| + |B - A| + |A \cap B|.$$

A Venn diagram helps make these formulas clearer:



Here are some examples of the rule:

1. How many integers between 1 and 1000 are divisible by 2 or 3?

Every second integer is divisible by 2, so there are 500 integers divisible by 2. Every third integer is divisible by 3, so there are $\lfloor 1000/3 \rfloor = 333$ integers divisible by 3. However, multiples of 6 are double-counted as they are divisible by both 2 and 3, so we have to subtract them off. There are $\lfloor 1000/6 \rfloor = 166$ of them. So in total, $500 + 333 - 166 = 667$ integers are divisible by 2 or 3.

2. How many three-character strings of capital letters start or end with a vowel (A, E, I, O, or U)?

We can add up the ones that start with a vowel and the ones that end with a vowel, but words that both start and end with a vowel are double-counted. To fix this problem, we have to subtract them from the total. See the slot diagram below:

$$\overline{5} \overline{26} \overline{26} + \overline{26} \overline{26} \overline{5} - \overline{5} \overline{26} \overline{5}$$

In total, there are $5 \times 26^2 + 5 \times 26^2 - 5^2 \times 26 = 6110$ ways.

Alternatively, we could approach the problem as shown in the slot diagram below:

$$\overline{5} \overline{26} \overline{21} + \overline{21} \overline{26} \overline{5}$$

That is, we break things up into three disjoint possibilities: things that start but don't end with a vowel, things that end but don't start with a vowel, and things that both start and end with a variable. This gives $5 \times 26 \times 21 + 21 \times 26 \times 5 + 5 \times 26 \times 21 = 6110$.

5.5 Negation rule

Another simple question: How many cards in a standard deck are not twos? There are 52 cards and 4 twos, so there are 48 cards that are not twos. In general, we have

Negation rule: To compute the number of ways that A can't happen, count the number of ways it can happen and subtract from the total number of possibilities.

Using set notation, we could write this as $|A^c| = |U| - |A|$, where U is the universal set (which changes depending on the problem). Here are a couple of examples:

1. How many integers from 1 to 1000 are not divisible by 2 or 3?

In the previous section, we found that 667 of those integers are divisible by 2 or 3, so there are $1000 - 667 = 333$ that are not divisible by 2 or 3.

- How many strings of 6 capital letters have at least one vowel?

The phrase “at least one vowel” means would could have 1, 2, 3, 4, 5, or 6 vowels. That could get tedious to compute. But the complement of having at least one vowel is having no vowels at all. So we can approach this problem by finding how many strings of 6 capital letters there are (which is 26^6) and subtracting how many contain no vowels (which is 21^6). So our answer is $26^6 - 21^6 = 223,149,655$.

5.6 Summary

Counting problems come in many varieties and can sometimes be tricky. But they can often be broken down into an application of the rules above. It is good to have a collection of “model problems” and to try to see if your counting problem fits one of the models. For instance, the “Mississippi problem,” about the number of ways to rearrange the letters of the word MISSISSIPPI, can be used as a model for other seemingly unrelated problems. Suppose we want to know how many ways there are to arrange 5 freshman, 4 sophomores, 6 juniors and 2 seniors, where only the order by class matters. We can think of this as rearranging the letters of the word FFFF0000JJJJSS, just like the Mississippi problem.

Here are model problems for each rule.

- Multiplication rule** The number of 4-letter strings of capital letters that end in a vowel is $26^3 \times 5$. Multiplication rule problems are sometimes easier to visualize with a slot diagram, like the one below:

$$\frac{A-Z}{26} \frac{A-Z}{26} \frac{A-Z}{26} \frac{AEIOU}{5}$$

The multiplication rule is used in counting problems involving the word “and.” It is the simplest and most important rule.

- Multiplication rule with no repeats**

The number of 4-letter strings of capital letters that don’t repeat letters is $26 \times 25 \times 24 \times 23$.

- Permutations**

The number of ways to rearrange the letters A, B, C, D, E is $5!$.

- Permutations with repeats**

The number of ways to rearrange the letters of the word MISSISSIPPI is $\frac{11!}{1! \cdot 4! \cdot 4! \cdot 2!}$

- Combinations**

The number of 5-card hands from a 52-card deck is $\binom{52}{5}$.

A combination is the answer to a question where the you want to know how many groups of a certain size you can make. Only the members of the group matter, not the order in which their chosen.

In particular, the number of k -element subsets of an n -element set is $\binom{n}{k}$.

- Simple addition rule**

The number of strings of 1, 2, or 3 capital letters is $26 + 26^2 + 26^3$.

Use the simple addition rule with questions involving the word “or” when the possibilities are all disjoint. If the possibilities are not disjoint, use the next rule.

- General addition rule**

The number of 2-letter strings of capital letters that start or end with a vowel is $26 \times 5 + 5 \times 26 - 5 \times 5$. It is the number that start with vowel, plus the number that end with a vowel, minus the ones that both start and end with vowels (which are double-counted).

Alternately, we could do this as $26 \times 21 + 21 \times 26 + 5 \times 5$, the number that start but don’t end with a vowel, plus the number that end but don’t start with a vowel, plus the number that both start and end with vowels.

8. Negation rule

The number of strings of four capital letters that don't contain a repeat is $26^4 - 26 \times 25 \times 24 \times 23$, gotten by subtracting the number that do contain repeats from the total.

To answer how many strings of six capital letters contain a vowel, a nice shortcut is to instead answer how many contain no vowels and subtract from the total to get $26^6 - 21^6$.

The negation rule is useful for questions involving the word "not" and it is also useful, as in the second example above, if counting the complement of something is easier than counting the thing itself.

