

Chapter 1: Introduction / Proofs

Weeks 1-2

UoA 2018

Welcome to Compsci 225! This is a set of typeset notes that are meant to serve as a companion to the lectures this term. Think of them as a slightly “remixed” version of the coursebook; in these notes, you’ll find the same concepts as in the coursebook, but with new examples and a bit more detail in the proofs.

If you have any questions, please email me at

padraic.bartlett@auckland.ac.nz

In particular, let me know if you spot any typos! I’ll do my best to avoid them, but given that I am writing these notes simultaneously with the lectures, some will certainly creep in over time. (As a tiny incentive, I’ll give the first person to spot any mathematical typo a chocolate fish! Email me to claim your reward whenever you spot such a typo.)

1.1 What Is Mathematics?

Compsci 225 (as stated in on the opening slides) is a course on mathematics. So, let’s start from the basics: what *is* mathematics?

If you ask¹ a random person on the street what they think mathematics is, I’d bet that they would say something like this:

“Mathematics is how you **calculate** things.”

In other words, most people in the world think mathematics, more or less, is a field in which you learn how to solve equations: i.e. a field in which you just write stuff like

$$\begin{aligned} 1 + 1 &= 2, \\ \sqrt{529} &= 23, \\ x^2 - 9x + 20 &= (x - 4)(x - 5), \text{ and} \\ \int_{-3}^3 \frac{\sqrt[3]{x}}{\sqrt[3]{3-x} + \sqrt[3]{3+x}} dx &= 0 \end{aligned}$$

They’re not wrong! Mathematics certainly involves lots of calculations, and in this class you’ll learn how to calculate lots of things:

- The prime factors of any integer,
- The chromatic number of a graph,
- The number of ways to choose n objects from k possibilities with and without repetition, and
- The accepting states of a finite automata, amongst others!

However, if **all** you know to do is calculate things, you’re doomed to be essentially a slower and fuzzier version of [Wolfram Alpha](#); this is not exactly a great thing to aspire to, or a particularly strong reason to learn mathematics. And yet some level of mathematics is required for a major in almost any subject under the

¹Well, first they’d probably look at you funny. But after a minute, I claim they’d say something like this

BSc here at Auckland (and indeed, for undergraduate study in almost any scientific field at almost any university!) This is because to most scientists, mathematics isn't just a bunch of calculational techniques: instead,

“Mathematics is a **tool** for solving problems.”

Gauss, one of the most prolific² scientists of all time, wrote that “**Mathematics is the Queen of the Sciences;**” by this, he meant that the framework and techniques of mathematics are integral and indispensable to the pursuit of almost every other scientific field, and that in this sense mathematics “rules” over everything. This is **not an isolated view**; the “**unreasonable effectiveness of mathematics**” has been something that scientists have been surprised by and relied on for generations.

If you want to **launch a rocket**, or **manage an economy**, or **design a processor**, you need mathematics.

At the University of Auckland itself, people in the applied mathematics unit are producing ground-breaking³ research on **pigeon navigation**, **modelling calcium flows across cellular membranes**, and **earthquakes** using mathematics.

Less seriously, you can use mathematics as a tool in lots of day-to-day things, like playing games! Here's a problem I encountered when playing Pokémon last year:

Congratulations; you're a Pokémon trainer! As such, you have a Pikachu that you've been training so that you can be the very best⁴. The land outside of your town is infested with Spearow; to prepare for the attacks you'll face when you leave the town, you've been improving your Pikachu's **health** (H) and **defense** (D) values. You have 200 points to spread out between these two stats: that is, $H + D = 200$. When you leave the town, you know that the number of attacks you can take from Spearow on the way to the next town is approximately⁵ $\frac{(1100 + H)(500 + D)}{60000 + 40D}$. What should you make H and D equal to to maximize the number of attacks you can handle? What is the maximum number of attacks you can encounter?

Pretty much every game out there has some sort of mechanic like the above (try to find one in a game you're familiar with!); while I don't recommend that you pull out a graphing calculator in the middle of a League of Legends match, it's still a fun way in which you can see how mathematics is a tool that lets you solve problems!

Both of these ideas about what mathematics “is” capture some very important aspects of our field. However, I claim that if you talk to your maths teachers back in high school, they might come up with a third definition:

“Mathematics is a **language**.”

That is: mathematics isn't just a field in which you learn to do calculations, it's one in which you are learning a **language**, that lets you describe things accurately and precisely!

That is: when you see an expression like $\frac{(1100 + H)(500 + D)}{60000 + 40D}$, you know that the long horizontal line means division, and that

²No, really: check out [this list](#) of things with Gauss's name on them.

³In the last case, this is meant quite literally.

the $+$ symbol refers to addition. You know that you perform the calculations inside the parentheses before the other calculations, and that the letters H and D are variables that you could substitute in all sorts of real numbers in for; you also know that D had better not be equal to $-\frac{60000}{40}$, otherwise the denominator will be 0 and division by 0 is not allowed.

In other words, you know how to translate all of the symbols in that equation into words and concepts, and moreover know how those concepts and words interact! You know the “grammar” of mathematics, which would tell you that something like $\frac{5x \cdot (\div 3)}{\sqrt{+}}$ makes no sense, and that you should always have “ $+C$ ” at the end of your indefinite integrals.

Even though most classes you’ll take at University don’t specifically call this out, I would claim that you spend most of your time in mathematics classes here working with mathematics on this “language” level! That is: in CS225 and any other mathematics class you take, you will see lots of new terms, concepts and notation. Learning this language and practicing “speaking” it (i.e. writing mathematics on your assignments, and working with each other in tutorials) will be one of the main goals of CS225, and indeed of any University career in mathematics!

In contrast to the above three ideas of mathematics, however, the attitude of many research mathematicians to their field is perhaps best summed up in the XKCD comic at left. To put it a bit more elegantly:

“Mathematics is the art of abstraction.”

We don’t study mathematics just because we care about rockets or earthquakes or cells, we study mathematics because⁶ we care about mathematics itself! **G.H. Hardy**, one of the most influential number theoreticians of all time, described his own work as follows: “No discovery of mine has made, or is likely to make, directly or indirectly, for good or ill, the least difference to the amenity of the world.” This, by the way, was something he was incredibly proud of; as an Englishman who saw the first world war he was horrified at what people did by applying the mathematics of his peers, and prided himself on studying only things that would never possibly have the same effects on the world.

In this world of abstraction, we study questions like

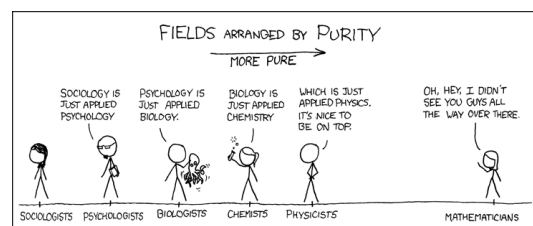
“How many solutions are there to the equation $x^2 + y^2 = z^2$, where x, y, z are integers?”

and

“How many solutions are there to the equation $x^n + y^n = z^n$, where n, x, y, z are integers and $n \geq 3$?”

These aren’t problems with obvious applications, nor are they things that we can directly calculate; instead we have to grapple with these problems in our minds, and come up with clever arguments and constructions that let us see the core of what’s being asked.

For the first, for instance, the answer is “infinitely many:” if we’re clever and think about this for a while, you can eventually notice that for any $m, n \in \mathbb{Z}$, $x = m^2 - n^2, y = 2mn$ gives us⁷ a pair of



<https://xkcd.com/435/>

⁶Though it doesn’t hurt to know about applications, if only so you can convince people to give you money to let you study more mathematics.

⁷If you haven’t seen this before: writing $x \in A$ is shorthand for saying “ x is an object in the set A .” For example, $1 \in \mathbb{N}$ and $\pi \in \mathbb{R}$, but $\sqrt{2} \notin \mathbb{Z}$.

values such that

$$\begin{aligned}x^2 + y^2 &= (m^2 - n^2)^2 + (2mn)^2 = m^4 - 2m^2n^2 + n^4 + 4m^2n^2 \\&= m^4 + 2m^2n^2 + n^4 \\&= (m^2 + n^2)^2,\end{aligned}$$

and therefore that setting $z = m^2 + n^2$ gives us infinitely many solutions to this equation.

The answer to the second question, if you're curious, is that “no solutions exist.” This result is called **Fermat's Last Theorem**; this problem was first formulated in 1637 by the mathematician Pierre de Fermat, and proven in 1995 by the mathematician Andrew Wiles, after 358 years of constant work by mathematicians; this problem is famous for attracting the largest number of recorded incorrect solutions of any mathematical theorem, and is one of the greatest⁸ feats of human ingenuity in the past century.

With that said, I should clarify a little bit and say that all of these viewpoints are correct! In other words,

“Mathematics is **everything**.”

All mathematicians calculate things, all mathematicians use notation and the language of mathematics to describe their results, all mathematicians either work with applications or eventually have their work used in applications⁹, and the art of abstraction is needed to understand and create almost any truly deep or interesting result.

Throughout this class, we'll see mathematics in all of these forms.

For the rest of this section, though, I want to start exploring one idea that's central to mathematics in any of its guises: the idea of **proof**.

1.2 What is a Proof?

Every major field of study in academia, roughly speaking, has a way of “showing” that something is true. In English, if you wanted to argue that the whale in Melville's *Moby Dick* was intrinsically tied up with mortality, you would write an essay that quoted Melville's story alongside some of his other writings and perhaps some contemporary literature, and logically argue (using these quotations as “evidence”) that your claim holds. Similarly, if you were a physicist and you wanted to show that the speed of light is roughly $3.0 \cdot 10^8$ meters per second, you'd set up a series of experiments, collect data, and see if it supports your claim.

In mathematics, a **proof** is an **argument** that mathematicians use to show that something is true. However, the concepts of “argument” and “truth” aren't quite as precise as you might like; certainly, you've had lots of “arguments” with siblings or classmates that haven't proven something is true!

In mathematics, the same sort of thing happens: there are many arguments that (to an outsider) look like a convincing reason for why something is true, but fail to live up to the standards of a mathematician. To help us understand the idea of proof, we look at two such “failed” proofs here:

⁸So, just a bit harder than the first question. It is surprising what changing $n = 2$ to $n \geq 3$ can do to a problem.

⁹Number theory, while seemingly useless in Hardy's time, is indispensable to the entire modern field of cryptography and computer security. While Hardy was alive, his dream that his work would not have the faintest practical use was largely intact; at the dawn of the twenty-first century, however, the opposite is true. The battlefield that most international skirmishes are taking place on today is online, the weapons of choice are cryptosystems, and it would be difficult to find a cryptographer alive that hasn't read Hardy's work.

Claim. The sum of any two odd numbers is even.

“Bad” proof: Well, $1+1=2$ is even, $3+7=10$ is even, $-13+5=-8$ is even, and $1001+2077=3078$ is even. Certainly seems to be true!

□

A defense of the “bad” proof: This might seem like a silly argument, but suppose we’d listed a thousand examples, or a billion examples. In most other fields of study, that would be enough to “prove” a claim! (Think about science labs: there, we prove claims via experimentation, and any theory you could test a billion times and get the same result would certainly seem very true!)

Why this proof is not acceptable in mathematics: In mathematics, however, this isn’t good enough. When we make a claim about “any” number, or say that something is true for “all” values, we really mean it: if we have not literally shown that the claim holds for every possible case, we don’t believe that this is a proof!

This is not just because mathematicians are fussy. In the world of numbers, there are tons of “eventual” counterexamples out there:

- Consider the following claim: “The sequence of numbers “12, 121, 1211, 12111, 121111, ...” are all not prime¹⁰.”

If you were to just go through and check things by hand, you’d probably be persuaded by the first few entries: $12 = 3 \cdot 4$, $121 = 11 \cdot 11$, $1211 = 1737$, $12111 = 367 \cdot 11$, $121111 = 431 \cdot 281$, ...

However, when you get to $12 \overbrace{111 \dots 1}^{136 \text{ 1's}}$, that one’s prime! This is well beyond the range of any reasonable human’s ability to calculate things, and yet something that could come up in the context of computer programming and cryptography (where we make heavy use of 500+ digit primes all the time.)

- Here’s another claim: “For any integer n , the two numbers $n^{17} + 9$ and $(n+1)^{17} + 9$ have no factors¹¹ in common.”

For example, when $n=0$ this claim says that $(0^{17}+9)=9$ and $(0+1)^{17}+9=10$ have no factors in common, which is true: $9=3 \cdot 3$ and $10=2 \cdot 5$. Similarly, when $n=1$ this says that $1^{17}+9=10$ and $(1+1)^{17}+9=131081$ have no factors in common; this is also true, as $10=2 \cdot 5$ while $131081=19 \cdot 6899$ (which are both primes, and so we can’t break this down further.)

This gets hard for humans to calculate *very* quickly, so I’ll spare you any more calculations and skip to the punchline: this pattern holds for all n until $n=8424432925592889329288197322308900672459420460792433$, where it fails. “_(?)/”

- In general, mathematics is full of things like this!

Instead, I claim that a “good” proof of this claim would go as follows:

Claim. The sum of any two odd numbers is even.

Proof. Take any two odd numbers N, M . By definition, because N and M are odd, we can write $N=2k+1$ and $M=2l+1$, for two integers k, l .

¹⁰A positive integer is **prime** if it only has two positive integer factors: 1 and itself. For example, 2, 3, 5, 7, 11, 13 are all prime, while numbers like 4, 6, 8, 9, 10, ... are all not prime. Note that 1 is not prime, as it doesn’t have two positive integer factors: it only has one such factor, namely itself!

¹¹So, if one of those numbers was a multiple of 3, the other one wouldn’t be; if one was even, the other would be odd, that kind of thing.

Therefore, $N + M = (2k + 1) + (2l + 1) = 2k + 2l + 2 = 2(k + l + 1)$. In particular, this means that $N + M$ is an even number, as we've written it as a multiple of 2! \square

Here are some key aspects of this “better” proof, to consider when writing your own proofs:

- We worked in *general*: that is, we didn't just look at a few examples, but instead considered arbitrary values!
- We defined our variables, and described what sets they come from.
- We used words to describe what we were doing and why it worked!

Here's another way in which a proof might “fail” us:

Claim. Given any two nonnegative real numbers x, y , we have $\frac{x+y}{2} \geq \sqrt{xy}$.

“Bad” proof:

$$\begin{aligned}\sqrt{xy} &\leq \frac{x+y}{2} \\ xy &\leq \frac{(x+y)^2}{4} \\ 4xy &\leq (x+y)^2 \\ 4xy &\leq x^2 + 2xy + y^2 \\ 0 &\leq x^2 - 2xy + y^2 \\ 0 &\leq (x-y)^2.\end{aligned}$$

\square

A defense of the “bad” proof: We're not using examples; we're working in general! Also, we *totally* showed that this claim is true: after all, we started with our claim and turned it into a true thing!

Why this proof is not acceptable in mathematics:

- We have no idea what x and y are! In particular, by plugging in some sample values of x and y , we can see that this is sometimes true and sometimes false: for $x = 1, y = 4$ we do indeed have $\sqrt{xy} = \sqrt{4} = 2 \leq \frac{1+4}{2} = 2.5$, but for $x = -1, y = -1$ the claim $\sqrt{(-1) \cdot (-1)} \leq \frac{-1-1}{2}$ is very false, as $-1 \not\leq 1$! So, to do anything here, we first need to know what x and y **are**. That is: we need to define what set x, y come from!
- This proof is “backwards:” that is, it starts by assuming our claim is true, and from there gets to a true statement. This is not a logically sound way to make an argument! For example, if we assume that $1=2$, we can easily deduce a true statement by multiplying both sides by 0:

$$\begin{aligned}1 &= 2 \\ \Rightarrow 0 \cdot 1 &= 0 \cdot 2 \\ \Rightarrow 0 &= 0.\end{aligned}$$

This doesn't prove that $1=2$, though! As we said above, proofs need to **start** with true things, and then through argument **get to** what you're trying to show.

- Finally, this proof has no words! This flaw in some sense is why the other two flaws could exist: if you had to write out in words what x and y were, and how you went from one line to the next, it would probably become clear that this proof was written backwards and also that we have to be careful with what x, y are allowed to be.

This sort of thing is often easy to fix, though! If your proof is “backwards,” simply try starting from the end and reasoning your way backwards to the start. If your logic was flawed, somewhere along the way you’ll encounter a nonreversible step.

For example, if we tried to reverse our proof that $1 = 2$, we could go from $0 = 0$ to $0 \cdot 1 = 0 \cdot 2$, but would see that we can’t “divide by 0” to get to the desired conclusion (and thus that this doesn’t work.)

With this in mind, let’s try a “fixed” version of this proof:

Theorem 1. (*The arithmetic mean-geometric mean inequality.*)
For any two nonnegative real numbers x, y , we have that the geometric mean of x and y is less than or equal to the arithmetic mean of x and y : in other words, we have that

$$\sqrt{xy} \leq \frac{x + y}{2}.$$

Proof. Take any pair of nonnegative real numbers x, y . We know that any squared real number is nonnegative: so, in specific, we have that the square of $x - y$, $(x - y)^2$ is nonnegative. If we take the equation $0 \leq (x - y)^2$ and perform some algebraic manipulations, we can deduce that

$$\begin{aligned} 0 &\leq (x - y)^2 \\ \Rightarrow 0 &\leq x^2 - 2xy + y^2 \\ \Rightarrow 4xy &\leq x^2 + 2xy + y^2 \\ \Rightarrow 4xy &\leq (x + y)^2 \\ \Rightarrow xy &\leq \frac{(x + y)^2}{4}. \end{aligned}$$

Because x and y are both nonnegative, we can take square roots of both sides to get

$$\sqrt{xy} \leq \frac{|x + y|}{2}.$$

Again, because both x and y are nonnegative, we can also remove the absolute-value signs on the sum $x + y$, which gives us

$$\sqrt{xy} \leq \frac{x + y}{2},$$

which is what we wanted to prove. \square

1.3 The Language of Proof: Propositional Logic

While the two examples and nonexamples of proofs above are good for getting a “gut feeling” for mathematical proof, to actually *write* proofs we’ll need to build up some language and concepts. To do this, let’s start with the fundamentals:

Definition. A **proposition** (or statement, or claim) is just something that is either true or false. For example, the following are propositions:

- P = “Every even number greater than 2 can be expressed as the sum of at most six primes” is a proposition; this one happens to be true (a result in number theory, proven in 1995 by the French mathematician **Olivier Ramaré**.)
- Q = “Every even number can be expressed as the sum of two primes” is another proposition; this one is false, as the number 2 cannot be expressed as the sum of two other primes (as there are no prime numbers smaller than 2.)

- R = “Every even number greater than 2 can be expressed as the sum of two primes” is a third proposition; this is Goldbach’s conjecture, a famous open problem in number theory. It is either true or false, but mathematicians have not yet discovered which.

Conversely, the following are *not* propositions:

- P = “What color is this pen?” This proposition evaluates to a color (i.e. “green,” “red”) and not something that is true or false.
- Q = “This sentence is false.” This proposition is a paradox (if it were true, it would be false, and if it were false, it would be true), and so we cannot assign it a single truth value.
- R = “Hello!” This proposition doesn’t evaluate to true or false; it’s just a declaration.

Often, we will work with mathematical propositions that depend on a variable. For example, we can write

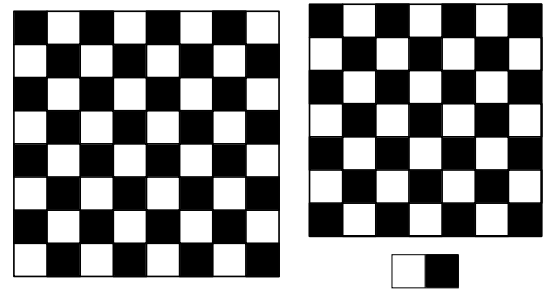
$P(n)$ = “A $n \times n$ checkerboard can be covered by nonoverlapping 2×1 dominoes;”

this statement will be false for odd values of n , and true for even values of n (if you don’t see why, prove this!)

Definition. Given some propositions, we will often want ways to combine them into new propositions. The following list contains some of the most common combinations:

1. Given two mathematical propositions P and Q , we will often want to form the mathematical proposition “ P and Q ”, denoted $P \wedge Q$. This denotes the mathematical proposition that is true precisely whenever both of P and Q are true, and is false otherwise.
2. Given two propositions P and Q , we can form the mathematical proposition “ P or Q ”, denoted $P \vee Q$. This denotes the mathematical proposition that is false if and only if both P and Q are false, and is true otherwise.¹²
3. Given a proposition P , we can formulate the mathematical proposition “not- P ,” which we denote $\neg P$. This is the mathematical proposition that is false whenever P is true, and true whenever P is false.
4. Given two propositions P and Q , we can form the mathematical proposition “ P is equivalent to Q ”, denoted $P \Leftrightarrow Q$. This denotes the mathematical proposition that is true when P and Q are equal (i.e. both true or both false), and false when P and Q are different (i.e. exactly one is true and the other is false.)
5. Given two propositions P and Q , we can form the mathematical proposition “ P implies Q ”, denoted $P \Rightarrow Q$. This proposition is equivalent to the claim that “if P is true, then Q must be true as well.” In particular, we say that $P \Rightarrow Q$ is false whenever P is true while Q is false (as this would break the claim “if P is true, then Q must be true as well,”) and is true otherwise.

In particular, notice that if P is false, $P \Rightarrow Q$ will evaluate to true no matter what Q is. This allows us to say that



¹²In mathematics, we almost always assume that our “or” is an inclusive-or: i.e. it is true when either P or Q is true, or even when **both** P and Q are true.

propositions like “If I am a purple elephant, then six is an odd number” are true¹³. This is because if the P part is false, it doesn’t matter whether the Q part is complete nonsense or not; our implication is automatically true! This is probably one of the harder things to get a grasp on, so take some time to absorb this.

We call propositions made by combining together other propositions in these ways **compound propositions**. Using these operations, we can create quite complicated logical statements — things like $\neg((\neg P) \wedge (\neg Q))$ or $\neg(P \rightarrow (\neg Q))$.

Sometimes we will want to study these objects formally on their own, i.e. without having to worry about what the propositions P, Q actually mean. To do this, we will often just define some **propositional variables**¹⁴ p, q, r as arbitrary elements of the set $\{T, F\}$, and make **truth tables**:

Definition. Given a compound proposition P in the propositional variables p_1, \dots, p_k , a **truth table** for P is a listing of all of the possible ways to assign the variables p_1, \dots, p_k to values in $\{T, F\}$, along with a corresponding value for P for each such assignment.

One of the main uses of a truth table is to determine when two compound propositions are logically equivalent:

Definition. We say that two compound propositions P, Q are **logically equivalent** if P and Q ’s truth values are always the same, no matter what truth values we assign to their propositional variables.

Equivalently, P and Q are equivalent if when we construct a truth table for P and Q , their columns are equal.

To illustrate this idea, here are a few examples:

Question. Construct a truth table for the propositions $A : \neg(p \wedge q)$ and $B : ((\neg p) \wedge (\neg q))$. Are A and B equivalent?

Solution: Here’s a truth table!

p	q	$p \wedge q$	$\neg(p \wedge q)$	$\neg p$	$\neg q$	$(\neg p) \wedge (\neg q)$
T	T	T	F	F	F	F
T	F	F	T	F	T	F
F	T	F	T	T	F	F
F	F	F	T	T	T	T

(Notice how we put in parts of each compound proposition into our truth table; this can help you correctly put in the right truth values for the entire compound proposition.)

Because the highlighted columns have different truth values, their corresponding propositions are **not** logically equivalent.

Question. Construct truth tables for the two statements $p \wedge (q \vee r)$ and $(p \wedge q) \vee (p \wedge r)$. Are they logically equivalent?

¹³Provided we are not purple elephants.

¹⁴When we are describing a compound proposition made out of propositional variables instead of proper mathematical statements, we’ll sometimes replace the symbols \Rightarrow and \Leftrightarrow with \rightarrow and \leftrightarrow . The only reason we do this is to help the reader distinguish between compound statements made out of actual mathematical claims, versus compound statements made out of variables from $\{T, F\}$.

Solution: We draw another truth table:

p	q	r	$q \vee r$	$p \wedge (q \vee r)$	$p \wedge q$	$p \wedge r$	$(p \wedge q) \vee (p \wedge r)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Because the highlighted columns have the same truth values, their corresponding propositions are logically equivalent!

By using these techniques, you can prove that the following statements are logically equivalent. Give a few of them a try for practice!

$\neg\neg p$	\Leftrightarrow	p	Double negation
$p \wedge q$	\Leftrightarrow	$q \wedge p$	Commutative laws
$p \vee q$	\Leftrightarrow	$q \vee p$	
$p \wedge (q \wedge r)$	\Leftrightarrow	$(p \wedge q) \wedge r$	Associative laws
$p \vee (q \vee r)$	\Leftrightarrow	$(p \vee q) \vee r$	
$p \wedge (q \vee r)$	\Leftrightarrow	$(p \wedge q) \vee (p \wedge r)$	Distributive laws
$p \vee (q \wedge r)$	\Leftrightarrow	$(p \vee q) \wedge (p \vee r)$	
$p \wedge p$	\Leftrightarrow	p	Idempotent laws
$p \vee p$	\Leftrightarrow	p	
$\neg(p \wedge q)$	\Leftrightarrow	$\neg p \vee \neg q$	De Morgan's laws
$\neg(p \vee q)$	\Leftrightarrow	$\neg p \wedge \neg q$	
$p \rightarrow q$	\Leftrightarrow	$\neg p \vee q$	Implication laws
$p \rightarrow q$	\Leftrightarrow	$\neg(p \wedge \neg q)$	

1.4 Proof Techniques

Earlier in these notes, we talked about what it means for a mathematical statement to be a **proof**; since then, we've looked at some promising examples and nonexamples of proofs, and built up some language for how to approach the idea of a proof. In this section, we're going to continue this process and introduce several useful mathematical *techniques* for how to approach a proof!

We start with the most 'straightforward' proof method:

1.4.1 Direct Proofs

To prove that a given claim is true, the most straightforward path is typically the following:

- Write down things that you know are true that relate to your claim. This typically includes the definitions of any terms referred to in the definition, any results from class or the tutorials/assignments that look related, and maybe some fundamental facts you know entering this class about numbers.
- Combine those things by using logic or algebra to create more things you know are true.
- Keep doing this until you get to the claim!

Both of the two “fixed” proofs above were direct proofs in this sense: in both cases,

- we started with definitions and known facts (that any odd number can be written in the form $2k + 1$ and that any real number when squared is nonnegative),
- combined these observations by using algebra, and then
- used more definitions and known facts to conclude the desired results!

A particularly common form of direct proof comes up when people want to prove a statement of the form “if A holds, then B must follow” for two propositions A and B (or equivalently, “ A implies B ,” which we write in symbols as $A \Rightarrow B$.)

To write a direct proof of such a statement, we proceed as before, but *also* throw in the assumption that A holds! That is, to prove “ A implies B ,” we assume that A is true, and try to combine this assumption with other known true things to deduce that B is true. (Logically speaking, this is because $A \Rightarrow B$ holds as long as we’re never in the situation where A is true and B is false. Therefore, if we can show that A being true forces B to also be true, then we know that our claim must hold!)

We illustrate this with an example here:

Claim. If n is an odd integer, then n^2 can be written as a multiple of 4 plus one.

Proof. We start by “assuming” the part by the “if:” that is, we assume that n is an odd integer. By definition, this means that we can write $n = 2k + 1$ for some other integer k .

We seek to study n^2 . By our observation above, this is just $(2k + 1)^2 = 4k^2 + 4k + 1 = 4(k^2 + k) + 1$. This is a multiple of 4 plus 1, as claimed! Therefore we have completed our proof. \square

1.4.2 Proofs by Contrapositive

Not all proofs are this straightforward, however! Sometimes we will need to be a bit more clever. Even statements of the form $P \Rightarrow Q$ can be rather tricky to prove: sometimes P is a really tricky condition to start from, and we’ll have no idea how to use our “assumption” that P is true to deduce Q .

One way to get around this sort of issue is via something called the **contrapositive**! Specifically, if we have a statement of the form $P \Rightarrow Q$, the contrapositive of this statement is simply the statement

$$\neg Q \Rightarrow \neg P.$$

The nice thing about the contrapositive of any statement is that it’s **exactly the same** as the original statement! For example, if our statement was “all UoA students are not soluble in water,” the contrapositive of our claim would be the statement “anything that is soluble in water is not a UoA student.” These two statements clearly express the same meaning – one just starts out by talking about UoA students, while the other starts out by talking about things that you can dissolve in water. So, if we want to prove a statement $P \Rightarrow Q$, we can always just prove the contrapositive $\neg Q \Rightarrow \neg P$ instead, because they’re the same thing! This can allow us to switch from relatively difficult starting points (situations where P is hard to work with) to easier ones (situations where $\neg Q$ is easy to work with.) In particular, in the example above, working with the statement “all UoA students are not soluble in

water” will make a brute-force proof much easier: it is far easier to drop every UoA student in a lake than to check every soluble object to see if it’s been to UoA recently.

To illustrate this, consider the following example:

Theorem. Let n be a natural number. Then, if $n \equiv 2 \pmod{3}$, n is not a square: in other words, we cannot find any integer k such that $k^2 = n$. (We write that $a \equiv b \pmod{c}$ iff $a - b$ is a multiple of c : in other words, that a and b are the “same” up to some number of copies of c .)

Proof. A direct approach to this problem looks . . . hard. Basically, if we were to prove this problem directly, we would take any $n \equiv 2 \pmod{3}$ – i.e. any n of the form $3m + 2$, for some integer m – and try to show that this can never be a square. Basically, we’d be looking at the equation $k^2 = 3m + 2$ and trying to show that there are no solutions to this equation, which just looks kind of. . . ugly, right?

So: because we are mathematicians, we are **lazy**. In particular, when presented with a tricky-looking problem, our instincts should be to try to make it trivial: in other words, to attempt different proof methods and ideas until one seems to “fit” our question. In this case, as suggested by our section title, let’s attempt to prove our theorem by studying its contrapositive:

If n is a square, then $n \not\equiv 2 \pmod{3}$.

Equivalently, because every number is equivalent to either 0, 1, or $2 \pmod{3}$, we’re trying to prove the following:

If n is a square, then $n \equiv 0$ or $1 \pmod{3}$.

This is now a much easier claim! – the initial condition is really easy to work with, and the later condition is rather easy to check.

Now that we have some confidence in our ability to prove our theorem, we proceed with the actual work: take any square n , and express it as k^2 , for some natural number k . We can break k into three cases:

1. $k \equiv 0 \pmod{3}$. In this case, we have that $k \equiv 3m$ for some m , which means that $k^2 = 9m^2 = 3(3m^2)$ is also a multiple of 3. Thus, $k^2 \equiv 0 \pmod{3}$.
2. $k \equiv 1 \pmod{3}$. In this case, we have that $k \equiv 3m + 1$ for some m , which means that $k^2 = 9m^2 + 6m + 1 = 3(3m^2 + 2m) + 1$. Thus, $k^2 \equiv 1 \pmod{3}$.
3. $k \equiv 2 \pmod{3}$. In this case, we have that $k \equiv 3m + 2$ for some m , which means that $k^2 = 9m^2 + 12m + 4 = 3(3m^2 + 4m + 1) + 1$. Thus, $k^2 \equiv 1 \pmod{3}$.

Therefore, we’ve shown that k^2 isn’t congruent to $2 \pmod{3}$, for any k . So we’ve proven our claim! \square

We give a second example here:

Theorem. For an integer x , if $x^2 + 6x + 91$ is odd then x is even.

Proof. This is another problem where direct proof seems hard. If we’re starting from the assumption that $x^2 + 6x + 91$ is odd, we could apply the definition of “odd” to write

$$x^2 + 6x + 91 = 2k + 1$$

for some integer k ; but what would we do from here to get information about x ? Absent any particularly clever ideas we’d have to just start solving for x , which will involve square roots and some pretty ugly calculations; not likely a great idea.

Instead, let’s try the contrapositive again! If we take our statement above and form the contrapositive, we get the following easier version of our claim:

Theorem. For an integer x , if x is odd then $x^2 + 6x + 91$ is even. This seems much easier! In particular, we have a much nicer starting place: we start with an odd integer x , which means we can write $x = 2k + 1$ for some other integer k .

From here, it's also relatively clear what we should do: let's plug this identity into the right-hand part! In particular, if $x = 2k + 1$, then $x^2 + 6x + 91 = (2k + 1)^2 + 6(2k + 1) + 91 = 4k^2 + 4k + 1 + 12k + 6 + 91 = 4k^2 + 16k + 98 = 2(2k^2 + 8k + 49)$. So this is even, as claimed! Therefore we've proven the contrapositive to our original claim, and with it proven our original claim itself. \square

1.4.3 Proofs by Contradiction

Contradiction is another proof method that can be remarkably useful when we're stuck on something difficult. The best way to understand how a proof by contradiction works is to start with an example:

Theorem. The number $\sqrt{2}$ is not rational.

Proof. As always, let's start by unpacking our definitions:

- $\sqrt{2}$ is the unique positive real number such that when we square it, we get 2.
- A number x is **rational** if we can write $x = \frac{m}{n}$, where m and n are integers and n is nonzero.

With this done, our claim can be unpacked to the following:

“For a real number x , if $x = \sqrt{2}$, then there are no values of $m, n \in \mathbb{Z}$ with $n \neq 0$ such that $x = \frac{m}{n}$.”

So: how do we do this? Because the problem wants us to show that we cannot write $\sqrt{2} = \frac{m}{n}$ for any integers m, n with $n \neq 0$, we can't just check a few examples: we'd have to look at *all* of them, and this could be quite difficult! We'd have to find some useful property that makes all examples of this form fail, and this could be quite hard to find.

Instead, consider the following way to “side-step” these difficulties. Instead of looking at all pairs m, n and trying to show that each one fails, let's assume that we *have* one such pair m, n such that $\sqrt{2} = \frac{m}{n}$!

With this assumption in hand, let's now show that this assumption “breaks mathematics” in some way: that starting from this assumption, we can get to something we know is impossible, like $1 + 1 = 0$.

If we can do this, then we know that our original assumption that there was such a fraction $\frac{m}{n}$ must have been nonsense (i.e. false), and therefore that our claim that no such fraction exists is true!

More generally, this is how proof by contradictions go:

- We have a claim we're trying to prove; let's denote it P , for shorthand.
- Instead of proving P is true directly, we want to prove that $\neg P$ is impossible.
- To do this, we can simply do the following:
 1. Assume, for the moment, that $\neg P$ is actually true!
 2. Working from this assumption, find a pair of contradictory statements that are implied by $\neg P$; i.e. a pair of statements $Q, \neg Q$ such that $(\neg P) \Rightarrow Q$ and $(\neg P) \Rightarrow \neg Q$. Common examples are $Q = “1 = 0”$, or $Q(n) = “n \text{ is even}”$ or other such things.

3. This proof demonstrates that $\neg P$ must be impossible, because it implies two contradictory things (like the two simultaneous claims $Q(n) = “n \text{ is even}”$ and $\neg Q(n) = “n \text{ is odd}.”$) Mathematics is free from contradictions by design¹⁵; therefore, we know that this must be impossible, i.e. that $\neg P$ must be false, i.e. that P must be true!

In general, this is how a **proof by contradiction** works;¹⁶ take your claim P , assume it's false, and use $\neg P$ to deduce contradictory statements, which you know mathematics cannot contain.

We do this here! Suppose that we can find two integers m, n with $n \neq 0$ such that $\sqrt{2} = \frac{m}{n}$. If m and n have common factors, divide through by those factors to write $\frac{m}{n}$ in its simplest possible form: that is, don't write something like $\frac{3}{6}$ or $\frac{12}{24}$, write $\frac{1}{2}$

Then if we square both sides, we get $2 = \frac{m^2}{n^2}$. Multiplying both sides by n^2 gives us $2n^2 = m^2$, which means that m^2 is even (because it is a multiple of 2)!

This means that m is even (see problem 5 on week 2's tutorial!), and therefore that we can write $m = 2k$ for some integer k . If we plug this into our equation $2n^2 = m^2$, we get $2n^2 = (2k)^2 = 4k^2$, and by dividing by 2 we have $n^2 = 2k^2$.

This means that n^2 is even, and therefore that n is even as well (same logic as before!)

But this means that both n and m are multiples of 2; that is, that they have a common factor! We said earlier that we'd divided through by any common factors to get rid of them, so this is a **contradiction**: from our initial assumption we got to something that is both true and false. As a result, our original assumption (that we could write $\sqrt{2} = \frac{m}{n}$) must be false; that is, we have shown that $\sqrt{2} \neq \frac{m}{n}$ for any integers m, n with $n \neq 0$, as desired. Yay! \square

We consider another example here:

Theorem. There are two irrational numbers a and b such that a^b is rational.

Proof. In the example we're studying here, we want to show that it's impossible for a^b to be irrational for every pair of irrational numbers a, b . To do this via a proof by contradiction, we do the following: first, assume that a^b **is** irrational for every pair of irrational numbers a, b ! If we apply this knowledge to one of the few numbers ($\sqrt{2}$) we know is irrational, our assumption tells us that in specific

$$\sqrt{2}^{\sqrt{2}} \text{ is irrational.}$$

What do we do from here? Well: pretty much the only thing we have is our assumption, our knowledge that $\sqrt{2}$ is irrational, and our new belief that $\sqrt{2}^{\sqrt{2}}$ is **also** irrational. The only thing really left to do, then, is to let $a = \sqrt{2}^{\sqrt{2}}$, $b = \sqrt{2}$, and apply our hypothesis again. But this is excellent! On one hand, our we have that a^b is irrational by our hypothesis. On the other hand, we have that a^b is equal to

$$\left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \sqrt{2}^{\sqrt{2} \cdot \sqrt{2}} = \sqrt{2}^2 = 2,$$

¹⁵This is why we tried to insure that we only start with true statements in our proofs.

¹⁶A beautiful quote about proofs by contradiction, by the mathematician G. H. Hardy: “[Proof by contradiction], which Euclid loved so much, is one of a mathematician's finest weapons. It is a far finer gambit than any chess gambit: a chess player may offer the sacrifice of a pawn or even a piece, but a mathematician offers the game.”

which is clearly rational. This is a contradiction! Therefore, we know that our hypothesis must be false: there must be a pair of irrational numbers a, b such that a^b is rational. \square

An interesting quirk of the above proof is that it didn't actually give us a pair of irrational numbers a, b such that a^b is rational! It simply told us that either

- $\sqrt{2}^{\sqrt{2}}$ is rational, in which case $a = b = \sqrt{2}$ is an example, or
- $\sqrt{2}^{\sqrt{2}}$ is irrational, in which case $a = \sqrt{2}^{\sqrt{2}}, b = \sqrt{2}$ is an example,

but it never actually tells us which pair satisfies our claim! This is a weird property of proofs by contradiction: they are often **non-constructive** proofs, in that they will tell you that a statement is true or false without necessarily giving you an example that demonstrates the truth of that statement.