Chapter 6

Probability

6.1 The basic rule

The most basic rule of probability is the following:

The probability of an event A occurring is

$$P(A) = \frac{\text{The number of outcomes involving A}}{\text{The total number of possible outcomes}}$$

For example, if a jar contains 5 red marbles and 10 blue marbles, and we randomly pick a marble, the probability of picking a red marble is $\frac{5}{15}$.

A slightly harder example: What is the probability that if we take a random string of 3 capital letters that it contains no repeated letters?

There are 26^3 total 3-letter strings and $26 \times 25 \times 24$ that contain no repeats. So the probability is

$$\frac{26 \times 25 \times 24}{26^3} = .888.$$

In general, there is a close connection between probability and counting. Many probability problems reduce to counting problems.

Here is a useful example. What are the probabilities for all the outcomes of rolling two dice? The possible outcomes are 2 through 12. It will be helpful to think of the dice being different colors, say a red and a green die, like below:



In the table below, the outcomes are given in shorthand. For example, the dice above (red 3, green 1) are represented by the shorthand 31.

total	outcomes	prob.	%
2	11	1/36	2.8
3	12, 21	2/36	5.6
4	13, 22, 31	3/36	8.3
5	14, 23, 32, 41	4/36	11.1
6	15, 24, 33, 42, 51	5/36	13.9
7	16, 25, 34, 43, 52, 61	6/36	16.7
8	26, 35, 44, 53, 62	5/36	13.9
9	36, 45, 54, 63	4/36	11.1
10	46, 55, 64	3/36	8.3
11	56, 65	2/36	5.6
12	66	1/36	2.8

There are 36 possible outcomes, and to get the probabilities we use the basic rule. Notice how much more likely a 7 is than a 2 or 12.

6.2 The multiplication rule

Related to the multiplication rule for counting is the multiplication rule for probability. It works for *independent events*. Two events are called independent if the fact that one of them occurs does not affect the probability that the other occurs. For instance, dice rolls are independent. What happens on the first roll has no effect on the second roll. On the other hand, if we are drawing cards from a deck and setting them aside, then the first and second draws are not independent. For instance, if the first card is an ace, then we know there are only 3 aces left in the deck and that affects our probability for the second card.

Below is the multiplication rule. We use the notation $P(A)$ to denote the probability of the event A occurring.

Multiplication rule: If events A and B are independent, then $P(A \text{ and } B) = P(A)P(B)$.

In other words, the probability that both A and B will occur is gotten by multiplying the probabilities of A and B . Here are a few examples:

1. Suppose we roll a die twice. What is the probability both rolls come out to be twos?

The probability of a two is $1/6$. The rolls are independent, so the probability they both come out as twos is

$$\frac{1}{6} \times \frac{1}{6},$$

which is $\frac{1}{36}$ or about 2.8%.

2. Suppose we roll a die 10 times. What is the probability all 10 rolls come out to be twos?

By the multiplication rule, the probability is

$$\frac{1}{6} \times \frac{1}{6} \times \cdots \times \frac{1}{6} = \left(\frac{1}{6}\right)^{10},$$

which is a very small number, only about a 1 in 60 million chance.

3. A strand of 100 Christmas lights needs all 100 lights to be working in order to light up. Assuming each individual light has a 98% chance of working, what is the probability that the strand lights up? Assume independence.

The solution is

$$(.98)^{100},$$

which is about 13.3%, somewhat surprisingly low. One way to think about this is that there is a 2% chance that any light will fail, and we are taking 100 such chances. We can only press our luck so long before something breaks down.

It is interesting to note that if the lights were a bit less effective, say 95% effective, then the probability that the strand lights up drops all the way to 0.5%. If the lights were a lot more effective, say 99.9% effective, then there is still only a 90% chance that the strand will light up.

It is worth noting that we have to assume the lights are all independent. There are many real-life scenarios that would negate this assumption, such as if all the lights were manufactured in the same batch at the factory when something went wrong, or if someone steps on the light strand and crushes several of the lights. Real-life problems are often very difficult to solve without assuming independence, so we often assume it. It means that our answer is only an approximation of the true probability.

4. As a somewhat similar example, suppose we have an unreliable alarm clock that fails 5% of the time (this is once every 20 days, on average). We can make things more reliable if we add a second alarm clock, acting independently of the first. If that alarm clock also fails 5% of the time, the probability that they both fail is $.05 \times .05 = .0025$, or once in every 400 days on average. Adding another alarm clock would bring the probability down to $.000125$, or once in every 8000 days.

This assumes the alarm clocks are independent of each other. If they are all plugged in to the same outlet and the power goes out, then things are not very independent.

General multiplication rule

For events that are not necessarily independent, we have the following rule

General multiplication rule $P(A \text{ and } B) = P(A) \cdot P(B|A)$.

The $P(B|A)$ part is read as “the probability of B given A .” In other words, given that A happens, what is the probability that B also happens? If the events are independent, then $P(B|A) = P(B)$, as the occurrence of A has no effect on the occurrence of B . On the other hand, if the events are not independent, then things are more interesting. Here are some examples:

1. We have a jar with 5 red marbles and 10 blue marbles. Suppose we pick 2 marbles from the jar, setting each one aside as we pick it. What is the probability both are blue?

The probability that the first is blue is $\frac{10}{15}$. For the second marble, there is one blue marble now missing from the jar, so the probability that is blue is $\frac{9}{14}$. By the general multiplication rule, the probability that both are blue is

$$\frac{10}{15} \cdot \frac{9}{14},$$

or about 43%. Referring back to the formula, $P(A)$ is the probability, $\frac{10}{15}$, that the first one is blue, and $P(B|A)$ is the probability that the second is blue, given that the first one was blue. This is $\frac{9}{14}$.

We can extend things further: Suppose we pick 4 marbles. What is the probability all 4 are blue? The answer is

$$\frac{10}{15} \cdot \frac{9}{14} \cdot \frac{8}{13} \cdot \frac{7}{12},$$

or about 15%.

On the other hand, if we put the marbles back after each pick, then the probability that all 4 are blue would be $(\frac{10}{15})^4$, since the probability would not change with each pick.

Note also that picking all 4 out of the jar at once, probabilistically speaking, is the same as picking one at a time and setting each aside.

Note further similarity between this rule and the multiplication rule for counting events with no repeats.

2. Suppose we are dealt 3 cards. What is the probability all of them are aces?

There are 52 cards in a deck, 4 of which are aces. Thus the probability all three cards we are dealt are aces is

$$\frac{4}{52} \cdot \frac{3}{51} \cdot \frac{2}{50},$$

which is $.00018$ or about 1 in every 5500 hands.

3. A class consists of 5 boys and 15 girls. A professor picks 3 students, supposedly randomly, for a project and they all turned out to be boys. What is the probability that this would happen purely by chance?

Using the general multiplication rule, the probability is

$$\frac{5}{20} \cdot \frac{4}{19} \cdot \frac{3}{18} \approx 0.00097.$$

This is roughly a 1 in 1000 chance.

6.3 Other rules

The multiplication rule corresponds to probabilities involving the word *and*, like the probability that *A* and *B* both happen. If we want to know the probability that *A* or *B* happens, we use the following rules:

Addition rule

1. If *A* and *B* are mutually exclusive (i.e. it is impossible for both to happen), then $P(A \text{ or } B) = P(A) + P(B)$.
2. In general, $P(A \text{ or } B) = P(A) + P(B) - P(A \text{ and } B)$.

Here are a few examples:

1. If we draw a single card from a deck, what is the probability that it is an ace or a heart?

There are 4 aces and 13 hearts in an ordinary 52-card deck. One of those hearts is the ace of hearts. (This is sort of double-counted, like we saw in the section on counting problems.) So the desired probability is

$$\frac{4}{52} + \frac{13}{52} - \frac{1}{52} \approx .308.$$

2. If we are dealt 5 cards from a deck, what is the probability they are all of the same suit (i.e., a flush)?

Starting with hearts, there are 13 hearts, so the probability all 5 cards are hearts is

$$\frac{13}{52} \cdot \frac{12}{51} \cdot \frac{11}{50} \cdot \frac{10}{49} \cdot \frac{9}{48} \approx .000495.$$

The same calculation works for the other suits. We want the probability that all the cards are hearts OR all of them are clubs OR all diamonds OR all spades, so we use the addition rule. Since these events are all mutually exclusive, the probability that the cards are of same suit is

$$.000495 + .000495 + .000495 + .000495,$$

or $4 \times .000495$, which is about .002 or 1 in every 500 hands, on average.

3. We flip a coin 3 times. What is the probability that at least one of the flips comes up tails?

“At least 1” means we could have 1 tail, 2 tails, or 3 tails, so we will have to add up the probabilities for each of those 3 possibilities. There are 8 outcomes in total from rolling a die, namely HHH, HHT, HTH, HTT, THH, THT, TTH, and TTT. Of these 3 have exactly one tail, 3 have exactly two tails, and 1 has exactly three tails. So the probability of at least one tail is

$$\frac{3}{8} + \frac{3}{8} + \frac{1}{8},$$

or $\frac{7}{8}$. Note that we could have just as easily done this using the basic rule, just noting that 7 of the 8 outcomes have a tail.

Negations

In the last example above, we considered the probability of getting at least one tail in three flips of a coin. Suppose we upped that to 10 flips of a coin? Computing the probabilities for exactly 1 tail, exactly 2, etc. up through exactly 10 tails would be tedious. Instead, we can look at the complement. The complement of at least one tail is getting no tails at all. That is something we can easily calculate. It is $(\frac{1}{2})^{10}$ or $\frac{1}{1024}$. Thus the probability of at least one tail is $1 - \frac{1}{1024}$ or $\frac{1023}{1024}$.

In general, we have the following rule:

Negation rule: $P(\text{not } A) = 1 - P(A)$.

Here are some more examples:

1. About 5% of people have red hair. If we meet three new people today, what is the probability none of them have red hair? Assume independence.

This is a very simple application of the rule. If 5% of people have red hair, then 95% don't. Thus the probability that all three don't have red hair is $(.95)^3$ or about 86%. Note that independence might not be a very good assumption here. Maybe the people you meet are related or all come from an area that has a lot of red-haired people.

2. About 1 in every 10,000 clover plants is a four-leaf clover. We spend all day examining plants, 6000 plants in total. What is the probability that we find a four-leaf clover?

We are looking for the probability of at least one four-leaf clover. So we might find one, two, three, etc. of them. Instead, we will look at the complement—finding no four-leaf clovers. That probability is $(\frac{9999}{10000})^{6000}$, which is approximately 55%. So there is a 55% chance we will *not* find a four-leaf clover, which means there is a 45% chance that we will find one.

We can take this one step further: How many clovers would we have to examine, in order to have a 95% chance of finding a four-leaf clover? To do this, we set up the equation $1 - (\frac{9999}{10000})^x = .95$. This is the same calculation as above, but with 6000 replaced by a variable and the right side set equal to our desired probability. We can solve this to get $x = \ln(.05) / \ln(\frac{9999}{10000}) = 29955.8$, so we need to examine nearly 30,000 clovers to have a 95% chance of finding a four-leaf clover.