1.4.4 Proofs of Equivalence

Another common proof technique comes up when we're trying to prove two statements are equivalent. For example, suppose that we have the following two statements:

- $P(x, y) = "(x + 1)^2 = (y + 1)^2."$
- $Q(x, y) = "x + y = -2," \text{ or } x = y."$

As it turns out, these two statements are equivalent: i.e. $P(x, y) \Leftrightarrow Q(x, y)$. How can we prove this? Well, one useful blueprint for such a proof is the following:

- First, show that $P(x, y) \Rightarrow Q(x, y)$: i.e. that if we assume $P(x, y)$ is true, then we can conclude that $Q(x, y)$ is also true.
- Then, show the opposite direction: that $Q(x, y) \Rightarrow P(x, y)$! I.e. we will assume that $P(x, y)$ is true, and attempt to prove that $Q(x, y)$ is also true.

If we have done this, we will have proven that $P(x, y)$ is true if and only if $Q(x, y)$ is also true: i.e. that $P(x, y) \Leftrightarrow Q(x, y)$! Excellent. Now, let's actually do this for these two statements, to illustrate how such a proof works:

$P(x, y) \Rightarrow Q(x, y)$: Assume that $P(x, y)$ holds: i.e. that $(x+1)^2 = (y+1)^2$. Then, by taking square roots of both sides, we have that

$$(x + 1) = \pm(y + 1),$$

where the \pm is because there are two possible square roots for any positive number, either its positive square root or that same positive square root times (-1) . So: if

$$(x + 1) = +(y + 1),$$

then subtracting 1 from both sides gives us $x = y$, which is one possible way to make $Q(x, y)$ true. Otherwise, if

$$(x + 1) = -(y + 1),$$

we can add y to both sides and subtract 1 from both sides to get $x + y = -2$, which is another way to make $Q(x, y)$ true. Therefore, in either case, if $P(x, y)$ is true, so is $Q(x, y)$!

$Q(x, y) \Rightarrow P(x, y)$: There are two different ways to make $Q(x, y)$ true: either set $x = y$ or set $x + y = -2$, i.e. $x + 1 = -(y + 1)$. In either case, we have that $(x + 1)^2 = (y + 1)^2$, so we know that $P(x, y)$ is true.

Therefore, we've shown that $P(x, y) \Leftrightarrow Q(x, y)$.

1.4.5 Proof by Construction

In all of the proofs above, we've been focused on proving claims about "all" numbers x, y , or "all" odd integers n , or other sorts of "universal" claims about things. When we're proving claims of these forms, then we need to use techniques and arguments like the ones above where we work in general / don't get to use examples to prove our claim!

Sometimes, however, we'll find ourselves with claims of the form "There exists a number n such that..." or "There is a value x with the property..." In this sort of situation, we're not being asked to show that something is true for **all** values: instead, we're just asked to find a single example!

In situations like this, a common technique is **proof by construction**, where we simply create an object with the desired properties. We illustrate this with an example:

Claim. There is an odd integer that is a power of two.

Proof. Notice that $2^0 = 1$. Therefore, 1 is a power of 2. 1 is also odd, as we can write it in the form $1 + 2k$ for some integer k (specifically, $k = 0$.) Therefore we've constructed the claimed integer, as desired. \square

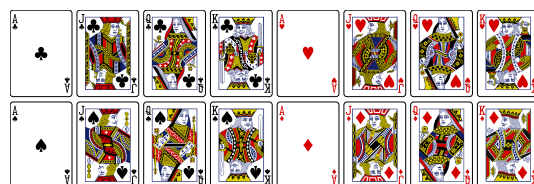
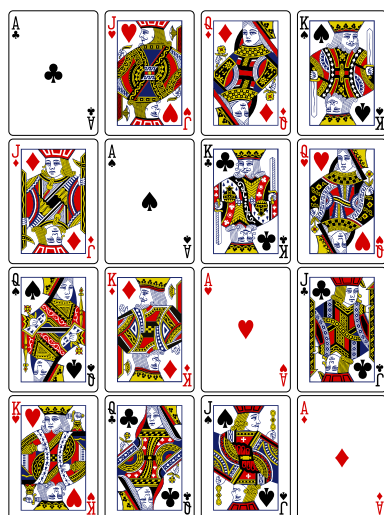
Notice the following two aspects of this proof:

- We didn't have to work with a general integer n ; instead, we got to give a specific example! This is because our claim was of the form "There is...", which means that we're just asked for a single example. If our proof had started "For all...", this would be different, and this proof would be invalid (just like how examples weren't enough for a proof in our earlier "the sum of any two odd numbers is even" claim.)
- Also notice that we didn't just say "1 is the answer" and ended our proof; we actually took the time to explain **why** 1 has the desired properties. You should expect to always do this!

We give a second example, to illustrate how these sorts of things come up in combinatorics and/or "puzzle" mathematics:

Claim. Take the aces and face cards from a standard 52-card deck. Can you arrange them in a 4×4 grid so that no suits or symbols are repeated in any row or column?

Proof. Behold!



In this proof, we don't have much to really explain: the solution presented self-evidently has the desired property (just check every row and column.) If it was unclear, though, we'd have to have some explanation along with our answer!

1.4.6 Disproofs

We've talked a lot about how to prove claims in the above subsections. It's also worth talking here about how to **disprove** claims: that is, how to show that something is false!

There are, broadly speaking, two sorts of claims we'll run into in mathematics:

- Claims of the form "There is some x such that ...", or more generally claims of **existence**. To prove a claim about existence, as noted above in the constructive proof section, we just need to construct an example that shows that such a thing exists!

To **disprove** such a claim, however, we'd need to show that such an example **cannot exist**. That is, we're making a claim about **all** things, and saying that **none** of them are examples for our claim! In other words, the opposite of "There is an x such that ..." is "For every x , it is false that ..."

To give a natural language example, suppose that someone made the claim that "There is a University of Auckland student that is 3 meters tall." To disprove this, we would need to look at **all** UoA students, and show that each one of them is not three meters tall! In other words, our claim about existence (*there is* a UoA student) became a claim about all students (*every* UoA student), and the property we wanted (is three metres tall) was made false.

- Conversely, we might want to study claims of the form "For all x , we have that ...", which we can think of as "**universal**" claims. These usually have words like "every," "any" or "all" in them. To prove a claim about all things, as noted before, we need to work in general and construct arguments by using definitions/etc to cover all possible cases.

To **disprove** such a claim, however, we can get away with a lot less work! To "break" a claim about all values of x , it suffices to find just one value of x such that our claim fails. In other words, the opposite of "For all x , we have that ..." is "There is an x such that ... fails." We call such a value x a **counterexample**, and often prove that such things exist via construction (as discussed earlier.)

To give a second natural language example, consider the claim "Every University of Auckland student does not have red hair." To prove this claim false, we just need to find **one** UoA student who **has** red hair! In other words, our universal claim (*every* UoA student) became an existence claim (*there is* a UoA student), and the condition we wanted (does not have red hair) was made false.

We give an example of each sort of disproof here:

Claim. For any integer n , at least one of the two numbers $n^2 + 1, n^2 + 2$ must be prime.

Disproof. We start by writing out what the opposite of this claim is, so that we know what a disproof would be. As noted above, we

disprove claims about all things by finding a single **counterexample**: so the negation of our claim should start with the phrase “There is an integer n such that . . .”

We follow this with the negation of our desired property. If our original claim was that at least one of $n^2 + 1, n^2 + 2$ is prime, then the negation of this claim is that **neither** of them are prime. That is, the negation of our original claim, in its entirety, is the following:

“There is an integer n such that both $n^2 + 1$ and $n^2 + 2$ are nonprime.”

If we’re disproving our original claim, then we’re just trying to prove this new, negated claim!

This is a “there is” proof, and so we can prove it by construction: that is, we just need to find a value of n that makes both $n^2 + 1$ and $n^2 + 2$ nonprime. To find such a value, we just try numbers until we find one that has the desired properties:

- If $n = 1$, then $n^2 + 1 = 2$ and $n^2 + 2 = 3$. At least one of these is a prime, so this isn’t a counterexample.
- If $n = 2$, then $n^2 + 1 = 5$ and $n^2 + 2 = 6$. At least one of these is a prime, so this isn’t a counterexample.
- If $n = 3$, then $n^2 + 1 = 10$ and $n^2 + 2 = 11$. At least one of these is a prime, so this isn’t a counterexample.
- If $n = 4$, then $n^2 + 1 = 17$ and $n^2 + 2 = 18$. At least one of these is a prime, so this isn’t a counterexample.
- If $n = 5$, then $n^2 + 1 = 26$ and $n^2 + 2 = 27$. Both of these are nonprimes (as $26 = 2 \cdot 13$ and $27 = 3 \cdot 3 \cdot 3$), so **this is a counterexample!**

□

Claim. There is an integer n such that $n(n + 1) + 3 = 0$.

Disproof. As before, we start by writing out what the opposite of this claim is. This is a claim about existence, so we can disprove it by showing that no examples exist: that is, we’ll write a “universal” proof that looks at all things, and shows that they are all not examples! In other words, the opposite of our claim is the following:

“For every integer n , we have that $n(n + 1) + 3 \neq 0$.”

To prove this new, negated version of our original claim, we proceed with a direct proof. Take any integer n (and notice that we have to consider **any** possible integer n , i.e. we can’t just find a single example, as we need to work in general.) Now, notice the following:

- For any integer n , n is either even or odd; that is, we can either write $n = 2k$ or $n = 2k + 1$ for some integer k .
- Therefore, we either have that $n(n + 1) + 3 = (2k)(2k + 1) + 3 = 2(k(k + 1) + 1) + 1$, or $n(n + 1) + 3 = (2k + 1)(2k + 2) + 3 = 2((2k + 1)(k + 1) + 1) + 1$. In both cases, we have that this number is odd, as we’ve written it as a multiple of 2 plus 1.
- 0 is even (as we can write $0 = 2 \cdot 0$), and thus is not odd.
- Therefore $n(n + 1) + 3 \neq 0$ for any integer n , as an odd number can never be equal to an even number!

□

1.5 Quantifiers

We close our first chapter of Compsci 225 by introducing the definition for a concept we used quite heavily in our previous sections: the idea of **quantifiers**.

Definition. In mathematics, we define the **universal quantifier**, \forall , as shorthand for the phrase “For every” (or equivalently “for all,” “for every.”)

Similarly, we define the **existential quantifier** \exists as shorthand for the phrase “There exists” (or equivalently “there is,” “you can find a . . .”)

As practice, we can rewrite the claims we studied in the previous section with this quantifier notation as follows:

- The claim “For any integer n , at least one of the two numbers $n^2 + 1, n^2 + 2$ must be prime” can be written with quantifiers as follows:

$$\forall n \in \mathbb{Z}, \text{ either } n^2 + 1 \text{ or } n^2 + 2 \text{ must be prime.}$$

- The claim “There is an integer n such that $n(n+1)+3=0$ ” can be written with quantifiers as follows:

$$\exists n \in \mathbb{Z} \text{ such that } n(n+1)+3=0.$$

We can often have mathematical statements that have “nested” quantifiers in them, i.e. multiple quantifiers in a row. For example, the statement “For every real number x , if $x \geq 0$, then there is some real number y such that $x = y^2$ ” can be written with quantifiers as follows:

$$\forall x \in \mathbb{R}, (x \geq 0) \Rightarrow (\exists y \in \mathbb{R} \text{ such that } x = y^2).$$

One thing to notice with quantifiers is that we always specify what set we’re choosing our variables from: that is, we always write things like $\forall x \in \mathbb{R}$ or $\exists y \in \mathbb{Z}$. We call the set that we’re choosing these variables from the **universe** for that quantifier: i.e. the universe in the expression “ $\forall x \in \mathbb{N}$ ” is just \mathbb{N} .

On their own, quantifiers are pretty unassuming; they’re just shorthand for mathematical phrases that we know and use on a daily basis! The reason that I’ve put them into a separate section here is because they have a few surprising properties that can catch students by surprise:

1.5.1 Order and Quantifiers

The first thing to stress is that with quantifiers, the **order** in which you write them is usually very important! To illustrate this with an example, consider the following two claims:

A: “There is a song that everyone in Auckland can sing.”

B: “Everyone in Auckland has some song that they can sing.”

If we were to write this with quantifiers and let A be the set of all people in Auckland and S be the set of all songs, these sentences would become

A: $\exists s \in S$ such that $\forall a \in A, a$ can sing s .

B: $\forall s \in S, \exists a \in A$ such that a can sing s .

On one hand, A and B are the “same” sentence, just with the order of the quantifiers swapped. They’re both claims about some song existing and about everyone in Auckland’s ability to sing!

However, the **meaning** of these two sentences is very different! In A, we're saying that there is **some** song such that **everyone** knows how to sing that specific song. If this were true, then this would be something like saying "everyone in Auckland knows how to sing Bohemian Rhapsody," or something else like that.

In B, we're making a much weaker claim: we're just saying that each person in Auckland has a song that they personally know how to sing. In particular, because the "for all" quantifier comes first, each person gets to have *their own song* that they know how to sing: so some people might know the NZ national anthem, others might know Bohemian Rhapsody, others might know the Pokémon theme song, and so on/so forth.

As a result, we can see that switching the order of quantifiers can dramatically change the meaning of a claim! There are certainly instances where you can change the order and you'll still get something that's logically equivalent, but this is not common.

In general, when you're reading a sentence with quantifiers in it, try to read left-to-right and imagine each part of the quantifier becoming fixed after you've parsed it. So, in A, once we've read the " $\exists s \in S$ " part, we've forced ourselves to pick out some song s , and now need the rest of the sentence to make sense given that chosen s . In B, however, we first read " $\forall a \in A$," and so the first thing we fix is the person a : from here we just need to find a song s for that specific person, which is much easier than finding one song that works for everyone!

Here's a second, more mathematical example. Consider the following two statements, that both depend on a set S of real numbers:

A: For every $x \in S$, there is a $y \in S$ such that $xy = 1$.

B: There is a $y \in S$ such that for every $x \in S$, $xy = 1$.

In quantifiers, we can write these sentences as follows:

A: $\forall x \in S, \exists y \in S$ such that $xy = 1$.

B: $\exists y \in S$ such that $\forall x \in S, xy = 1$.

Again, these sentences are the "same" except the order of the quantifiers has been reversed. Just like before, though, this dramatically changes the meaning of each sentence!

A, for instance, is the claim that for any $x \in S$, we can find some value to multiply x by to get back to 1. In other words, A is the claim that every number in S has a *multiplicative inverse* also in S .

So, for example, a set like $S = \{3, 2, \frac{1}{2}, \frac{1}{3}\}$ would satisfy¹⁷ A, because for every element in S we can find another element in S such that their product is 1: if $x = 2$ we pick $y = \frac{1}{2}$, if $x = 3$ we pick $y = \frac{1}{3}$, if $x = \frac{1}{2}$ we pick $y = 2$, and if $x = \frac{1}{3}$ we pick $y = 3$.

B, however, is the claim that there is one element in S that when multiplied by *anything else* will always yield 1! In other words, a set like $\{3, 2, \frac{1}{2}, \frac{1}{3}\}$ would not satisfy B, because no matter what element you pick for y from $\{3, 2, \frac{1}{2}, \frac{1}{3}\}$, it will not be true that y times *every other element in B* will always yield 1.

In particular, if we choose $x = y$ $y \cdot x \neq 1$ for any y in the set $\{3, 2, \frac{1}{2}, \frac{1}{3}\}$, which is enough to show that this claim does not hold for *all* values of x .

Therefore these two statements are inequivalent: that is, their truth values can disagree with each other!

¹⁷We say that an object A **satisfies** a proposition that depends on some variable A if plugging in that specific object into the proposition makes it true. For instance, $x = 1$ satisfies the proposition " $x^2 = 1$ ", while $x = 2$ does not satisfy the proposition "For all $y \in \mathbb{R}, xy = y$."

1.5.2 Unexpected Quantifiers

A second note to be aware of is that some mathematical phrases have quantifiers “implicitly” baked into them, or might not have the quantifiers you expect:

- If someone says something like “If n is an integer, then $n(n+1)$ is always even,” then they’re making a claim about **all** integers. That is: you wouldn’t prove this claim by just looking at $n = 3$ and saying that it works, just like you wouldn’t prove a claim of the form “if you always guess (d) you’ll ace the mid-sem test” by saying “Hey, it worked in 2012 therefore it’ll always be true.”

Therefore, if you wanted to translate this sentence into quantifiers, you’d say “ $\forall n \in \mathbb{Z}, n(n+1)$ is always even.”

- Similarly, in many theorem / claim / exercise statements in mathematics, you’ll see phrases like
 - “Let x be a real number. Prove that ...”
 - “Assume that A is a set of integers. Show that ...”
 - “Suppose that n is a natural number. Prove that ...”

In all of these cases, there is an implicit \forall quantifier on the front of these statements. That is, if we were to translate these statements into quantifiers, we would do so as follows:

$$\text{– } \forall x \in \mathbb{R} \dots \quad \text{– } \forall A \subseteq \mathbb{Z} \dots \quad \text{– } \forall n \in \mathbb{N} \dots$$

This is because phrases like “let”, “suppose” and “assume” are telling you to consider an arbitrary object, and to then show that it has the desired property. That is: imagine if someone started a conversation with you by saying “Suppose you got to have a dog. What kind of breed would you pick?” The first part of this sentence invites you to imagine **every** kind of dog; as a result, a universal quantifier is the right choice for describing this sort of claim.

- Finally, claims of the form “There is no x such that ...” sometimes trip people up because of the way in which the English translates to quantifiers and negation.

That is: suppose you had the claim “There is no x such that $x^2 < 0$.” In quantifiers, I claim you would translate this as $\neg(\exists x \text{ such that } x^2 < 0)$, i.e. “It is not true that there exists a value of x such that $x^2 < 0$.”

This makes sense if you parse it through: saying that “no thing exists” is equivalent to saying that “it is false that the thing exists,” which is how we translated the statement. As we discussed before, we can simplify this to the statement “ $\forall x, x^2 \geq 0$,” this is because saying that “no thing exists with property blah” is saying “for all things, they do not have property blah.”

When you’re first starting out with proofs, however, it is tempting to translate “There is no x such that $x^2 < 0$ ” as “ $\exists \neg x$ such that $x^2 < 0$.” This, however, doesn’t make any sense! That is: what should $\neg x$ mean? If x is a real number like π , what would $\neg \pi$ even be?

In an attempt to fix this, sometimes people just decide “eh, the negation should have probably been to the right of x ” and write “ $\exists x$ such that $\neg(x^2 < 0)$.” This at least makes sense, but on the other hand is *not at all* what you want to say!

In this case, this mistaken translation is the claim that “there is a value of x such that $x^2 \geq 0$.” In the original, we were making a very strong statement about **all** real numbers and that their squares can never be negative; in the mistake, however, we have the much weaker claim that there is just some real number with nonnegative square (which is much less useful!)

1.5.3 Negation and Quantifiers

One last thing to specifically notice about quantifiers is how their **negations** work. In particular, notice that in our earlier chapter we saw that the negation of

“There is an integer n such that $n(n + 1) + 3 = 0$ ”

was

“For every integer n , we have that $n(n + 1) + 3 \neq 0$,”

and similarly that the negation of

“For any integer n , at least one of the two numbers
 $n^2 + 1, n^2 + 2$ must be prime”

was

“There is an integer n such that both $n^2 + 1$ and $n^2 + 2$ are nonprime.”

In each of these cases, negating our claims did two things:

- We “flipped” our quantifiers: that is, we switched the \forall and \exists quantifiers. This is because we disprove a “for all” claim by showing that a counterexample *exists*, and we disprove a “there exists” claim by showing that *all* things are not examples for that claim.
- We then negated the property that comes afterwards, which is how we get the “counterexample” / “not example” part.