6.4 Binomial probabilities

Suppose we roll a die 10 times. What is the probability of rolling a two exactly 3 times?

If we roll the 3 twos in a row at the start and then roll 7 other numbers, we could notate that as 222NNNNNNN, with N standing for a non-two. The probability of exactly this sequence of events occurring is $(\frac{1}{6})^3(\frac{5}{6})^7$.

But the twos could occur in many other orders, like 2NN2NN2NNN or NNN222NNNN. How many such orders are there? There are $\binom{10}{3}$ or $\frac{10!}{3!7!}$ ways. We can think of this as the number of 3-element subsets of a 10-element set, where the 3 elements are the positions of the twos. We could also think of this like the MISSISSIPPI problem from Section 5.2, where here we are looking at the number of ways to rearrange the characters of the word 222NNNNNNN.

In general, we have the following rule:

Binomial probability Suppose we have a situation where some event A can happen with probability p . If we repeat this with n independent trials, the probability that A occurs in exactly k of those trials is

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

Here are a couple of examples:

1. Suppose a student randomly guesses on a 20-question multiple choice test, where each answer could be a, b, c, or d. What is the probability that the student gets exactly 5 right?

This is a very typical binomial problem. Each guess is either right (probability $\frac{1}{4}$) or wrong (probability $\frac{3}{4}$), and we want exactly 5 right. We take $n = 20$, $k = 5$, and $p = 1/4$ in the formula to get

$$\binom{20}{5} \left(\frac{1}{4}\right)^5 \left(\frac{3}{4}\right)^{15},$$

which is approximately 20%.

2. A basketball player makes 77% of his free throws. Suppose he takes 10 shots. What is the probability the player makes exactly 9 of them?

We take $n = 10$, $k = 9$, and $p = .77$ to get

$$\binom{10}{9} (.77)^9 (.23)^1,$$

which is approximately 22%.

If we wanted the probability that the player makes at least 9, we would add the probability of exactly 9 and the probability of exactly 10 (which is $(.77)^{10}$) to get approximately 29%.

6.5 Expected value

The *expected value* is a way to compute an average of some outcomes weighted by their probabilities of occurring.

In general, if we have values x_1, x_2, \dots, x_n with probabilities p_1, p_2, \dots, p_n , respectively, of occurring, then the expected value is given by

$$p_1x_1 + p_2x_2 + \dots + p_nx_n.$$

Here are a few examples:

1. Suppose an investment of \$10,000 can turn out one of three ways: there is a 50% chance it drops to \$9000, a 30% chance it increases to 10,500, and a 20% chance it increases to \$13,000.

The expected value is

$$(.5)(9000) + (.3)(10500) + (.2)(13000) = 10250.$$

Thus, on average, the investment results in a \$250 gain. This does not apply to one investment, but rather it is a long-term average. This would probably be a good investment for a company that makes lots of investments.

2. Consider the following game: A random number between 1 and 10000 is generated. If the number generated is not 1234, then you win \$100. If it is 1234, then you lose one million dollars.

The expected value of the game is

$$\frac{9999}{10000}(100) + \frac{1}{10000}(-1000000) = -.01.$$

So you lose about a penny per game on average. If you play this game a few times, you will almost certainly win every time, coming away with a few hundred dollars. But if you play this game lots of times, you will eventually lose, and that big loss will offset a lot of winnings.

If this were a casino game, with millions of people playing this game every year, then the negative expected value means that the casino will make money on average. If 10 million people played this game, the casino would make somewhere around \$100,000.

3. Expected values are used for a lot more than just money. For instance, suppose a computer program can take 18, 19, 20, 21, or 22 milliseconds to run, with the probabilities of these being .03, .15, .44, .30, and .08, respectively. What is the expected value?

The answer is

$$(.03)(18) + (.15)(19) + (.44)(20) + (.30)(21) + (.08)(22) = 20.25 \text{ milliseconds.}$$

We can think of this as a sort of average running time for the program.

6.6 Bayes' theorem

Consider the following problem: A company buys 90% of its chairs from Supplier X and 10% from Supplier Y. Suppose that 15% of X's chairs are defective and 45% of Y's chairs are defective. Suppose my chair breaks. What is the probability it was made by Supplier X?

There are two competing influences here. On the one hand, Y's chairs are three times as likely as X's to break. On the other hand, we have many more chairs made by X than we do chairs made by Y. One approach to this type of problem is to make a table. Suppose we have 1000 chairs. Then 900 of them are made by X and 100 by Y. Of those 900 of X's chairs 135 (15% of 900) are defective. Of the 100 made by B, 45 (45% of 100) are defective. So we have 180 total defective chairs, of which 135 are made by X. So the probability that the defective chair is one of X's is 135/180 or 75%.

There was nothing special about choosing 1000 chairs to start with. We could have chosen any amount to work with. If we abstract the argument above, we get the following rule, known as *Bayes' theorem*:

Bayes' theorem:
$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|A^c)P(A^c)}$$

In this formula A^c denotes the complement of A . In the chair example above, we have A being the event that a chair is made by Supplier X and B being the event that the chair is defective. So $P(A|B)$ is what we are looking for, the probability that the chair was made by Supplier X given that it is defective.

Bayes' theorem allows us to turn this around and use $P(B|A)$, which is the probability the chair is defective given it is made by X. We also need $P(A)$, $P(A^c)$ and $P(B|A^c)$. In summary, we have

Term	Description	Value
$P(B A)$	probability of a defective given it is made by X	.15
$P(B A^c)$	probability of a defective given it is made by Y	.45
$P(A)$	probability of a chair being one of X's	.90
$P(A^c)$	probability of a chair being one of Y's	.10

Putting all this together, we get

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|A^c)P(A^c)} = \frac{(.15)(.90)}{(.15)(.90) + (.45)(.10)} = .75.$$

Another example

As another example, suppose we have a test for a disease that is 95% accurate. That is, if a person has the disease, there is a 95% chance that the test will detect it and if they don't have the disease, there is a 95% chance that the test tells them they don't have it. Suppose further that 1% of the population has the disease. Given a positive test result, what is the probability of actually having the disease?

Let A be the event of having the disease and B be the event of a positive test result. So we are looking for $P(A|B)$. We have the following

Term	Description	Value
$P(B A)$	probability of a positive for someone with the disease	.95
$P(B A^c)$	probability of a positive for someone who doesn't have the disease	.05
$P(A)$	probability of someone having the disease	.01
$P(A^c)$	probability of someone not having the disease	.99

Then by Bayes' theorem we have

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|A^c)P(A^c)} = \frac{(.95)(.01)}{(.95)(.01) + (.05)(.99)} = .161.$$

So, given a positive test result, there is only a 16% chance of actually having the disease.

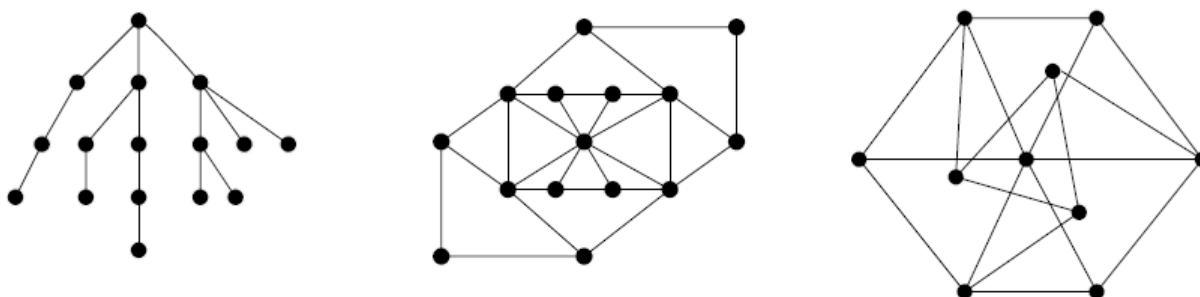
Numerically, in a population of 1,000,000, there are 10,000 people that have the disease. Of them, 9500 will get a positive test result. On the other hand, there are 990,000 people that don't have the disease, of whom 5%, or 49,500 will have a positive test result. We call these *false positives*, because they have a positive result even though they don't have the disease. We see that the false positives overwhelm the results to the point that of the 59,000 people with a positive test result, only 9500, or about 16%, actually have the disease.

On the other hand, a *false negative*, a negative test result for a person that actually has the disease, is a lot less likely. Of the 10,000 with the disease, 500 will get a negative result, while of the 990,000 that don't have the disease, 940,500 will get have a negative result, so that if you get a negative result, there is a $940500/(940500 + 500) = .9995$ probability that you don't have the disease.

Chapter 7

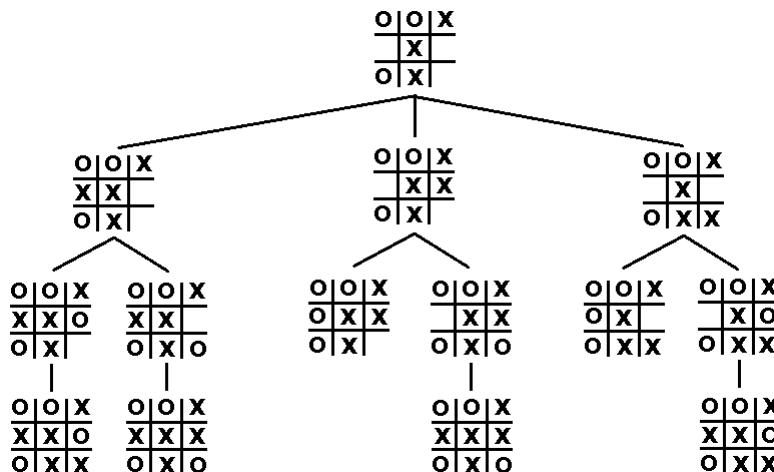
Graphs

Graphs are networks of points and lines, like in the figure below:



The points are called *vertices* or *nodes*, and the lines are called *edges* or *arcs*. A lot of real-life problems can be modeled with graphs. For example, we can model a social network with a graph. The vertices are the people and there is an edge between two vertices if their corresponding people know each other. Asking questions about the social network, like the average number of people that people know or the average number of connections it takes to get from any one person to another can be answered by computing properties of the graph.

Another example of a use of graphs is modeling games. In the game Tic-tac-toe, we can create a game tree to represent the game. The vertices are the states of the game and there is an edge between states if there is single move that gets us from one state to another. Part of the tree is shown below.



Having represented the game this way, a computer player can scan through the game tree to find an optimal move. More complicated games, where it is possible to go back to a previous state can be represented by more

general graphs.

One more example involves scheduling. Suppose we have several people who have to be at certain meetings at a conference. We want to schedule the meetings so that everyone who has to be at a meeting can be there. The main constraint then is that two meetings can't be scheduled for the same time if there is someone who has to be at both. We can represent this as a graph if we let the vertices be the meetings, with an edge between two vertices if there is someone that has to be at both meetings. We then have to assign times to the vertices such that vertices that are connected by an edge must get different times. This is an example of what is called *graph coloring* and it can be used to model a lot of problems like this. If we implement an algorithm to find proper colorings of graphs, then all we have to do is represent our real-life problem with a graph and then run the algorithm on that.