In general, the rule for negating quantifiers is as follows:

- $\neg(\exists x \in A \text{ such that } \dots)$ is equivalent to $\forall x \in A, \neg(\dots)$
- $\neg(\forall x \in A \text{ such that } \dots)$ is equivalent to $\exists x \in A, \neg(\dots)$

In particular, notice that when we negate a quantifier, the “universe” of the quantifier (i.e. the set that the variable that we’re quantifying comes from) remains the same: i.e. we don’t flip from “ $\forall x \in \mathbb{R}$ ” to “ $\exists x \notin \mathbb{R}$ ”, we flip to “ $\exists x \in \mathbb{R}$.” This is because we’d disprove a claim about all real numbers by looking at other real numbers; in general, this is why we do not change the universe that we’re quantifying over.

Here’s an example, to illustrate how this works:

Problem. Without using any words of negation (i.e. without using the words not, no, nor, neither, ...), write down a sentence that describes the negation of the following:

“If a book on my bookshelf has a page with more than fifty words on it, then the first letter of every word on that page is a vowel.”

Solution: First off, we try to translate our claim into quantifiers. As noted before, this sort of “if” statement has a quantifier built into it: we’re implicitly talking about **all** of the books on my bookshelf. Therefore, this sentence could be roughly translated into quantifiers as follows:

$\forall b$ on my shelf, $\left((\exists \text{ a page } p \text{ in } b \text{ with } 50+ \text{ words}) \Rightarrow (\forall \text{ word } w \text{ on } p, w \text{ starts with a vowel.}) \right)$

By using our rules from earlier, this sentence has the negation

$\exists b$ on my shelf such that $\neg \left((\exists \text{ a page } p \text{ in } b \text{ with } 50+ \text{ words}) \Rightarrow (\forall \text{ word } w \text{ on } p, w \text{ starts with a vowel.}) \right)$

Recall that $A \Rightarrow B$ fails if and only if A is true and B is false. Therefore, if we're negating a claim of the form $A \rightarrow B$, we're saying that the A part must be true and the B part must be false! In our claim, then, the negation of the section in parentheses is just saying that $(\exists \text{ a page } p \text{ in } b \text{ with } 50+ \text{ words})$ should be true and $(\forall \text{ word } w \text{ on } p, w \text{ starts with a vowel})$ should be false: i.e.

$\exists b$ on my shelf such that $(\exists \text{ a page } p \text{ in } b \text{ with } 50+ \text{ words}) \wedge \neg(\forall \text{ word } w \text{ on } p, w \text{ starts with a vowel.})$

We again apply our negation-of-quantifiers rule to simplify this last bit, to get

$\exists b$ on my shelf such that $(\exists \text{ a page } p \text{ in } b \text{ with } 50+ \text{ words}) \wedge (\exists \text{ word } w \text{ on } p, w \text{ starts with a consonant.})$

We can translate this back into plain English as follows:

“There is a book on my bookshelf with a page with more than fifty words on it, such that there is some word on that page that starts with a consonant.”

Here's a more math-y example to consider:

Problem. Fermat's Last Theorem is the following claim:

If n is an integer that is greater than or equal to 3, then there are no positive integers x, y, z such that the equation $x^n + y^n = z^n$ has a solution.

Write the negation of this sentence, and then simplify it.

Solution: Written with quantifiers, and recalling that these sorts of “if” statements have an implicit “for all” hanging around the front, our statement is just

$$\forall n \geq 3 \in \mathbb{Z}, \neg(\exists x, y, z > 0 \in \mathbb{Z} \text{ such that } x^n + y^n = z^n)$$

We can simplify this using our negation rules to

$$\forall n \geq 3 \in \mathbb{Z}, \forall x, y, z > 0 \in \mathbb{Z}, x^n + y^n \neq z^n.$$

The negation of this is also straightforward to write with our negation rules:

$$\neg(\forall n \geq 3 \in \mathbb{Z}, \neg(\exists x, y, z > 0 \in \mathbb{Z} \text{ such that } x^n + y^n = z^n)) \\ \Rightarrow \exists n \geq 3, \exists x, y, z > 0 \in \mathbb{Z} \text{ such that } x^n + y^n = z^n.$$

Notice how throughout this process we kept the “universe” parts of these claims the same: that is, when we negated $\forall n \geq 3 \in \mathbb{Z}$, we switched this to $\exists n \geq 3 \in \mathbb{Z}$, not to $\exists n < 3 \in \mathbb{Z}$. This, again, is because our original claim was about all integers that are at least 3! Therefore, if we want to disprove such a claim, we should be arguing that we can find a counterexample amongst the integers that are at least 3; that is, we preserve the universe, and just negate the property.

Chapter 2: Integers, Algorithms, and Mods

Weeks 3-4

UoA 2018

In the past section, we spent a lot of time talking about the technique and language of mathematical proof. Here, we put these skills to the test by studying a single subject in depth: the **integers**! In this section, we will study many different concepts involving whole numbers, ranging from primes to greatest common divisors to modular arithmetic.

Alongside these concepts, we will study a new proof technique that we didn't examine in the past section: the idea of "proof by algorithm!" In this section, we will use algorithms to prove that certain mathematical theorems hold; as well, we will "reverse" this process and sometimes write mathematical proofs to show that certain algorithms are guaranteed to work.

We start with some fundamental definitions:

2.1 Factors, Primes, and Algorithmic Proofs

Definition. The set of **integers**, \mathbb{Z} , is the set¹⁸ of all whole numbers (where we consider negative numbers and 0 to be whole numbers.) In other words,

$$\mathbb{Z} = \{\dots -4, -3, -2, -1, 0, 1, 2, 3, 4 \dots\}$$

Similarly, the set of **natural numbers**, \mathbb{N} , is the set of all non-negative¹⁹ integers. In other words,

$$\mathbb{N} = \{0, 1, 2, 3, 4, \dots\}$$

Definition. Given two integers a, b , we say that a is a **factor** of b if there is some other integer k such that $ak = b$. We write $a \mid b$ as shorthand for " a is a factor of b ." Another synonym for factor is "divisor;" that is, someone could write " a is a divisor of b " or " a divides b ", and these would both mean the same thing as $a \mid b$.

Here's a string of examples, to make this clear:

- $4 \mid 12$; this is because we can multiply 4 by 3 to get 12.
- $-6 \mid 72$; this is because we can multiply 6 by -12 to get 72.
- $2 \nmid 15$; this is because for any integer k , $2k$ is an even number, and so is in particular never equal to 15.
- $1 \mid n$ for any integer n ; this is because we can always multiply 1 by n to get n . Similarly, $n \mid 1$.
- $n \mid 0$ for any integer n ; this is because we can always multiply n by 0 to get 0.

Definition. A **prime number** is any positive integer with precisely two distinct positive factors; namely, 1 and itself. The first few prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...

¹⁸If you're wondering why we use the letter "Z" here: it's because the German word for number is "zahlen," and many German mathematicians (e.g. Gauss) contributed to the foundations of modern number theory.

¹⁹Note: mathematicians often disagree about whether 0 is a natural number. Some mathematicians, notably number theorists, want \mathbb{N} to consist of only the positive integers, and so do not consider 0 to be a natural number. In theoretical computer science, though, we usually consider 0 to be a natural number; this is because we want our arrays in coding to start at 0, not 1! (Relatedly, if you're wondering about why MATLAB starts its arrays at 1 unlike all other languages, this is why.)

Note that 1 is not a prime number! This is because 1's only factor is 1, and so 1 does not have two distinct prime factors. As well, notice that 2 is the only even prime number; this is because every other positive even number by definition has the form $2k$, and so has at least 1, 2, k , $2k$ as its set of factors (and thereby has at least three distinct factors, namely 1, 2 and $2k$.)

Definition. A **composite number** is any positive integer n that can be written as the product of two integers a, b , both of which are at least 2 (and thus both of which are strictly smaller than n .) For example, $6 = 2 \cdot 3$, $9 = 3 \cdot 3$, and $24 = 2 \cdot 12$ are all composite. Notice that any positive number is either a prime, composite, or 1.

Definition. Given a positive integer n , a **prime factorization** of n is any way to write n as a product of prime numbers.

Here are a few example prime factorizations:

- $120 = 2^3 \cdot 3 \cdot 5$,
- $243 = 3^5$,
- $30031 = 59 \cdot 509$

A remarkably useful fact about prime numbers is that **every** positive integer has a prime factorization: in this sense, we can think of prime numbers as the “building blocks” of the integers. Finding such factorizations is both incredibly useful (see: [RSA cryptography](#), many other forms of encryption) and incredibly hard to do efficiently (see [here](#) for more details!)

For our first proof in this section, let's show that this fact is true! That is, let's prove the following theorem:

Theorem. If n is a positive integer, then we can find a list of prime numbers p_1, p_2, \dots, p_k such that $n = p_1 \cdot p_2 \cdot \dots \cdot p_k$.

To prove this result, we use an **algorithm**. In case you haven't seen what an algorithm is here, we define this term here:

Definition. An **algorithm** is a precise and unambiguous set of instructions.

Typically, people think of algorithms as a set of instructions for solving some problem; when they do so, they typically have some restrictions in mind for the kinds of instructions they consider to be valid. For example, consider the following algorithm for proving the [Riemann hypothesis](#):

1. Prove the Riemann hypothesis.
2. Rejoice!

On one hand, this is a fairly precise and unambiguous set of instructions: step 1 has us come up with a proof of the Riemann hypothesis, and step 2 tells us to rejoice. (I guess if you wanted to nitpick, “rejoice” isn't very carefully defined; but I don't think this is really the problem with implementing the above algorithm.)

On the other hand: this is not a terribly useful algorithm. In particular, its steps are in some sense “too big” to be of any use: they reduce the problem of proving the Riemann hypothesis to ... proving the Riemann hypothesis. Typically, we'll want to limit the steps in our algorithms to **simple**, mechanically-reproducible steps: i.e. operations that a computer could easily perform, or operations that a person could do with relatively minimal training.

In practice, the definition of “simple” depends on the context in which you are creating your algorithm. Consider the algorithm for making pancakes, given at right. This algorithm's notion of

1. Acquire and measure out the following ingredients:
 - 2 cups of buttermilk, or 1 cup milk + 1 cup yoghurt whisked together.
 - 2 cups of flour.
 - 2 tablespoons of sugar.
 - 2 teaspoons of baking powder.
 - 1/2 teaspoon of baking soda.
 - 1/2 teaspoon of salt.
 - 1 large egg.
 - 3 tablespoons unsalted butter.
 - 1-2 teaspoons more butter
2. Whisk the flour, sugar, baking powder, baking soda, and salt in a medium bowl.
3. Melt the 3 tablespoons of butter.
4. Whisk the egg and melted butter into the milk until combined.
5. Pour the milk mixture into the dry ingredients, and whisk until just combined (a few lumps should remain.)
6. Heat a nonstick griddle/frypan on medium heat until hot; grease with a teaspoon or two of butter.
7. Pour 1/4 cup of batter onto the skillet. Repeat in various disjoint places until there is no spare room on the skillet. Leave gaps of 1cm between pancakes.
8. Cook until large bubbles form and the edges set (i.e. turn a slightly darker color and are no longer liquid,) about 2 minutes.
9. Using a spatula, flip pancakes, and cook a little less than 2 minutes longer, until golden brown.
10. If there is still unused batter, go to 5; else, top pancakes with maple syrup and butter, and eat.

“simple” is someone who is (1) able to measure out quantities of various foods, and (2) knows the meaning of various culinary operations like “whisk” and “flip.” If we wanted, we could make an algorithm that includes additional steps that define “whisking” and “flipping”: i.e. at each step where we told someone to whisk the flour, we could instead have

- (a) Grab a whisk. If you do not know what a whisk is, go to this [Wikipedia article](#) and grab the closest thing to a whisk that you can find. A fork will work if it is all that you can find.
- (b) Insert the whisk into the object you are whisking.
- (c) Move the whisk around through the object you are whisking in counterclockwise circles of varying diameter, in such a fashion to mix together the contents of the whisked object.

In this sense, we can extend our earlier algorithm to reflect a different notion of “simple,” where we no longer assume that our person knows how to whisk things. It still describes the same sets of steps, and in this sense is still the “same” algorithm – it just has more detail now!

This concept of “adding” or “removing” detail from an algorithm isn’t something that will always work; some algorithms will simply demand steps that cannot be implemented on some systems. For example, no matter how many times you type “sudo apt-get 2 cups of flour,” your laptop isn’t going to be able to implement our above pancake algorithm. As well, there may be times where a step that was previously considered “simple” becomes hideously complex on the system you’re trying to implement it on!

We’re not going to worry too much about the precise definition of “simple” in this class, because we’re not writing any code here (and so our notion of “simple” isn’t one we can precisely nail down!) Instead, we’re going to just make sure that our algorithms consist of steps that we understand, and focus instead on trying to **prove** that these algorithms produce the desired output.

For instance, consider the following algorithm to find the prime factorization of any integer $n \geq 2$:

Init: Take in as input any integer n that is at least 2. Initialize our “factor” variable f to be equal to 2.

1. If $f \mid n$, print “ f .” Then replace n with $\frac{n}{f}$, and go to 1.
2. Otherwise, if $f \nmid n$:
 - (a) If $n = 1$, halt.
 - (b) Otherwise, replace f with $f + 1$ and go to 1.

To illustrate this algorithm, we run it on the input $n = 90$ at right. Its outputs were the numbers 2, 3, 3, 5, which in product form $2 \cdot 3 \cdot 3 \cdot 5 = 90$, as desired!

We seek to prove that this algorithm will **always** work: that is, that for any positive integer $n \geq 2$ this process will generate a prime factorization of n . To do this, we will prove the following three properties:

- First, we will prove that this process always **halts**: that is, if we input any integer $n \geq 2$ into this process, it will not run forever and will eventually halt.
- Then, we will prove that this process does not **crash**: that is, we will prove that every step of this process is well-defined, and we never have any steps where we’re dividing by 0 or claiming that something exists when it doesn’t.

Current step	n	f	Next step	Output
Initialization	90	2	1	
1	45	2	1	2
1			2	
2		3	1	
1	15		1	3
1	5		1	3
1			2	
2		4	1	
1			2	
2		5	1	
1	1		1	5
1			2	
2	(halt)			

- Finally, we will prove that the process's outputs **have the desired property**: that is, we will prove that the outputs of this process actually form a prime factorization of n .

In general, most proofs that a given algorithm “works” will have this sort of feel! Ensuring that a given program doesn't either loop forever or crash is typically the first two things anyone does when debugging. With that out of the way, we can typically turn to actually studying the outputs, and from there prove that it has the desired properties.

We prove these three properties one-by-one here:

Claim. The prime factorization algorithm given earlier does not “crash:” that is, given any integer $n \geq 2$, every step in this algorithm is well-defined throughout its run.

Proof. The only comparisons and operations we perform in our algorithm are the following:

- At the start, we initialize f to be equal to 2.
- We repeatedly check if $f \mid n$; this is well-defined, as n is an integer and f is a positive integer (it starts at 2 and increases by increments of 1 from there.)
- If $f \mid n$, we replace n with $\frac{n}{f}$. By definition, this is well-defined if $f \mid n$.
- If $f \nmid n$, we check if $n = 1$; if $n \neq 1$, we add 1 to f . This is trivially a well-defined operation.

□

Claim. The prime factorization algorithm given earlier does not “run forever:” that is, given any integer $n \geq 2$, this algorithm will eventually halt on input n .

Proof. First, notice that if our algorithm eventually halts on input n , it does so because n was reduced to 1. The only way that n ever changes in our algorithm is if in step 1 we replace n with $\frac{n}{f}$; therefore, if n is ever reduced to 1, on the loop²⁰ prior to this happening we had $n = f$.

Therefore, it suffices to show that given any integer n , our algorithm eventually sets $n = f$.

To see why this must hold, notice that at the start of our algorithm we have $n \geq f$, as $n \geq 2$ by assumption and $f = 2$. In the case where $n = f = 2$ we're immediately done, so we can in fact assume that $n > f$ at the start of our algorithm. As well, on each loop of our algorithm we either decrease n (by dividing it by f) or increase f ; so eventually at some point in time we will have $n \leq f$.

If we transition to $n \leq f$ by using the $f \rightarrow f + 1$ step, then at this time we have $n = f$. This is because n and f are integers; so, if we originally had $n > f$, then n is at least 1 greater than f , and so adding 1 to f can make n and f equal at best.

So the only remaining case to study is if our algorithm goes from $n > f$ to $n \leq f$ by using the $n \rightarrow \frac{n}{f}$ step. I claim that in this scenario, it is impossible for $\frac{n}{f}$ to be less than f (and therefore after this step we have $n = f$, as desired.)

²⁰In an algorithm, we typically have a sequence of steps, each of which either leads into a “next” step or returns to an earlier step. A **loop** of an algorithm, roughly speaking, is a sequence of steps from one of those start steps until we return to another of those start steps. In the algorithm we're studying here, there are two possible loops: we could do $1 \rightarrow 1$ if $f \mid n$, or $1 \rightarrow 2 \rightarrow 1$ otherwise.

To see why, we proceed by contradiction: suppose that we had positive integers n, f such that $n > f$ and yet $\frac{n}{f} < f$. Let $\frac{n}{f} = d$, for shorthand. Then the observation here means that we can write $n = d \cdot f$, where d is a factor of n that is at least 2 (because $n > f \Rightarrow \frac{n}{f} > 1$) and strictly less than f (by contradictory assumption.)

However, we know that we started f at 2 and increased it one-by-one to its current size; as a result, at a strictly earlier stage the value stored in f must have been equal to d . But at that stage, our algorithm would have repeatedly divided n by d until there were no factors of d left in n ! In other words we've shown that it's impossible for d to be a factor of n if $d < f$, which contradicts our assumption earlier.

This completes our proof, as we've shown that in all cases we eventually get to a case where our algorithm halts. \square

Perhaps surprisingly, the proof that our algorithm always halts is the hardest of our three; with this established, proving that the algorithm actually generates a prime factorization is relatively simple!

Claim. Given any integer $n \geq 2$, the prime factorization algorithm given earlier will output a prime factorization of n .

Proof. At any step of our algorithm, consider the value formed by multiplying the value currently stored in n by all of the outputs generated by our algorithm so far. At the start, this is just n ; after each time n changes this is still n (because we replace n by $\frac{n}{f}$ and output f , the product of these terms doesn't change!), and at the end it is just the product of all of the outputs of our program (as $n = 1$ by the end.)

Therefore this program creates a factorization of n into integers. The only question, then, is whether they are all primes!

To see why each output f must be a prime number, consider what happens for any nonprime f . Because f starts off as 2 and grows one-by-one, if f is not prime it is also not 1 and so is composite, and thus we can write $f = ab$ for two positive integers $a, b \geq 2$.

However, because f started at 2 and grew until it was ab , along the way it was equal to both a, b and at that stage we repeatedly divided n by a, b until none of those factors remained in n . As a result, when we get to $f = ab$, we know that n no longer has any factors of either a or b left in it, and so must have $f \nmid n$! Therefore we never print any composite value of f ; i.e. the only values printed are primes, as desired. \square

Success! With this first algorithm thoroughly studied, we move to our next target:

2.2 The Euclidean Algorithm

Definition. Given two integers a, b , we say that the **greatest common divisor** of a and b , denoted $\gcd(a, b)$, is the largest integer d such that $d \mid a$ and $d \mid b$. Similarly, we say that the **least common multiple** $\text{lcm}(a, b)$ of two integers is the smallest positive integer divisible by both a and b .

For example:

- $\gcd(6, 4) = 2$; this is because 2 divides both of these numbers, while no larger integer divides both of these numbers (the only larger divisor of 6 is 6 itself, which does not divide 4.)
- Similarly, $\gcd(18, 36) = 18$, $\gcd(17, 21) = 1$ and $\gcd(27, 75) = 3$.