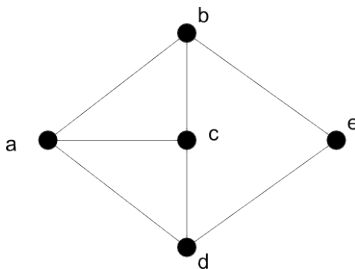
In general, to model a problem with a graph, identify the fundamental units of the problem (people, game states, meetings) and the fundamental relationship between the units (acquaintance, can get from one state to the next by a single move, one person has to be at both meetings).

7.1 Terminology

As mentioned, the dots can be called vertices or nodes and the lines can be called edges or arcs. Different authors use different terminology. We will stick with vertices and edges. Here is a list of the most important terms:

- If there is an edge between two vertices, we say the vertices are *adjacent*.
- The *neighbors* of a vertex are the vertices it is adjacent to. The *neighborhood* of a vertex is the set of all its neighbors.
- The *degree* of a vertex is the number of neighbors it has.
- A *path* in a graph is an alternating sequence of vertices and edges, with no repeated vertices or edges.
- A *cycle* in a graph is a path that starts and ends at the same vertex.
- A graph is said to be *connected* if it is possible to get from every vertex to every other vertex by a path.

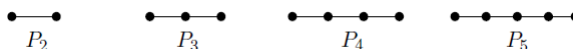
To illustrate these definitions, consider the graph below.



The vertex c has three neighbors: a , b , and d . Therefore it has degree 3. On the other hand, vertex e has two neighbors, c and d , so it has degree 2. Some example paths are abe and $acdeb$. Some example cycles are $acba$, $adcba$, and $adeba$.

Here are some of the more common types of graphs:

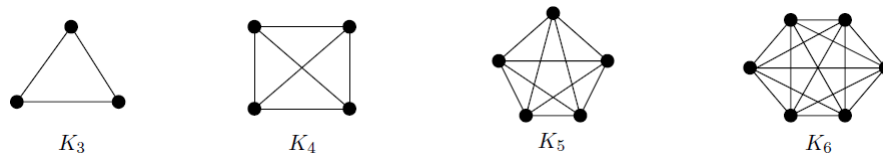
- Paths — all vertices laid out in a line



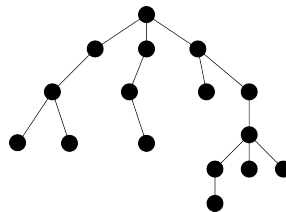
- Cycles — polygon shape



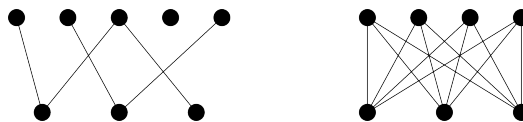
- Complete graphs — every vertex connected to every other vertex



- Trees — connected graphs that have no cycles.

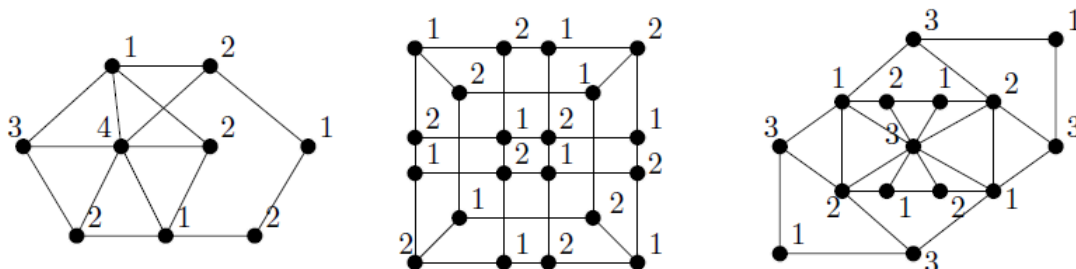


- Bipartite graphs — The vertices of a bipartite graph are in two groups. There are no edges possible within the groups; the only possible edges are between the two groups. A *complete bipartite graph* is one that has all possible edges between the two groups. On the left below is an ordinary bipartite graph and on the right is a complete bipartite graph.



7.2 Graph coloring

Graph coloring is a process where we assign colors to the vertices of a graph so that adjacent vertices get different colors. The goal is to use as few colors as possible. The name *coloring* is partly historical. Usually people use positive integers for color names. Here are a few examples.



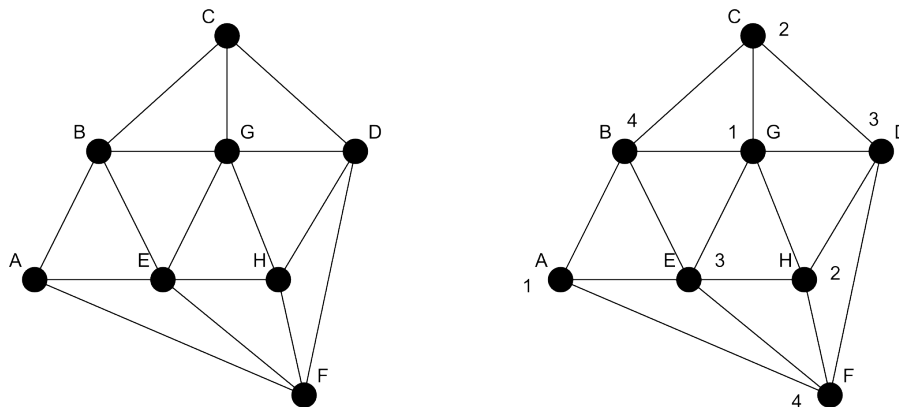
Notice in each example above that vertices that are adjacent receive different colors. Also notice that we have used the minimum possible number of colors. It would not be possible to use fewer colors without breaking the rule about adjacent vertices getting different colors.

Graph coloring can be used to solve a variety of problems. Here is one example:

Eight chemicals are to be shipped in containers. We want to use as few containers as possible. The only problem is that some chemicals react with certain other chemicals, and chemicals that react must be in different containers. Use graph coloring to find the minimum number of containers. The reactions are shown below.

- A reacts with B, E, and F
- B reacts with C, E, and G
- C reacts with D and G
- D reacts with F, G, and H
- E reacts with F, G, and H
- F reacts with H
- G reacts with H

We formulate this as a graph coloring problem by letting the vertices be the chemicals, the containers be the colors, and we have an edge between two vertices if their corresponding chemicals react. This way, assigning colors to the vertices will be like assigning containers to the chemicals. Since we have an edge between reacting chemicals, we are guaranteed that two chemicals won't be assigned the same container. The graph is shown below along with a proper coloring.



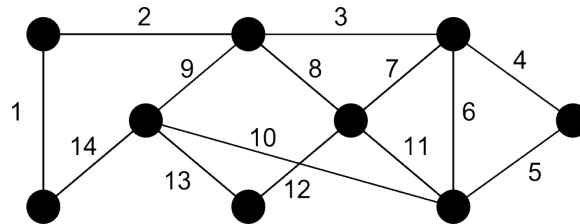
We see that 4 containers are required. Container 1 holds chemicals A and G. Container 2 will hold C and H. Container 3 holds D and E, and container 4 holds B and F.

We could not get away with fewer containers as the Cycle BCDHE requires 3 colors and G must be a different color than all of them since it is adjacent to all of them.

In general, the colors are the things you are assigning, the vertices are the things being assigned to, and the edges represent the relationships that would prevent two vertices from being assigned the same color.

7.3 Eulerian tours

An *Eulerian tour* (or *Eulerian circuit*) in a graph is a trip through the graph that includes every edge exactly once, starting and ending at the same vertex. Shown below is an example Eulerian tour starting and ending at the lower left vertex. The order in which the edges are traced is indicated by the numbers.



It turns out to be remarkably simple to tell if a graph has an Eulerian tour:

Criterion for Eulerian tours — A graph has an Eulerian tour if and only if every vertex has even degree.

This is an *if and only if* statement, so it is actually two statements in one:

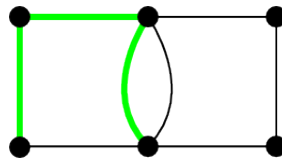
1. If every vertex degree is even, then there is an Eulerian tour.
2. If any vertex has odd degree, then there is no Eulerian tour.

It is not too hard to see why the criterion works: Imagine starting at a vertex with degree 3. We start our tour by leaving the vertex, using up one of the three edges out of the vertex. Later on, we come back into the vertex and leave again, using up the remaining two edges and leaving ourselves with no way to get back to the vertex. A similar argument works in general for any odd degree regardless of whether the vertex is at the start, end or in the middle of the tour.

The criterion above tells us when there is an Eulerian tour, but it doesn't tell us how to find one. However, there is a pretty simple algorithm, called Fleury's algorithm, for finding one. Roughly speaking, it says the following:

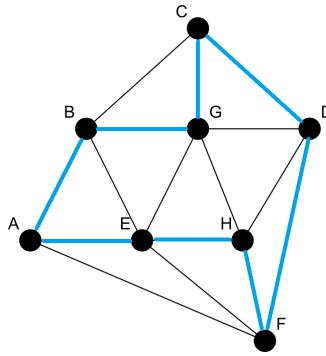
Start anywhere and walk along the graph, at each vertex choosing our next edge to be one that we haven't already visited, such that visiting it will not prevent us from getting to an unvisited portion of the graph.

We have a lot of freedom in terms of the order of edges to visit. The one thing to be careful about is shown below. Suppose we start at the bottom left and after three steps we are at the middle bottom. We must not turn left at this point, as doing so would cut off the right side of the graph, preventing us from ever getting to it.



7.4 Hamiltonian cycles

A *Hamiltonian cycle* is a cycle in a graph that includes every vertex (exactly once). A Hamiltonian cycle is highlighted in the graph below.



Unlike Eulerian tours, finding Hamiltonian cycles is not easy. There is no known fast and simple if and only if condition like with Eulerian tours.¹

However, there are some theorems that work in one direction. For example, we have the following theorem:

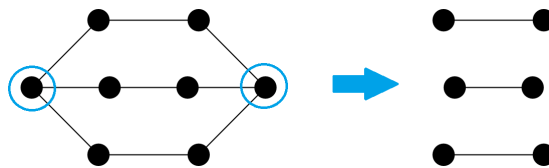
If G is Hamiltonian, then for any subset S of the graph's vertices if S and all the edges into S are deleted, then the resulting graph has no more than $|S|$ connected components.

A connected component of a graph is sort of like a “piece” of the graph, where all the vertices in the piece are connected to each other but not connected to any vertices in the rest of the graph.

This condition is a necessary condition for G to be Hamiltonian. The way we use a necessary condition is we use its contrapositive, namely that if G doesn't satisfy the condition about those sets S , then it has no Hamiltonian cycle. This gives us the following test:

If we can find a subset of vertices of the graph which, when deleted, breaks the graph into more pieces than there were vertices in the subset, then the graph has no Hamiltonian cycle.

For instance, if deleting two vertices (and their edges) from a graph breaks the graph into three pieces, then we can be sure there is no Hamiltonian cycle. For example, deleting the two circled vertices in the graph below on the left breaks the graph into three pieces. So there is no Hamiltonian cycle.