- $\gcd(0, n) = n$ for any positive integer n ; this is because $n \mid 0$ for any n (as noted before!) Similarly, $\gcd(1, n) = 1$ for any integer n , as the only positive divisor of 1 is 1 (which divides all other integers.)
- If $\gcd(a, b) = d$, then $\gcd(a/d, b/d) = 1$; this is because d contained all of the common factors between a and b , and so dividing by d leaves these numbers with no factors in common at all.
- $\text{lcm}(4, 6) = 12$; this is because 12 has both 4 and 6 as factors, and no positive smaller multiple of 4 has 6 as a factor.
- Similarly, $\text{lcm}(18, 36) = 36$, $\text{lcm}(5, 7) = 35$ and $\text{lcm}(8, 28) = 56$.

We say that two numbers a, b are **relatively prime** if $\gcd(a, b) = 1$; this means that they have no factors in common.

Calculating the GCD of two numbers is a remarkably useful operation to be able to do:

- If you have a fraction $\frac{a}{b}$, a task we often want to do is “simplify” this fraction: that is, to write it as a ratio in which the numerator and denominator have no factors in common. For example, we typically would prefer to write something like $\frac{2}{3}$ in place of $\frac{34}{51}$ or $\frac{14}{21}$, even though they’re all the “same” thing.

The GCD lets us do this! Specifically, given any fraction $\frac{a}{b}$ in which the numerator and denominator are integers, if $d = \gcd(a, b)$, then $\frac{a/d}{b/d}$ is the “simplified” form of $\frac{a}{b}$.

- A task that comes up frequently in cryptography is finding two numbers a, b that are relatively prime: i.e. whose GCD is 1. (We don’t have time to get into it here, but read up on [RSA cryptography](#) for more on this!)
- In real life, we often have problems of the form “you have sets of size d_1, d_2, \dots, d_k , and want to divide all of these sets into blocks of the same size.” For instance:
 - You have a 244×176 bolt of cloth, and want to chop it up into $k \times k$ squares. What is the largest value of k for which you can do this?
 - You have 720 daisies, 96 roses, 80 lilies and 128 lilacs. You want to arrange these into k bouquets, each of which has the same number of daisies, roses and lilacs. What is the largest number of bouquets you can make, and what flowers go into each bouquet?
 - You’re a teacher, and have three streams of the same class with enrolment numbers 312, 177 and 243 respectively. You want to divide students in each stream into groups, and want to have the same group size for all groups across all classes. What is the largest value of k that you can set the group sizes to?

In all of these situations, the GCD of the numbers d_1, \dots, d_k is the largest block size that you can pick, and solves this problem.

In this section, we present an algorithm for calculating the GCD of two numbers: namely, the Euclidean algorithm! To define it, we first describe a handful of useful operations:

Definition. Given two positive integers a, b , we say that the **quotient** of $\frac{a}{b}$ is $\lfloor \frac{a}{b} \rfloor$, i.e. the quotient is $\frac{a}{b}$ “rounded down.” Similarly, the **remainder** of $\frac{a}{b}$ is b times the difference between $\frac{a}{b}$ and its quotient.

Equivalently, we say that the **quotient** of $\frac{a}{b}$ is the largest number of times q we can subtract b from a and still have a nonnegative integer left over, and the **remainder** is just what's left after we do that subtraction (i.e. the remainder is $a - qb$.)

For example, the quotient of $\frac{34}{5} = \boxed{6}$. This can be seen by using the first definition (because $\lfloor \frac{34}{5} \rfloor = \lfloor 6.8 \rfloor = 6$), or the second (we can subtract 5 from 34 *six* times, and will be left with 4 at the end of this process, from which we cannot take away another 5 and still have a nonnegative integer.) As well, the remainder of $\frac{34}{5}$ is $\boxed{4}$; this can be seen by either the first definition (because $5 \cdot (6.8 - 6) = 5 \cdot .8 = 4$) or the second (after subtracting 5 from 34 six times, we had 4 left over.)

We prove that these definitions are equivalent here:

Claim. The above definitions for the quotient and remainder are equivalent.

Proof. Take any two positive integers a, b .

Observe that if $q = \lfloor \frac{a}{b} \rfloor$, then by the definition of “rounding down” we have $0 \leq \frac{a}{b} - q < 1$. Multiplying through by b gives us $0 \leq a - bq < b$. This means that q is the largest multiple of b that we can subtract from a and still have a nonnegative value left over, because subtracting one more copy of b from all sides would give us $a - (b + 1)q < 0$. In other words, any quotient q that satisfies the first of our definitions satisfies the second, as well!

Similarly, if q is the largest number of copies of b we can take away from a and still have a positive value, then $0 \leq a - bq < b$ by definition. Reversing the steps above gives us $0 \leq \frac{a}{b} - q < 1$; i.e. q is $\frac{a}{b}$ “rounded down,” and thus that these definitions are actually equivalent.

Similarly, if we let $r = b(\frac{a}{b} - q)$ be the remainder under our first definition, then simplifying the right-hand side gives us $r = a - bq$ (i.e. the second definition of the remainder); so these definitions are also equivalent. \square

As a consequence of the above, we have the following:

Corollary 2. For any two positive integers a, b , if q is the quotient of $\frac{a}{b}$ and r is the remainder, then $a = bq + r$, and $0 \leq r < b$.

Proof. Take any two positive integers a, b , and let q, r be the quotient and remainder of $\frac{a}{b}$. We saw above that $a - bq = r$, and that $0 \leq a - bq < b$; therefore we have $0 \leq r < b$. As well, rearranging $a - bq = r$ gives us $a = bq + r$, as desired. \square

The quotient and remainder are nicely related to the process of finding the GCD of two numbers, as the following lemma shows:

Lemma. For any two positive integers a, b , if r is the remainder of $\frac{a}{b}$, we have $\gcd(a, b) = \gcd(b, r)$.

Proof. To prove that these two pairs have the same greatest common divisor, we'll actually go one step further and prove that each pair's set of divisors are the same! That is: let

$$\begin{aligned} D_{a,b} &= \{d \in \mathbb{N} \mid d \mid a \text{ and } d \mid b\}, \\ D_{b,r} &= \{d \in \mathbb{N} \mid d \mid b \text{ and } d \mid r\}, \end{aligned}$$

We seek to show that these sets are the same. Notice that if we can do this, then clearly the largest element in these sets must be the same. Because $D_{a,b}$ is the set of all divisors common to a, b

and $D_{b,r}$ is the set of all divisors common to b, r , this would mean that $\gcd(a, b) = \gcd(b, r)$, as desired.

We do this in two steps:

- Take any $d \in D_{a,b}$. By definition, this means that $d \mid a$ and $d \mid b$; i.e. that we can find integers k, l such that $dk = a$ and $dl = b$. We know that $a - bq = r$ from our work above; therefore we have $dk - dlq = r$, i.e. $d(k - lq) = r$. In other words d is a divisor of r !

This means that $d \in D_{b,r}$, as we've just shown that d is a divisor of both b and r . In other words, every element of $D_{a,b}$ is also in $D_{b,r}$; that is, $D_{b,r}$ "contains" all of $D_{a,b}$!

- Now, we reverse this process: take any $d \in D_{b,r}$. By definition, this means that $d \mid b$ and $d \mid r$; i.e. that we can find integers m, n such that $dm = b$ and $dn = r$. We know that $a = bq + r$ from our work above; therefore we have $a = dmq + dn$, i.e. $a = d(mq + n)$. In other words d is a divisor of a !

This means that $d \in D_{a,b}$, as we've just shown that d is a divisor of both a and b . In other words, every element of $D_{b,r}$ is also in $D_{a,b}$; this observation, plus our earlier work, tells us that $D_{a,b} = D_{b,r}$, as desired.

□

We can use this result to define the **Euclidean algorithm**, a fast and efficient way to calculate the GCD of any two numbers:

Definition. The **Euclidean algorithm** is the following process for calculating the GCD of two positive integers a, b :

Init: Take in as input any two positive integers a, b . Initialize r by setting it equal to the remainder of $\frac{a}{b}$.

1. If $r = 0$, halt, and print b .
2. Otherwise, replace a with b , replace b with r , and recalculate r on this new pair of values a, b .

As before, we have three tasks that we must perform to prove that this algorithm works: we must prove that its steps are **well-defined**, that the program **eventually halts**, and that its output is indeed **correct** (in this case, that it calculates the GCD.) We do this here:

Claim. The Euclidean algorithm does not "crash:" that is, given any two positive integers a, b , every step of the Euclidean algorithm is well-defined.

Proof. The only comparisons and operations we perform in our algorithm are the following:

- We initialize r by setting it equal to the remainder of $\frac{a}{b}$; because a, b are both positive integers, this is well-defined as shown before.
- On each step, if $r \neq 0$, we simply replace a with b , b with r , and recalculate r as the remainder of this new pair of values a, b . Because each remainder r is nonnegative and we only go to this step if $r \neq 0$, we know that this new pair of values for a, b are still positive integers, and so the remainder is still well-defined.

□

Claim. The Euclidean algorithm does not “run forever:” that is, given any two positive integers a, b , the Euclidean algorithm will eventually halt.

Proof. This is actually pretty straightforward! Look at the value stored in the “b” slot. On each loop, we replace b with r , which is a nonnegative integer strictly less than b (as shown before.) Therefore, the value in b decreases on each loop, but stays nonnegative. Similarly, because the remainder r we calculate is always nonnegative but less than b , the remainder values r must also decrease but stay nonnegative.

Therefore, this process must eventually stop; if the remainders are decreasing, integers, and nonnegative at each step, then they must eventually become 0. \square

Claim. Given any two positive integers a, b , the Euclidean algorithm outputs the GCD of a and b .

Proof. Recall that as shown before, if a, b are any two positive integers and r is the remainder of $\frac{a}{b}$, then $\gcd(a, b) = \gcd(b, r)$. Therefore, on each loop of the Euclidean algorithm, the GCD of the values stored in the a, b values does not change!

At the start, this value is the GCD of our two inputs, by definition. At the end, this value is $\gcd(b, 0)$, because we halt precisely when $r = 0$. As noted before, $\gcd(b, 0) = b$. But b is the value we output! Therefore, our algorithm works, as claimed. \square

Success! We’ve proven that this algorithm works.

2.3 Modular Arithmetic

We close our discussion of the integers with one last, remarkably useful concept: the idea of **modular arithmetic**.

Definition. Take any three integers $a, b, n \in \mathbb{Z}$. We say that a is **equivalent to b modulo n** , and write $a \equiv b \pmod{n}$, if $a - b$ is a multiple of n .

For example,

- $21 \equiv 5 \pmod{8}$; this is because $21 - 5 = 16 = 2 \cdot 8$ is a multiple of 8.
- $-19 \equiv 7 \pmod{2}$; this is because $-19 - 7 = -26 = (-13) \cdot 2$ is a multiple of 2.
- $14 \not\equiv 18 \pmod{5}$; this is because $-14 - 18 = -32$ is not a multiple of 5.
- For any $a, b \in \mathbb{Z}$, $a \equiv b \pmod{1}$; this is because $a - b$ is always a multiple of 1 (as any integer is a multiple of 1!)
- For any $a, b \in \mathbb{Z}$, we only have $a \equiv b \pmod{0}$ if $a = b$, and have $a \not\equiv b \pmod{0}$ otherwise. This is because $a \equiv b \pmod{0}$ means that $a - b$ is a multiple of 0, and the only multiple of 0 is 0 itself; i.e. we’d have $a - b = 0$, which forces $a = b$.
- For any $a, b \in \mathbb{Z}$, we have $a \equiv b \pmod{2}$ if and only if a, b are both even or a, b are both odd. This is because $a \equiv b \pmod{2}$ holds if and only if $a - b$ is a multiple of 2, i.e. $a - b$ is even, and this only happens when a, b are the same **parity**: i.e. when they’re both even or both odd.

Modular arithmetic is **remarkably useful** for a number of things:

- Calculating remainders modulo n is key to essentially the entire field of cryptography; there is essentially no way to keep information secret in the modern world that does not use modular arithmetic in some way.
- Modular arithmetic is also used in every sort of “checksum” property. That is: in essentially every piece of information in the modern world (bar codes, bank account numbers, library book numbers), you will find a few extra digits appended to the end calculated by taking your information mod n for some value of n to help someone look for errors. We’ll talk more about this in the error-correcting codes section!
- Working modulo 2 is fundamental to arithmetic and working with binary strings, the foundation of modern computing!

To help us get some practice with modular arithmetic, we work two problems here: one abstract / proof-oriented, and one that’s just cute and short:

Claim. If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $a + c \equiv b + d \pmod{n}$, $ac \equiv bd \pmod{n}$, and $a^k \equiv c^k \pmod{n}$ for any positive integer k . In other words, we can “perform arithmetic” modulo n .

Proof. Note that by definition, if $a \equiv b \pmod{n}$ then $a - b$ is a multiple of n ; that is, we can write $a - b = kn$ for some integer k . Similarly, if $c \equiv d \pmod{n}$, then $c - d$ is a multiple of n , and we can write $c - d = ln$ for some integer l .

By adding these equations together, we get $(a + c) - (b + d) = kn + ln = (k + l)n$; that is, that $(a + c) - (b + d)$ is a multiple of n . This means that $a + c \equiv b + d \pmod{n}$, as desired.

Conversely, if we take $a - b = kn$ and multiply both sides by c , we get $ac - bc = kcn$; similarly, if we take $c - d = ln$ and multiply by b we get $bc - bd = bln$. Adding these equations together gives us $ac - bc + bc - bd = kcn + bln = (kc + bl)n$; i.e. $ac - bd$ is a multiple of n . This means that $ac \equiv bd \pmod{n}$, as claimed.

Finally, take our result that $ac \equiv bd \pmod{n}$ and consider the case where $c = a, d = b$. This tells us that if $a \equiv c \pmod{n}$ then $a^2 \equiv c^2 \pmod{n}$. Now, combine the statements $a \equiv c \pmod{n}$ and $a^2 \equiv c^2 \pmod{n}$; this gives you $a^3 \equiv c^3 \pmod{n}$. Repeatedly doing this gives you $a^k \equiv c^k \pmod{n}$, as desired. \square

Claim. The last digit of 213047^{129314} is 9.

Proof. The clever thing here is not that we can calculate this (if we just wanted an answer, we could plug this into WolframAlpha), but rather that we can calculate this *easily*! That is: we can use modular arithmetic to find this number with almost no calculation or effort.

To do so, make the following observations:

- First, $213047 \equiv 7 \pmod{10}$, and in general any number is equivalent to its last digit modulo 10. This is by definition: any number minus its last digit will have a last digit of 0, which means it’s a multiple of 10!
- Therefore, we have that $213047^{129314} \equiv 7^{129314} \pmod{10}$, by using our “exponentiation” result from earlier.
- Now, notice that $7^2 = 49 \equiv 9 \pmod{10}$, and thus that $7^4 = 7^2 \cdot 7^2 \equiv 9 \cdot 9 \pmod{10}$. $9 \cdot 9 = 81 \equiv 1 \pmod{10}$, so we have that $7^4 \equiv 1 \pmod{10}$. As a result, we have that for any k , $7^{4k} = (7^4)^k \equiv 1^k = 1 \pmod{10}$

- Therefore, because $129314 = 129300 + 12 + 2$, and any multiple of 100 is a multiple of 4, we have that $7^{129314} = 7^2 \cdot 7^{\text{a multiple of } 4} \equiv 7^2 \cdot 1 \pmod{10}$. $7^2 \equiv 9 \pmod{10}$, as noted before; therefore our entire expression is equivalent to 9 modulo 10.
- Finally, as noted before: any number is congruent to its last digit modulo 10. Therefore, because we've shown that our number 213047^{129314} is congruent to 9 modulo 10, we know that its last digit is 9, as claimed.

□

Chapter 3: Induction

Weeks 4-5

UoA 2018

3.1 Induction: Definitions

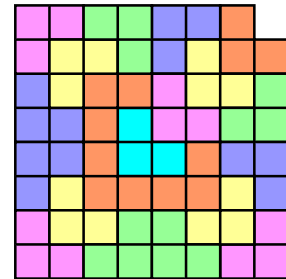
Sometimes, in mathematics, we will want to study a statement $P(n)$ that depends on some variable n . For example:

1. $P(n)$ = “The sum of the first n natural numbers is $\frac{n(n+1)}{2}$.”
2. $P(n)$ = “If $q \geq 2$, we have $n \leq q^n$.”
3. $P(n)$ = “Every polynomial of degree n has at most n roots.”
4. $P(n)$ = Take a $2^n \times 2^n$ grid of unit squares, and remove one square from the top-right-hand corner of your grid. The resulting shape can be tiled²¹ by $\begin{smallmatrix} \square & \square \end{smallmatrix}$ - shapes.

For any fixed n , we can usually use our earlier proof methods to prove that the claim holds! For instance, let $P(n)$ be the fourth example above, and consider $P(3)$, which is the claim that if we take a 8×8 grid of squares and delete the top-right-hand corner square, we can tile the rest of the shape with $\begin{smallmatrix} \square & \square \end{smallmatrix}$ tiles. We can prove this by construction by just giving an explicit way to do it: see the drawing at right!

However, sometimes we will want to prove that one of these statements holds for **every** value $n \in \mathbb{N}$. How can we do this?

The answer here is **mathematical induction**! Mathematical induction is just a formal way of writing up our “building-block plus preserved property” process, in a way that will hopefully let us avoid everyone having the same shoe size. We describe it here:



- To start, take a claim $P(n)$ that we want to prove holds for every $n \in \mathbb{N}$.
- The first step in an inductive proof is the **base step**: in this step, we explicitly prove that the statement P holds for a few small cases using normal proof methods (typically construction or just calculation.)

Usually you just prove $P(0)$, but sometimes you start with $P(1)$ if your claim is one where 0 is a “dumb” case, or prove a handful of cases like $P(0), P(1), P(2), P(3)$ to get the hang of things before moving on. You can think of this as the “building block” step from before!

- With this done, we move to the **induction step**: here, we prove the statement

$\forall n, \text{ if } P(k) \text{ holds for all } k \leq n, \text{ then } P(n+1) \text{ also holds.}$

Because this is an implication, i.e. an if-then proof, we usually prove it directly by assuming that $P(k)$ holds for all $k \leq n$ (i.e. start with the “if”), and use assumption this to conclude that $P(n+1)$ holds (conclude with the “then.”)

Once we’ve done these two steps, I claim that we’ve proven that $P(n)$ holds for all $n \in \mathbb{N}$! To see why, take any natural number n , like $n = 17$ or $n = 3 \cdot 10^8$. If you want to see why $P(n)$ holds, simply use the following logic:

²¹We say that a shape S can be tiled by a tile T if you can completely cover all of S with copies of T , so that none of those copies overlap or stick off the side of S . You’re allowed to rotate and flip the tile T when doing this.

- By our “base case” reasoning, we know that $P(0)$ is true.
- Because $(P(0) \text{ and } P(1) \text{ and } \dots \text{ and } P(n)) \Rightarrow P(n+1)$, if we let $n = 0$, this tells us that $P(0) \Rightarrow P(1)$. Because $P(0)$ is true, this means $P(1)$ is true.
- Again, because $(P(0) \text{ and } P(1) \text{ and } \dots \text{ and } P(n)) \Rightarrow P(n+1)$, if we let $n = 1$, this tells us that $(P(0) \text{ and } P(1)) \Rightarrow P(2)$; so because we know that $P(1)$ is true, this means $P(2)$ is true as well!
- Similarly, $P(0), P(1)$ and $P(2)$ tell us $P(3)$ is true,
- ... which tells us $P(4)$ is true,
- ... which tells us $P(5)$ is true,
- ...
- ... which eventually tells us that $P(n)$ is true, for every n !

Sometimes people will call this kind of induction **strong induction**. By contrast, in “weak induction” we proceed as follows:


- Base case: as before, we prove our claim for $P(0)$, or maybe for the first few values where our claim is true.
- Inductive step: we prove that $P(n) \Rightarrow P(n+1)$.


The difference here is just in the inductive step: in this form of induction, you only use the fact that $P(n)$ is true to prove $P(n+1)$ is true, while in strong induction you get to remember “hey, I’ve actually proved this for $P(0)$ and $P(1)$ and ... all the way up to $P(n)$ ” when you try to prove $P(n+1)$ also holds. In this sense, weak induction is strictly less useful than strong induction: all strong induction does is just give you more facts to use when you’re doing your inductive step! However, many inductive proofs can be done with just “weak” induction; so if you’re just using the $P(n)$ case in your inductive step, don’t worry!

The way I usually think of inductive proofs is to think of **toppling dominoes**. Specifically, think of each of your $P(n)$ propositions as individual dominoes – one labeled $P(0)$, one labeled $P(1)$, one labeled $P(2)$, and so on/so forth. With our inductive step, we are insuring that all of our dominoes are *lined up* – in other words, that if we’ve knocked over some of them, the “next one” will also be knocked over. Then, we can think of the base step as “knocking over” the first domino. Once we do that, the inductive step makes it so that all of the later dominoes also have to fall, and therefore that our proposition must be true for all n (because all the dominoes fell!)

3.2 Induction: Two Examples

To illustrate how these kinds of proofs go, let’s go back to our tiling problem, and prove that we can tile this grids for **every** $n \in \mathbb{N}$! (As an added bonus, let’s prove it for grids where we remove one square from anywhere, not just the top-right-hand corner!)

Claim. For any $n \in \mathbb{N}$, take a $2^n \times 2^n$ grid of unit squares, and remove one square from somewhere in your grid. The resulting grid can be tiled by  - shapes.

Proof. As suggested by the section title, we proceed by induction, where our proposition $P(n)$ is “we can tile a $2^n \times 2^n$ grid of 1×1 squares with one square deleted by using  - shapes.”

Base case: we want to prove $P(0)$. So: what *is* $P(0)$?

Well: for $n = 0$, we have a $2^0 \times 2^0 = 1 \times 1$ grid, which we’ve removed a 1×1 square from. In other words, we have **nothing**.

If you want, you can think of “nothing” as being something we can trivially cover by placing no three-square shapes!

Alternately, you can decide that 0 is a stupid case and look at $n = 1$ instead. For $n = 1$, we simply have a 2×2 grid with one square punched out. As this *is* one of our three-square shapes, we are done here; just place a tile on top of our grid!

Either starting place is fine. In general, I recommend doing as many base cases as you need to do in order to feel comfortable with the pattern and believe that you’ve done something concrete! Most of the time, though, the base case will feel kinda silly; don’t worry about this! The inductive step will do all of the heavy lifting for us.

Inductive step: We want to prove that if we know that our claim holds up to n , then it holds for $n + 1$ as well; formally, this means that we want to show that if $P(0)$ and $P(1)$ and \dots and $P(n)$ all hold, then $P(n + 1)$ must follow.

In this problem in particular, this means that we’re assuming that we can tile a $2^k \times 2^k$ -grid with a square deleted for any $k \leq n$, and want to use this assumption to tile a $2^{n+1} \times 2^{n+1}$ grid with a square deleted.

To do this, take any $2^{n+1} \times 2^{n+1}$ grid with a square deleted. Divide it into four $2^n \times 2^n$ squares by cutting it in half horizontally and vertically. Finally, by rotating our grid if needed, make it so that the one missing square is in the upper-right hand corner.

Take this grid, and carefully cut out one three-square shape in the center as drawn at right.

Now, look at each of the four $2^n \times 2^n$ squares in this picture. They all are missing exactly one square: the upper-right hand one because of our original setup, and the other three because of our placed three-square-shape.

By our inductive hypothesis $P(n)$ we know that all of these smaller squares can be tiled! Doing so then gives us a tiling of the whole shape; in other words, we’ve shown how to use our $P(n)$ results to get a tiling of the $2^{n+1} \times 2^{n+1}$ grid.

As this completes our inductive step, we are thus done with our proof by induction. \square

The claim we proved above — one where we were some sense “growing” or “extending” a result on small values of n to get to larger values of n — is precisely the kind of question that induction is set up to solve! The Fibonacci numbers, which we introduce in the next question, is another object where this sort of “extension” approach is useful to consider.

Definition. The **Fibonacci numbers** f_n are defined as follows:

- $f_0 = 0, f_1 = 1$.
- For any $n \geq 2$, $f_n = f_{n-2} + f_{n-1}$.

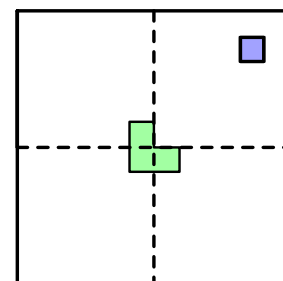
This sort of definition is called a **recursive**²² definition²³. To illustrate how it works, let’s use it to calculate the first few values of the Fibonacci sequence! We know that $f_0 = 0, f_1 = 1$ by definition.

To find f_2 , we can use the fact that for any $n \geq 2$, $f_n = f_{n-2} + f_{n-1}$ to calculate that.

$$f_2 = f_0 + f_1 = 0 + 1 = 1.$$

We can calculate further values of f_n similarly (see right!)

When doing this, you’ll likely notice a number of interesting properties about the Fibonacci sequence: see



$$\begin{aligned} f_3 &= f_1 + f_2 = 1 + 1 = 2, \\ f_4 &= f_2 + f_3 = 1 + 2 = 3, \\ f_5 &= f_3 + f_4 = 2 + 3 = 5, \\ f_6 &= f_4 + f_5 = 3 + 5 = 8, \\ f_7 &= f_5 + f_6 = 5 + 8 = 13, \\ f_8 &= f_6 + f_7 = 8 + 13 = 21, \\ f_9 &= f_7 + f_8 = 13 + 21 = 34, \\ f_{10} &= f_8 + f_9 = 21 + 34 = 55, \\ f_{11} &= f_9 + f_{10} = 34 + 55 = 89, \end{aligned}$$

²²For a definition of recursion, see footnote 23.

²³For a definition of recursion, see footnote 22.

for a ton of weird/beautiful properties these numbers have! We prove one of these properties here:

Claim: For any $n \in \mathbb{N}$, the n -th Fibonacci number is even if and only if n is a multiple of 3.

Proof. Let $P(n)$ denote the claim “(the n -th Fibonacci number is even) \Leftrightarrow (n is a multiple of 3).” We want to prove that $P(n)$ holds for all $n \in \mathbb{N}$, and proceed to prove this claim by induction.

Our **base cases** are pretty easy to check! We calculated the Fibonacci numbers from f_0 to f_{12} above, and we can see that the only ones that are even are f_0, f_3, f_6, f_9 and f_{12} ; so we know that $P(0), P(3), P(6), P(9)$, and $P(12)$ all hold.

We now move to the **inductive step**: here, we want to prove $(P(0) \wedge P(1) \wedge P(2) \wedge \dots \wedge P(n)) \Rightarrow P(n+1)$. We start with what we’re assuming, namely $P(0) \wedge P(1) \wedge \dots \wedge P(n)$: that is, we’re assuming that the k -th Fibonacci number is even if and only if it is a multiple of 3, for every $k \in \{0, 1, \dots, n\}$.

We want to prove $P(n+1)$, i.e. that the $n+1$ -th Fibonacci number is even if and only if it is a multiple of 3.

So: let’s consider cases! There are two possible cases for the value $n+1$: either it is a multiple of 3, or it’s not.

- If $n+1$ is a multiple of 3, we can write $n+1 = 3k$ for some $k \in \mathbb{Z}$. Notice that this means that $n = 3k-1$ and $n-1 = 3k-2$, and in particular that both of the values $n, n-1$ are not multiples of 3!

As a result, our inductive assumption tells us that f_n, f_{n-1} are both not even, because they’re not multiples of 3! But being not-even just means that these numbers are both odd. As a result, because $f_{n+1} = f_n + f_{n-1} = \text{odd} + \text{odd} = \text{even}$, we have shown that f_{n+1} is even in this case.

- If $n+1$ is not a multiple of 3, then $n+1$ either has remainder 1 or 2 when we divide it by 3; this is because any number has remainder 0, 1 or 2 when divided by 3. This means we can write $n+1 = 3k+1$ or $3k+2$, for some $k \in \mathbb{Z}$.

As a result, we can see that of the two numbers $n, n-1$, exactly one of them is a multiple of 3; if $n+1 = 3k+1$ then $n, n-1 = 3k, 3k-1$, and if $n+1 = 3k+2$ then $n, n-1 = 3k+1, 3k$. As a result, our inductive hypothesis tells us that exactly one of f_n, f_{n-1} are odd, and the other is even.

Therefore, because $f_{n+1} = f_n + f_{n-1} = (\text{one odd number plus one even number}) = \text{odd}$, we have shown that f_{n+1} is odd in this case.

So, by using strong induction, we have proven that f_n is even if and only if it is a multiple of 3! \square

We’ll use induction several times in our next subject, graph theory; so if you’d like more examples, read on!

Chapter 4: Graph Theory

Weeks 5-6

UoA 2018

4.1 Where it all started: Königsberg

Consider the following problem:

Puzzle. The city of Königsberg is divided by the river Pregel into four parts: a northern region, a southern region, and two islands. These regions were connected by seven bridges, drawn in red in the map at right.

Can you come up with a path through the city that starts and ends at the same place, and walks over each bridge exactly once?

This problem was something that the inhabitants of the city had tried to solve for some time; as a result it became somewhat famous, and came to the attention of Leonhard Euler (arguably the most prolific mathematician in history.) To solve it, he started the field of **graph theory**; a branch of mathematics centered around studying **objects** and **the connections between them**.

While graph theory was originally seen as being a recreational branch of mathematics with few applications beyond **coloring maps**, in recent years its power to describe networks has made it the perfect tool for studying many problems in the modern world. Graphs can be used to model the internet, social networks, the spread of diseases through a population, travel, computer chip design, and countless other phenomena; they are everywhere

In short, graph theory is **amazing**. In these notes, we'll start by building up some definitions that will let us talk about graphs and their properties; from there, we'll move to solving various famous problems in graph theory, ranging from the Bridges of Königsberg to more recent phenomena such as the traveling salesman problem, the shortest path problem, and the graph isomorphism problem. These notes are just the tip of the iceberg; if you're interested in further resources, books like **Douglas West's Graph Theory** have an immense wealth of problems, theorems and applications! Also feel free to talk to me; I love this stuff!

4.2 What is a graph?

A **graph** is just any way of modeling a collection of objects and the connections between them. Here is one way to do this:

Definition. A **simple graph** G consists of two things: a set V of **vertices**, and another set E of **edges**, which we think of as distinct unordered pairs of distinct elements in V .

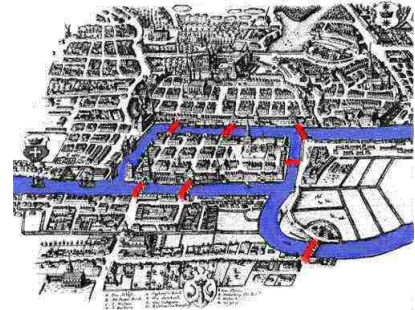
For example, the following describes a graph G :

- $V = \{a, b, c, d, e\}$
- $E = \{\{a,b\}, \{b,c\}, \{c,d\}, \{d,e\}, \{e,a\}\}$

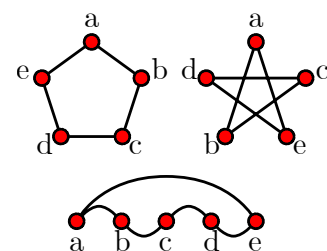
Given a graph $G = (V, E)$, we can **visualize** G by drawing its vertices as points on a piece of paper, and its edges as connections between those points. Several ways of drawing the graph G defined above are drawn at right.

Notice that edges don't have to be straight lines, and that we allow them to cross!

Just as there are many different ways to model the connections between a set of objects, there are other notions of graphs beyond that of a **simple graph**. Here are some such definitions:



A map of Königsberg, c. 1730.



Definition. A **simple graph with loops** is just like a simple graph G , except we no longer require the pairs of elements in E to be distinct; that is, we can have things like $\{v, v\} \in E$.

A **multigraph** is a simple graph, except we allow ourselves to repeat edges in E multiple times; i.e. we could have three distinct edges $e_1, e_2, e_3 \in E$ with each equal to the same pair $\{x, y\}$.

A **directed graph** is a simple graph, except we think of our edges as ordered pairs: i.e. we have things like $(x, y) \in E$, not $\{x, y\}$.

You can mix-and-match these definitions: you can have a directed graph with loops, or a multigraph with loops but not directions, or pretty much anything else you'd want!

In this course, we'll assume that graph means simple graph unless explicitly stated otherwise, and work mostly with simple graphs.

4.3 Famous graphs

There are many graphs that come up frequently in mathematics! Several families of these graphs are listed here:

Definition. The **complete graph on n vertices K_n** is defined as follows:

- $V = \{v_1, v_2, v_3, \dots, v_n\}$.
- $E = \{\{v_x, v_y\} \mid x, y \in \{1, 2, \dots, n\}, x \neq y\}$

The **complete bipartite graph on m, n vertices $K_{m,n}$** is defined as follows:

- $V = \{v_1, v_2, v_3, \dots, v_m\} \cup \{w_1, w_2, w_3, \dots, w_n\}$.
- $E = \{\{v_x, w_y\} \mid x \in \{1, 2, \dots, m\}, y \in \{1, 2, \dots, n\}\}$

The **cycle graph on n vertices C_n** is defined as follows:

- $V = \{v_1, v_2, v_3, \dots, v_n\}$.
- $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_1\}\}$

4.4 Graph properties: degree, walk, circuits

When we study objects using graph theory, there are certain properties that come up all of the time! We'll introduce useful properties throughout these notes, but three key ones to start with are the following:

Definition. In a graph $G = (V, E)$, we say that an edge $\{x, y\} \in E$ has x and y as its **endpoints**. We also say that a vertex $v \in V$ has **degree m** if there are exactly m different edges in E that have v as an endpoint.

Definition. In a graph $G = (V, E)$, we define a **walk**²⁴ of length n to be any sequence of n edges $\{v_0, v_1\}, \{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, v_n\}$, all in E . We say that this walk starts at v_0 and ends at v_n .

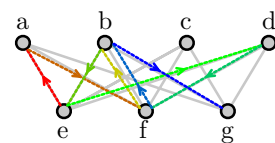
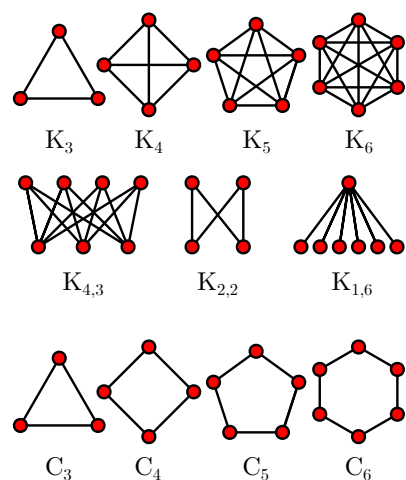
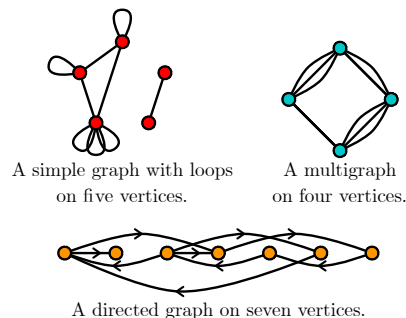
If $v_0 = v_n$, we call this walk a **circuit**.

If every edge in E shows up exactly once in the walk, then we call our walk an **Eulerian walk**.

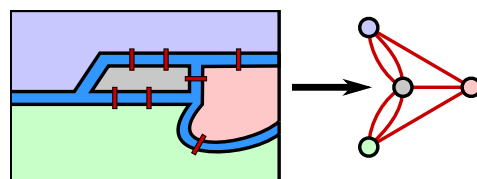
If there is a walk connecting any two vertices $x, y \in V$, we say that G is **connected**.

With this language at hand, we can turn the bridges of Königsberg problem into a task in graph theory! Specifically: consider the multigraph $G = (V, E)$ drawn at right, given by letting each of the four regions of land be a vertex, and representing the seven bridges as edges connecting these vertices.

²⁴Note: your coursebook calls these things **paths**. This is someone non-standard, as most authors require that a **path** does not repeat any vertex, which the coursebook doesn't require. For this course, feel free to use either term interchangeably; we won't mark you down regardless of which you use.



A walk from e to g on the graph $K_{4,3}$.



Converting Königsberg to a graph.

In the language of graph theory that we’ve built up, finding a path that starts and ends at the same place and walks over each bridge exactly once is just finding a **Eulerian circuit** in this graph!

So: when do those exist?

4.5 Eulerian circuits

In 1735, Euler presented the following theorem, arguably the first result ever proven in the field of graph theory, to the St. Petersburg Academy:

Theorem. A connected graph G has an Eulerian circuit if and only if it has no vertices of odd degree.

Proof. As this is an if and only if proof, i.e. a proof of equivalence, given any connected graph G we must prove two things:

- (♥): If G has an Eulerian circuit, it has no odd-degree vertices.
- (♣): If G has no vertices of odd degree, then it has an Eulerian circuit.

We start by proving (♥). Suppose that $G = (V, E)$ is a graph with a Eulerian circuit. Write down that Eulerian circuit here:

$$\{v_0, v_1\}, \{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_0\}.$$

Pick any vertex $x \in V$. Notice that each time x comes up in the above circuit, it does so twice: if $x = v_i$ for some i , it shows up in both $\{v_{i-1}, v_i\}$ and $\{v_i, v_{i+1}\}$. You can think of this as saying that each time our circuit “enters” a vertex along some edge, it must “leave” it along another edge!

As a result, any vertex x shows up an even number of times in the circuit we’ve came up with here. But our circuit is Eulerian; that is, it contains every edge in E exactly once. As a result, every vertex x shows up in an even number of edges in E ; that is, $\deg(x)$ is even for every vertex x , as claimed.

We now prove (♣). Suppose that G is a connected graph in which all of our vertices have even degree. Consider the following process for generating a cycle in G :

- Init: Pick a vertex v_0 at random from V . Think of v_0 as our current location, and our current walk as the empty walk.
1. If we are currently at some vertex v_i , randomly choose a vertex v_{i+1} so that the edge $\{v_i, v_{i+1}\}$ is not yet in our walk. Add $\{v_i, v_{i+1}\}$ to our walk, and update our current location to v_{i+1} .
 2. Repeatedly do step 1 above until we get back to v_0 .

Notice that because the degree of every vertex in G is even, step 1 in this process can never fail: if we are able to “enter” a vertex along some edge, there must be a corresponding edge we can “leave” on! Because G has a finite number of edges, we can’t get stuck on 1 forever as well; so we must eventually get back to v_0 . In other words, the process above generates a circuit! Call it C .

If this circuit is Eulerian, sweet; we’re done. If not, though, it’s not too hard to make it Eulerian! Simply do the following:

- Init: Take G , and delete C ’s edges from G . Because every vertex shows up an even number of times in a circuit (as shown earlier!), this doesn’t change our “all vertices have even degree” property.
1. If G has edges that aren’t in C , then (because G ’s connected) there must be some vertex v_i in our circuit that still has nonzero degree.