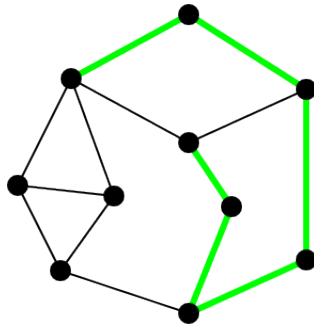


A second way to tell if a graph has no Hamiltonian cycle uses the following fact:

If a vertex has degree 2, then both of its edges must be part of any Hamiltonian cycle.

This is because there is only one way in and one way out of a vertex of degree 2, so any Hamiltonian cycle must use those edges. In the figure below, the edges into vertices of degree 2 are highlighted. All of those edges must be on a Hamiltonian cycle. But that makes it impossible to include the vertices on the lower left in the cycle. So there must not be a Hamiltonian cycle.

¹Finding one or proving that one doesn't exist would actually solve the P=NP problems, one of the most famous and difficult problems in mathematics (with a \$1,000,000 prize attached to it).



To show that a graph has a Hamiltonian cycle, one way is to just find a cycle, but if the graph is large, then this isn't so easy. There are some theorems that guarantee the presence of a Hamiltonian cycle. For instance, we have the following:

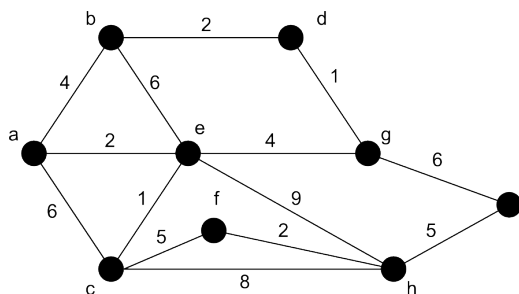
Let G be a graph with at least three vertices and no loops (vertices adjacent to themselves) or multiple edges (edges that share the same start and end point). If each vertex of G is adjacent to at least half of the other vertices, then G has a Hamiltonian cycle

In short, if there are enough routes into and out of every vertex, then a graph will have a Hamiltonian cycle. This is an example of a sufficient condition for a Hamiltonian cycle. Any graph that passes this condition is guaranteed to have a Hamiltonian cycle, but a graph could fail this condition and still have a Hamiltonian cycle (for example, a cycle on 6 or more vertices fails the condition yet clearly has a Hamiltonian cycle).

There are many more sufficient conditions and necessary conditions for Hamiltonian cycles. See a book on graph theory for more. Hamiltonian cycles have a variety of applications to real-world problems. A famous one is the traveling salesman problem, where a salesman needs to visit a number of cities and return home, doing so in the cheapest way possible.

7.5 Weighted graphs and minimum spanning trees

There is one more type of graph that we will consider, called a *weighted graph*, which is like an ordinary graph except that the edges are given weights. We can think of edge weights as the cost of using edges. Here is an example:



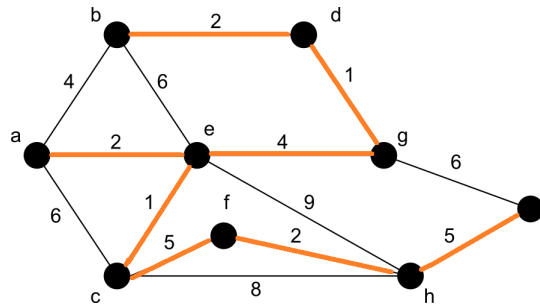
Now suppose the vertices of a weighted graph represent cities and the edges represent the cost of building a road from one city to another. Let's say we want to connect up all the cities so that it is possible to get from any city to any other (possibly passing through some other cities) and we want to do so as cheaply as possible. How can we do this?

What we are looking for is called a *minimum spanning tree*—*minimum* for least total cost, *spanning* since we have to include all the vertices, and *tree* because it should have no cycles. A cycle would mean we have redundant edges; we could remove any one of the cycle's edges and still be able to get from any city to any other.

One algorithm for finding a minimum spanning tree is called *Kruskal's algorithm*. The idea is so simple it is almost surprising that it actually works.

Kruskal's Algorithm — Start by choosing the edge of least weight. Then, at each stage, choose the edge of least weight that hasn't already been chosen and that doesn't create a cycle among the edges already chosen. Stop when all the vertices are connected. Break any ties arbitrarily.

In the graph below, we would start by adding edges ce and dg of weight 1. We then add edges bd , ae , and fh , each of weight 2. Then we add edge eg of weight 4. We don't add edge ab of weight 4 because that would create a cycle. Alternatively, we could have added ab and not eg . Either way would work. There are often several spanning trees of the same total weight. Finally, we add edges ci and hi to get a minimum spanning tree of total weight 22. No other spanning tree has a smaller weight.



Kruskal's algorithm is an example of what is known as a *greedy algorithm*. It is called greedy because at each step we take the best option in terms of cost (that satisfies the rules of the problem), not worrying about the consequences of our actions at all. Greedy algorithms don't always produce an optimal solution, but sometimes, as in the case of minimum spanning trees, they do.

There is another greedy algorithm for finding a minimum spanning tree, called *Prim's algorithm*. It works similarly, but where Kruskal's algorithm adds the cheapest working edge from anywhere on the graph, Prim's algorithm builds up the tree outward from a single starting vertex. Here is the idea of how it works:

Prim's algorithm — Pick any starting vertex on the graph. Choose the cheapest edge from that vertex. Then, at each stage, pick the cheapest edge from a vertex that has already been reached to an edge that has not yet been reached. Stop when all the vertices have been reached. Break any ties arbitrarily.