2. Starting from v_i , run our “find a circuit” algorithm, to get another circuit C' that starts and ends at v_i .
3. Now, “paste” that circuit C' into our original circuit, by traveling along C until we get to v_i , then taking the circuit C' which starts and ends back at v_i , and then resuming the original circuit C . We’ve made a bigger circuit!
4. If G still has edges, go to 1 and do it all again!

This process will “grow” our circuit on each pass, and is again guaranteed to work because our degrees stay even on each loop of our algorithm. So doing this repeatedly will generate an Eulerian circuit for us, and thus complete our proof! \square

4.6 Useful graph results: the degree-sum formula, bounding edges

In the above section we studied graphs from a “historical” perspective; that is, we proved the first theorem properly studied in graph theory, and built up a lot of machinery and useful concepts along the way!

To give ourselves some more practice with proofs and graphs, we prove a handful of smaller theorems in this subsection. These results are short and sweet, but also quite handy when studying graphs later on:

Lemma 3. (*The “degree-sum formula,” or “handshaking theorem.”*) *If G is a graph, then the sum of the degrees of the vertices in G is always twice the number of edges in G .*

Proof. We prove this by “counting” the number of times vertices in our graph show up as endpoints of edges in our graph. We do this in two ways:

- On one hand, every edge has two endpoints. Therefore, the total number of times vertices are used as endpoints in our graph is simply twice the number of edges.
- On the other hand, every endpoint is counted once when we calculate the degree of the corresponding vertex. Therefore, if we sum up the degrees of all of the vertices in our graph, this counts the total number of times vertices are used as endpoints in our graph.

Therefore we have that the sum of degrees of vertices in our graph is twice the number of edges, as they’re both valid ways of counting the same object! \square

Lemma 4. *If G is a connected multigraph with loops (i.e. we allow multiple edges, and also allow an edge to have both of its endpoints be equal) on n vertices, then G contains at least $n - 1$ edges.*

Proof. We proceed by induction on n . For $n = 0, 1$, this claim is trivially true, as we always have that E is a nonnegative number.

This establishes our base cases, so we now turn to the inductive step: here, we assume that our claim holds for all connected graphs on at most n vertices, and seek to use that assumption to prove that our claim holds for connected graphs on $n + 1$ vertices.

To do this, consider the following operation, called **edge contraction**. We define this as follows: take any graph G and any edge e in G with two distinct endpoints. We define G_e , the graph

that this edge, as follows: take G , delete e , and then combine e 's two endpoints together into a single vertex, preserving all of the other edges that the graph has along the way.

We draw examples of this process at right: here, we have started with a graph on six vertices, and then contracted one by one the edges highlighted in red at each step.

Notice that contracting an edge decreases the number of vertices by 1 at each step, as it “squishes together” two adjacent vertices into one vertex. It also decreases the number of edges by 1 at each step, as we are contracting an edge to a point!

Finally, notice that contracting an edge preserves the property that our graph is connected. To see why, take any walk

$$\{v_0, v_1\}, \{v_1, v_2\}, \dots \{v_{i-1}, v_i\}, \{v_i, v_{i+1}\}, \{v_{i+1}, v_{i+2}\}, \dots \{v_{n-1}, v_n\}$$

in our graph. Notice that if we contracted an edge $\{v_i, v_{i+1}\}$ in this walk, this would collapse the vertices v_i, v_{i+1} into some new vertex $v_{i \oplus i+1}$ and preserve all of the edges other than $\{v_i, v_{i+1}\}$. As a result, our walk would just become

$$\{v_0, v_1\}, \{v_1, v_2\}, \dots \{v_{i-1}, v_{i \oplus i+1}\}, \{v_{i \oplus i+1}, v_{i+2}\}, \dots \{v_{n-1}, v_n\},$$

and thus still connects the vertices v_0, v_n . Therefore, edge contraction cannot “break” any pre-existing walks, and so preserves the property that our graph is connected.

We can use this process to prove our claim via induction:

- Take any connected multigraph graph G on $n + 1$ vertices.
- Take any edge e in G with two distinct endpoints (such an edge exists, because G contains at least two different vertices and G is connected) and contract that edge. This gives us a new graph G_e , which is connected and contains n vertices.
- Therefore, by induction, we know that in G_e , the number of edges is at least $n - 1$.
- We also know that G has exactly one more edge than G_e .
- Therefore, in G , we know that we have at least $n - 1 + 1 = (n + 1) - 1$ edges. In other words, we’ve proven that our claim holds for graphs on $n + 1$ vertices, as desired!

□

Notice that this result applies to simple graphs as well, as any simple graph is certainly a multigraph!

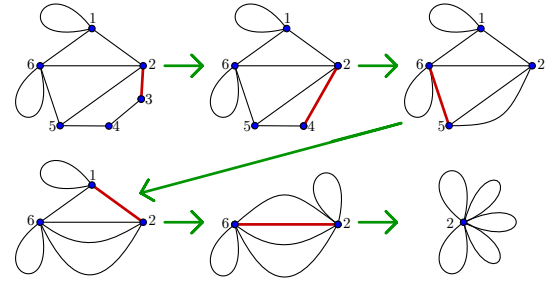
4.7 How Induction Can Go Wrong: Two Examples

In the above proof, we used induction to successfully prove a claim about graphs! Induction is often a great proof method to study graphs; however, it is also a tricky one that can sometimes go awry. In this section, we study two “failed” induction proofs, to illustrate some common pitfalls with this method:

Claim. All shoes in the world are the same size.

“Flawed” inductive proof: We proceed by induction. In specific, we seek to prove the stronger claim “Any set of n shoes in the world are all the same size;” if we make this set be the collection of all shoes in the world, then it will prove our claim.

Our base case is straightforward. For $n = 1$, our claim is that in any set of shoes containing one shoe, then all of the shoes are the



same size. This is trivially true: because there's only one thing in our set, it is automatically the "same" size as itself!

With the base case established, we move on to the inductive step. Here, we assume that in any set of up to n shoes, they are all the same size, and seek to use this assumption to prove that all of the shoes in any set of $n + 1$ shoes are the same size.

To do this, proceed as follows: take any set of $n + 1$ shoes. Number these shoes $1, 2, \dots, n, n + 1$, and split this set into two subsets $\{1, 2, \dots, n - 1, n\}$ and $\{2, 3, \dots, n, n + 1\}$, each containing just n shoes.

By our inductive hypothesis we know that all of the shoes in $\{1, 2, \dots, n - 1, n\}$ are the same size: let's call that size s . Induction also tells us that all of the shoes in $\{2, 3, \dots, n, n + 1\}$ are the same size; call that size s' .

We also know that these two sets overlap; specifically, they overlap on the shoes $\{2, 3, \dots, n - 2, n - 1\}$. Therefore the shoes in this overlap simultaneously have sizes s and s' . This forces $s = s'$, and therefore tells us that all of the shoes in our original set $\{1, 2, \dots, n, n + 1\}$ are the same size (namely, whatever size is equal to $s = s'$)

□

Before reading the next bit, look at this proof and try to come up with a flaw²⁵ in the logic!

Once you've tried this, read on...

Why this proof fails: Our inductive step only lets us go from n to $n + 1$ if $n \geq 2$! If we try to run our claim for $n = 1, n + 1 = 2$, then the two sets $\{1, 2, \dots, n - 1, n\}$ and $\{2, 3, \dots, n, n + 1\}$ are just $\{1\}$ and $\{2\}$. This means that their "overlap set" $\{2, 3, \dots, n - 2, n - 1\}$ is actually the empty set $\{\} = \emptyset$, and so we cannot conclude that $s = s'$ as there is nothing in the overlap! Therefore, this inductive proof is not valid.

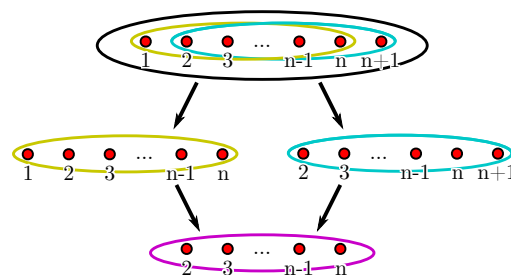
How to avoid this error: When doing a proof by induction, if you can, do a few base cases until you see a "pattern!" If you do this, then this will help you in two ways:

- First, that "pattern" is likely going to be what you prove holds in your inductive step. Proving that this works will involve just making that "pattern" formal; so seeing the pattern first will usually help you later on!
- Second, if you've seen the "pattern" in your base cases, then you probably won't ever be in the situation above. That is, if in your base case work you used the pattern to get to some more base cases, this means that you've kind-of checked that the inductive step does pick up from where your base case leaves off (assuming that your inductive step is that pattern made formal.)
- Finally, I should say that you don't have to do this: if you've written an inductive step that is a valid proof, then you're good! But doing a few more base cases never hurts.

I mention this in the graph theory chapter because it's related to a common mistake people make when writing induction proofs on graphs! To illustrate this, we consider a second "false" proof by induction:

Claim. If G is a simple graph in which the degree of every vertex is at least 1, then G is connected.

²⁵Note: a flaw in the logic isn't something like "the conclusion is false:" that's more of an argument against induction itself, which is something that we do believe! Instead, try to spot the place in the proof above where we have failed to apply induction properly!



“Flawed” inductive proof: We induct on the number of vertices in our graphs: that is, we will prove the claim “For any graph G on n vertices, if the degree of every vertex in G is at least 1, then G is connected.”

To avoid the issue we saw in our last inductive proof, we prove a handful of base cases:

- For $n = 1$, we would have one vertex. There is no simple graph on one vertex in which every vertex has degree at least 1 (we only have one vertex, and so can have no edges unless we have a loop, which is not allowed in a simple graph) so there are no graphs our claim applies to.

Therefore, the “if...then...” claim here is vacuously true; there’s nothing that makes the “if” part true, and so the entire thing cannot be falsified and thus is true.

- If that feels flimsy, $n = 2$ is pretty straightforward: the only graph on 2 vertices in which all vertices have degree 1 is $K_2 = \bullet \text{---} \bullet$, which is clearly connected.

- Just to be extra-safe, we look at $n = 3$ as well. In this case, there are only two graphs: either $K_3 = \triangle$ or $P_3 = \bullet \text{---} \bullet \text{---} \bullet$, both of which are also connected.

With this in place, we move to our induction step. That is: we assume that our claim holds for all graphs on at least n vertices, and seek to use this assumption to study graphs on $n + 1$ vertices.

To do this: take any graph G on n vertices, in which the degree of every vertex is at least 1. Add to this graph a new vertex v , and draw any number of edges from v to other vertices in G , making sure to draw at least one such edge. Call this new graph G' .

I claim that G' is connected. To see why, take any two vertices in G' . We have two possible cases:

- If both of the vertices x, y we’ve chosen are in the original graph G , then because G is connected by our inductive assumption, there is a path from x to y in G . Adding the new vertex v and some more edges certainly would not *break* such a path; therefore this path still exists in G' , and so x and y are connected.
- Otherwise, we picked one vertex x from the original graph G , while the other was v , our new vertex.

Because the degree of v was at least 1, there is some other vertex y in the original graph G that we’ve connected to v . Again, because G is connected by our inductive hypothesis, there is a path from x to y ; if we attach the $\{y, v\}$ edge to the end of this path, we have a path from x to v , and thus have shown that x and v are connected.




Therefore we’ve shown that any two vertices in G' , this new graph, are connected. Because we can create any graph on $n + 1$ vertices by taking a graph on n vertices and just adding a new vertex, we’ve therefore proven this claim for all graphs on $n + 1$ vertices, as desired. \square

As before, look at this proof and try to come up with a flaw before reading further!

Once you’ve tried this, continue on...

Why this proof fails: This proof fails in its last paragraph, where we say that we can get “any” graph on $n + 1$ vertices by taking a graph on n vertices and adding a vertex. While this is true on its face, in this proof we are making claims about **graphs in which**

the degree of every vertex is at least 1. In this case, it is false that you can create any such graph on $n + 1$ vertices by taking a graph on n vertices and adding a new vertex + at least one edge!

For example, consider the graph $K_2 \sqcup K_2 =$ . You cannot create this graph by taking any of the graphs  or $P_3 =$  on three vertices in which all vertices have degree at least 1 and adding a new vertex + edge. Therefore, our inductive argument is flawed: we have failed to write an argument that considers *all* possible graphs on $n + 1$ vertices.

How to avoid this error: When doing inductive proofs, it's dangerous to start with the n case and “grow” it to the $n + 1$ case: this makes it easy to make logical mistakes like this, where you assume that you've considered all of the possible $n + 1$ cases. In particular, this can easily happen with graphs, as it can be hard to make an inductive way to “grow” a graph on n vertices to a graph on $n + 1$ vertices that captures all possible graphs on $n + 1$ vertices.

Instead, I recommend taking a graph on $n + 1$ vertices and finding a way to “shrink” it to a graph on n vertices, and then using the inductive hypothesis/etc to make an argument about the original $n + 1$ vertex graph (like we did with edge contraction!)

4.8 Planar graphs

To close our graph theory section, we turn to a particularly beautiful concept: planarity!

Definition. We say a graph G is **planar** if we can draw it in the plane so that none of its edges intersect.

Sometimes, it will help to think of planarity in the following way:

Definition. We say that a connected graph G is **planar** if we can draw it on a sphere in the following fashion:

- Each vertex of G is represented by a point on the sphere.
- Each edge in G is represented by a continuous path drawn on the sphere connecting the points corresponding to its vertices.
- These paths do not intersect each other, except for the trivial situation where two paths share a common endpoint.

It is not hard to see that this definition is equivalent to our earlier definition of planarity! Simply use the “stereographic projection map” (drawn at right) to translate any graph on the plane to a graph on the sphere, and vice-versa.

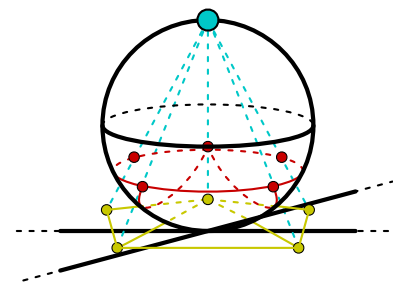
Definition. For any connected planar graph G , we can define a **face** of G to be a connected region of \mathbb{R} whose boundary is given by the edges of G .

For example, the graph at right has four faces.

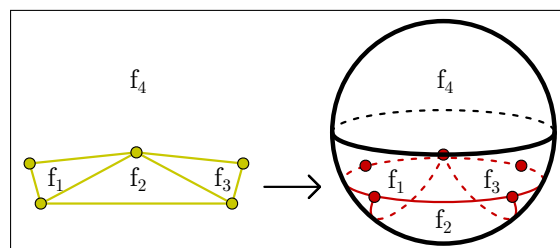
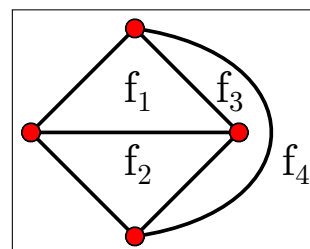
Notice that we always have the “outside” face in these drawings, which can be easy to forget about when drawing our graphs on the plane. This is one reason why I like to think about these graphs as drawn on the sphere; in this setting, there is no “outside” face, as all of the faces are equally natural to work with.

This observation has a nice accompanying lemma:

Lemma. Take any connected planar graph G , and any face F of G . Then G can be drawn on the plane in such a way that F is the outside face of G .



By drawing lines from the “north pole” $(0,0,1)$ through points either in the xy -plane or on the surface of the sphere, we can translate graphs drawn on the sphere (in red) to graphs drawn in the plane (in yellow.)



Proof. Take a planar embedding of G on the unit sphere. Rotate this “drawn-upon” sphere so that the face F contains the north pole $(0, 0, 1)$ of the sphere. Now, perform stereographic projection to create a planar embedding of G in \mathbb{R}^2 . By construction, the face F is now the outside face, which proves our claim. \square

It bears noting that not all graphs are planar:

Proposition. The graph $K_{3,3}$ is not planar.

Proof. We proceed by contradiction: let’s assume that there is some way to draw $K_{3,3}$ on the plane without any edges crossing. To get to a contradiction, first notice that $K_{3,3}$ contains a “hexagon,” i.e. a C_6 , as drawn at right. Therefore, in any drawing of $K_{3,3}$, we will have to draw this cycle C_6 .

Because this cycle is a closed loop, it separates space into an “inside” and an “outside.” Therefore, if we are creating a planar drawing after drawing this C_6 , any remaining edge will have to either be drawn entirely “inside” the C_6 or “outside” the C_6 ; that is, we can’t have an edge cross from inside to outside or vice-versa, because that would involve us crossing over pre-existing edges.

Therefore, if we have a planar drawing of $K_{3,3}$, after we draw the C_6 part of this graph, when we go to draw the $\{d, c\}$ edge we have two options: either draw this edge entirely on the inside of our C_6 , or entirely on the outside.

If we draw this edge on the inside, then on the inside the vertices f and a are separated by this edge; therefore, to draw the edge $\{a, f\}$ we must go around the outside. Similarly, if we draw this edge on the outside, then a, f are separated from each other on any outside path, and the edge $\{a, f\}$ must be drawn inside of the hexagon.

In either of these cases, notice that there is no path that can be drawn from b to e on either the inside or the outside! Therefore we cannot draw our last edge $\{b, e\}$ without breaking our planar condition, and thus have a contradiction to our claim that such a planar drawing of $K_{3,3}$ was possible. \square

If you are skeptical of the above proof, get a ping-pong ball or a grapefruit or something, and draw on it as described by the proof! In general, it’s a lot easier to understand most graph theory arguments if you’re drawing examples alongside the proofs at the same time.

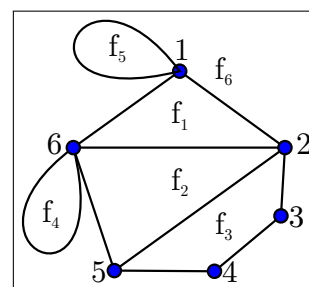
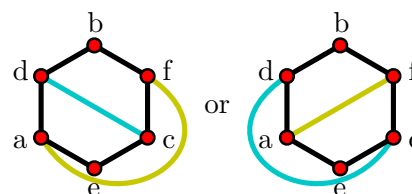
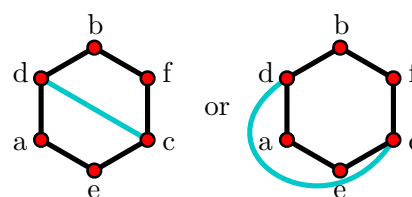
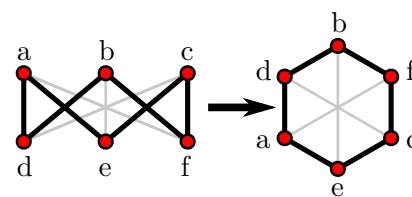
4.9 The Euler Characteristic

Planar graphs have many particularly beautiful properties! One of them is something called the **Euler characteristic**:

Theorem. (Euler characteristic.) Take any connected graph that has been drawn in \mathbb{R}^2 as a planar graph. Then, if V is the number of vertices, E is the number of edges, and F is the number of faces in this graph, we have the following relation:

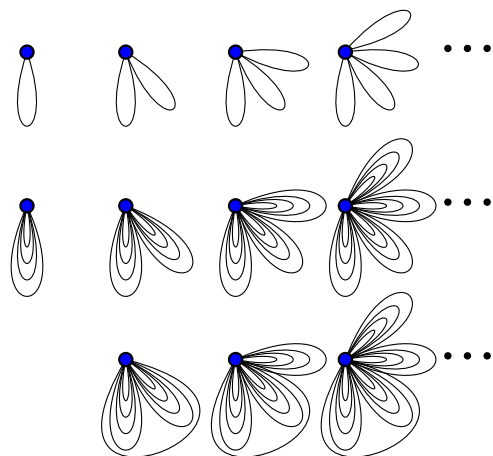
$$V - E + F = 2.$$

Proof. We will actually prove a stronger claim: we will show that any planar **multigraph** (a graph, but where we allow multiple edges between vertices, and also edges that start and end at the same vertex) satisfies the $V - E + F = 2$ formula. For the rest of this proof, we will assume that graph and multigraph are synonymous; once we are done with this proof, though, we will stop assuming this.



A planar multigraph with loops.

We proceed by induction on the number of vertices. Suppose that $V = 1$. Then our graph looks like something of the following form:



I claim that $V - E + F = 2$ for any of these graphs, and prove it by a second induction on the number of edges. For a zero-edge graph, this is easy; we have one vertex, no edges and one face, we have $V - E + F = 1 - 0 + 1 = 2$. Now, assume via induction that every one-vertex multigraph on n edges has $V - E + F = 2$. Take any graph on one vertex with $n + 1$ edges. Pick one of these edges, and look at it.

I claim that this edge borders exactly two faces. To see why, take any edge, and assign an orientation to it (i.e. if our edge is $\{x, y\}$, then orient the edge so that we travel from x to y .) If you do this, then our edge has two “sides,” the left- and right-hand sides, if we travel along it via this orientation.

There are two possibilities, as drawn above: either the left- and right-hand sides are different, or they are the same. This tells us that our edge either borders one or two faces! To see that we have exactly two, we now recall that our edge (because our graph has exactly one vertex) must start and end at the same vertex. In other words, it is a **closed** loop: i.e. its **outside is different from its inside**! In other words, our left- and right-hand sides are different, and our edge separates two distinct faces.

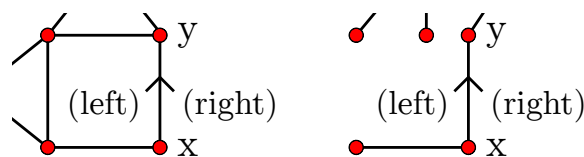
Therefore, deleting this edge does the following things to the graph: it decreases our edge count by 1, and also decreases our face count by 1 (as we merge two faces when we delete this edge.) In other words, deleting this edge does not change $V - E + F$! But by induction we know that $V - E + F = 2$ for all 1-vertex graphs on n edges, which is what we get if we delete this edge from a $n + 1$ -edge graph. So we’re done!

This settles our base case for our larger induction on V , the number of vertices. We now go to the second phase of an inductive proof: we show how to reduce larger cases to smaller cases!

To do this, we use **edge contraction**, a tool that was already shown to be quite handy for inductive arguments on graphs. As noticed before, contracting an edge decreases the number of vertices by 1 at each step, and also decreases the number of edges by 1 at each step. Finally, it never changes the number of faces; if two faces were distinct before this process happens, they stay distinct, as we’re not making any cuts in any of our boundaries (and instead are just shrinking them a bit!)

But this means that $V - E + F$ is still constant! Therefore, by induction, if $V - E + F$ holds for every n -vertex multigraph, it holds for any $n + 1$ -vertex multigraph by just contracting an edge. This finishes our induction, and thus our proof. \square

There’s a fairly beautiful consequence of this theorem, which is a nice note to end the first half of the course on: the classification



of all Platonic solids!

Definition. A **regular polygon** is one in which all of the side lengths are equal and all of the interior angles are equal. Equilateral triangles and squares are examples of regular polygons.

A **Platonic solid** is a 3-D shape whose faces are all copies of the same regular polygon, such that the same number of edges meet at each vertex.

Tetrahedra, cubes, octahedra, dodecahedra and icosahedra are all examples of Platonic solids that you've seen before. In particular, you've likely encountered several of these shapes as **dice**, as these are convenient shapes with (4/6/8/12/20) sides that if you were to "roll," you'd expect every side to have the same likelihood of being on top.

Perhaps surprisingly, these are the **only** Platonic solids: that is, there isn't some 3D-shape made out of heptagons, or out of 54 equilateral triangles, or anything else you might think of! If you're sticking together regular polygons, using the same number of polygons at each vertex, and not mixing-and-matching your polygons, this is it: these are the only five.

... why? Well: let's **prove** this!

Claim: There are only five Platonic solids: the tetrahedron, the cube, the octahedron, the dodecahedron, and the icosahedron.

Proof. Take any Platonic solid. As defined, we know that all of the faces of our Platonic solid are copies of the same regular polygon: let n denote the number of sides of that polygon. (For example, if our Platonic solid's faces were all triangles, n would be 3.) As well, we know that the same number of polygons meet at every vertex of our polyhedron: call that value k .

Finally, let V denote the number of vertices in our Platonic solid, E the number of edges, and F the number of faces.

What do we know about these values? Well:

- Take all of the faces of our Platonic solid, and on each face add up the number of edges that face contains. On one hand, we should get nF , because each face is a n -gon and contains n edges.

On the other, though, we should get $2E$! This is because every edge is counted in two faces; the face to the left of that edge, and the face to the right.

As a result, because these were just two ways of counting the same object, we know that these things must be equal: that is, that $nF = 2E$. Dividing by n gives us $F = \frac{2}{n}E$.

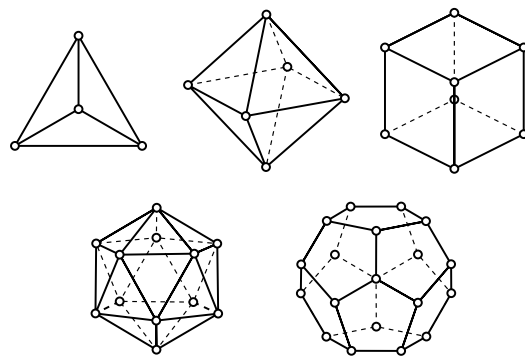
- Similarly, take all of the vertices of our Platonic solid, and for each vertex add up the number of edges leaving that vertex. On one hand, we know that this is kV , because each vertex has k edges leaving it.

On the other hand, though, this is $2E$ again! This is because (as we saw earlier) the sum of the degrees of vertices in any graph is always twice the number of edges.

As before, this tells us that these two quantities are equal; that is, that $kV = 2E$. Dividing by k gives us $V = \frac{2}{k}E$.

- Finally, we know that our Platonic solid, thought of as a graph, is **planar**! That is, if you were to draw your Platonic solid on a sphere (by "blowing it up" a bit so that it is a sphere, basically), none of its edges would cross.

This means that $V - E + F = 2$.



The five regular polyhedra.

If we combine all of these observations together, we get

$$\begin{aligned} V - E + F &= 2 \\ \Rightarrow \frac{2}{n}E - E + \frac{2}{k}E &= 2 \\ \Rightarrow \left(\frac{2}{n} + \frac{2}{k} - 1 \right) &= \frac{2}{E} \end{aligned}$$

Because $\frac{2}{E}$ is positive, this tells us that $\frac{2}{n} + \frac{2}{k} - 1$ is positive as well; that is, that $\frac{2}{n} + \frac{2}{k} > 1$!

However, there just aren't that many values of n, k that make this possible. Notice that if we have a 3D shape, we have to have at least three edges leaving each vertex (two would make us flat), and if our faces are polygons they need to each have at least three edges in them (as a 2-gon isn't a polygon!) In other words, $n, k \geq 3$.

But if $n = 3$, the only values of k that make $\frac{2}{n} + \frac{2}{k} > 1$ are when $k = 3, 4, 5$; if $k \geq 6$, this inequality does not hold. If $n = 4$, the only value of k that works is when $k = 3$; if $k \geq 4$ our inequality fails again. Finally, if $n = 5$ the only value of k that works is $k = 3$, and if $n \geq 6$ there are no values of $k \geq 3$ that work.

In other words, the only (n, k) pairs that work are $(3, 3)$, $(3, 4)$, $(3, 5)$, $(4, 3)$, $(5, 3)$; that is, the tetrahedron, octahedron, icosahedron, cube, and dodecahedron!

□

Chapter 5: Trees

Week 7

UoA 2018

A particularly useful kind of graph to study (especially in computer science) are **trees**! Trees are a remarkably applicable family of graphs, with uses ranging from chemistry to file systems to transit maps. In this section, we define the concepts we need to study trees, study notable applications of trees, and then prove a handful of fundamental results about trees.

5.1 Trees: Definitions and Examples

To define what a tree is, we introduce a useful concept from graph theory that we didn't have time to discuss last chapter:

Definition. A graph G has another graph H as a **subgraph** if H is “contained within” G . In other words, if you can take G and remove vertices and/or edges from it until you get the graph H , then H is a subgraph of G .

For example, the graph at right has C_5 as a subgraph, because we can delete the “inside” vertices and edges to have just a C_5 left over.

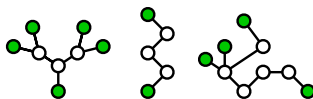
Note that any graph G is “trivially” a subgraph of itself, as we can just delete “nothing” from G and have G left over.

With this stated, we can define a tree as follows:

Definition. A **tree** is a graph T that is connected and has no cycle graph C_n as a subgraph.

For example, the three graphs at right are not trees, as each of them has a cycle graph of some length as a subgraph.

However, the three graphs below are all trees:



Happy little trees.

We call vertices of degree 1 in a tree the **leaves** of the tree. For example, the leaves of the trees above are colored green.

5.2 Trees: Some Fundamental Theorems

To help us understand how to work with trees, let's prove a handful of fundamental theorems about trees as graphs:

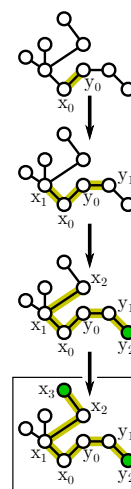
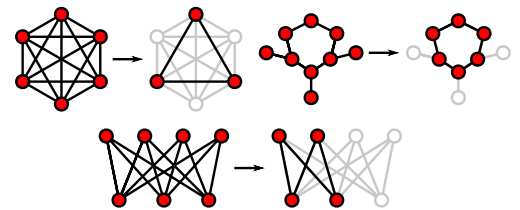
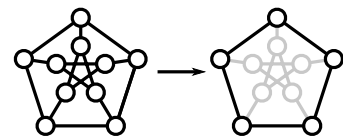
Theorem. If T is a tree containing at least one edge, then T has at least two leaves.

Proof. Consider the following process for generating a path in T :

Init: Choose any edge $e = \{x_0, y_0\}$ in G .

1. Starting from $i = 0$, repeatedly do the following: if x_i has degree ≥ 2 , then pick a new edge $\{x_i, x_{i+1}\}$ leaving x_i . Because T is a tree, x_{i+1} is not equal to any of our previously-chosen vertices (if it was, we'd have a cycle.) Stop when x_i eventually has degree 1.
2. Starting from $i = 0$, do the same thing for y_i .

This process must eventually stop: on a tree with n vertices, we can only put n vertices in our path because the “no cycle” property



stops us from repeating vertices. When it stops, the endpoints of the path generated are both leaves because this is the only way we stop this process. Therefore, this process eventually finds two leaves in any tree! \square

We use this result to prove a very useful property about trees:

Theorem. If T is a tree on n vertices, then T contains exactly $n - 1$ edges.

Proof. We proceed by induction. Our base case is straightforward: any tree on 1 vertex clearly has no edges (as it's a simple graph.) If you want, you can also consider 2-vertex graphs as well; the only connected two-vertex graph is $\bullet \text{---} \bullet$, which has one edge as desired.

For the inductive step, let's assume that our property holds for all trees on up to n vertices. Let T be any tree on $n + 1$ vertices; we want to use our assumption to prove that T contains exactly $(n + 1) - 1 = n$ edges.

To do this, let l be a leaf vertex in T (we know that l exists by our earlier theorem.) Delete l and the edge connecting l to the rest of T from T ; call the resulting graph $T - l$.

$T - l$ contains n vertices, because we started with $n + 1$ vertices and deleted one vertex. It is also still connected (because l was degree 1, the only walk that would need to use the edge to l is a walk going directly to l , and we deleted l .) Finally $T - l$ contains no cycle subgraphs, because T contained no cycle subgraphs and deleting things from T cannot have somehow caused a cycle to exist.

Therefore $T - l$ is a tree! By induction, $T - l$ contains $n - 1$ edges.

Therefore T itself contains $(n - 1) + 1 = n$ edges, because T is just $T - l$ plus the vertex l and the single edge connecting l to the rest of T . In other words, we've proven our inductive claim! \square

The above property (that any tree on n vertices contains exactly $n - 1$ edges) actually turns out to **completely characterize** trees: that is, if you take any connected graph on n vertices, if it has $n - 1$ edges, then it is forced to be a tree! We prove this here:

Theorem. If G is a connected graph on n vertices containing exactly $n - 1$ edges, then G is a tree.

Proof. We proceed by contradiction; suppose that G is a connected graph on n vertices containing $n - 1$ edges that is somehow not a tree.

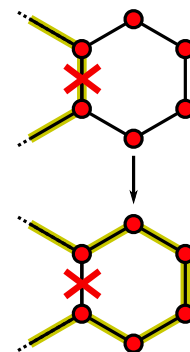
Because G is connected, the only way that G can fail to be a tree is if it contains a cycle subgraph. Let $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_k, v_1\}$ be such a cycle subgraph.

Take G and delete the edge $\{v_1, v_2\}$ from G . I claim that G is still connected.

To see why, take any walk in G that uses the edge $\{v_1, v_2\}$, and replace each use of $\{v_1, v_2\}$ with the sequence of edges $\{v_1, v_k\}, \{v_k, v_{k-1}\}, \dots, \{v_3, v_2\}$. In other words, every time you'd go directly from v_1 to v_2 along that edge, instead use the cycle to go the "other" way around!

As a result, if two vertices x, y used to be connected by a walk in G , they are still connected after deleting $\{v_1, v_2\}$; in other words, $G - \{v_1, v_2\}$ is still connected.

But $G - \{v_1, v_2\}$ is a graph on n vertices containing $n - 2$ edges, as we had $n - 1$ edges and deleted one. But in chapter 4, we proved that a connected graph on n vertices must contain at least $n - 1$ edges! In other words, we have a **contradiction**, and so our claim that G was a tree must have been correct. \square



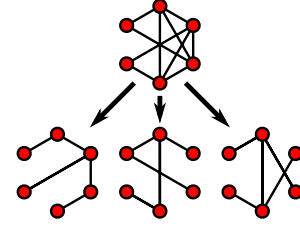
5.3 Spanning Trees

In this section, we study a particularly useful application of the concept of trees: namely, **spanning trees**.

Definition. A **spanning tree** T of a graph G is a subgraph of G that contains every vertex in G and is a tree.

We draw several examples of spanning trees at right.

Spanning trees are remarkably useful objects in computer science and mathematics! One of the particularly useful properties of spanning trees is the following:

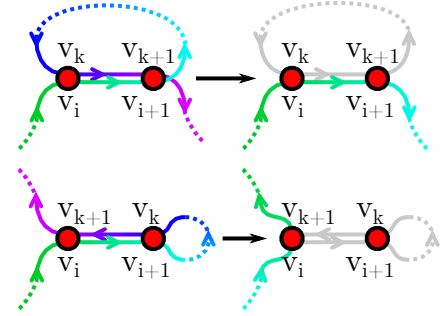


Theorem. If T is a tree, then there is exactly one path²⁶ between any two vertices in T . In particular, if T is a spanning tree of a graph G , then there is exactly one path in T between any two vertices of G .

Proof. Because T is a tree, T is connected; as a result we know that for any two vertices x, y there is at least one walk that goes from x to y .

To turn any such walk into a path, do the following:

- Take any walk $\{v_0, v_1\}, \{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}$.
- If this walk repeats an edge e , let $\{v_i, v_{i+1}\}$ be the first time e is used, and $\{v_k, v_{k+1}\}$ be the last time e is used.
- There are two possibilities: either $v_i = v_k, v_{i+1} = v_{k+1}$ (in which case we walked on e in the same direction both times) or $v_i = v_{k+1}, v_{i+1} = v_k$ (in which case the second time we walked on e , we did so “backwards.”)
- In the first possibility, simply cut out everything in between the first and last use of this edge, as well as the last use of this edge. This leaves the walk $\{v_0, v_1\}, \dots, \{v_{i-1}, v_i\}, \{v_i, v_{i+1}\}, \{v_{k+1}, v_{k+2}\}, \{v_{k+2}, v_{k+3}\}, \dots, \{v_{n-1}, v_n\}$. This is still a walk because $v_{i+1} = v_{k+1}$.
- In the second possibility, cut out everything between the first and last use of this edge, as well as both times the edge is used. This leaves the walk $\{v_0, v_1\}, \dots, \{v_{i-1}, v_i\}, \{v_{k+1}, v_{k+2}\}, \{v_{k+2}, v_{k+3}\}, \dots, \{v_{n-1}, v_n\}$. This is still a walk because $v_i = v_{k+1}$.
- Repeat this process until it cannot be repeated any more. The result remains a walk, as shown: it also still has the same start and end points, as this process does not change those!



This completes half of our proof: we’ve shown that there is *at least* one path between any two vertices. To see why two distinct paths is impossible, we proceed by contradiction.

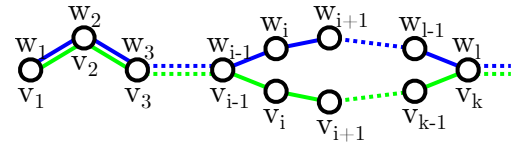
Suppose that there are vertices x, y linked by two different paths $P_1 = \{v_1 = x, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, y\}$ and $P_2 = \{w_1 = x, w_2\}, \{w_2, w_3\}, \dots, \{w_{m-1}, y\}$.

Because these two paths are different, there must be some value i such that $v_i \neq w_i$. Let i be the smallest such value, so that these paths agree at $v_i = w_i$ and diverge immediately afterwards.

These two paths must eventually meet back up, as they end at the same vertex y . Let k, l be the two smallest values greater than i such that $v_k = w_l$. Notice that this means that all of the vertices $v_{i-1}, v_i, v_{i+1}, \dots, v_{k-1}, v_k, w_{l-1}, w_{l-2}, \dots, w_i$ are distinct (as otherwise we could have picked even smaller values at which these paths met back up.)

Now, look at the walk formed by starting P_1 at v_{i-1} , proceeding until v_k , and then taking P_2 backwards from w_l until w_{i-1} . This walk repeats no vertices other than the starting and ending one, by construction. Therefore it is a cycle!

But we are in a tree, and trees do not contain cycles. Therefore this is a *contradiction* to our assumption that we had two distinct paths.



□

The reason we care about this property is because graphs are often used to model **connected networks**. To give a few examples:

- On a hardware level, you can model the Internet (and indeed the local network at any business / University / etc) as a graph. In this setting, you have lots of different routers / computers / etc all linked to each other; you can think of the devices as our **vertices**, and the connections (either via WiFi or via physical cables) as our **edges**.
- You can also model a city as a graph: make each neighborhood or point of interest (e.g. UoA, or Britomart, or the Domain) a vertex, and connect two points by an edge if they're linked by a direct road. (The same idea applies to cities and direct airport flights, or classrooms on campus and footpaths.)

In both of these situations, you often do not want to deal with the **entire graph** all at once!

- In the “internet” graph, you'll often have routers and devices that want to broadcast a message to everyone else. To do this, they'll send out a message to devices they're connected to, and ask them to “rebroadcast” this to their neighbors.

However, if you have a cycle in your network graph (i.e. device A is connected to device B, who is connected to device C, who's connected back to A) this process may send you into an **infinite loop**, in which all of the looped devices just keep rebroadcasting your message around and around in a loop. That's bad!

To fix this, routers and other internet devices often construct the graph of everything they're connected to, and trim it down to a **spanning tree**. They then use this tree to communicate, as doing so means that we do not have any cycles and thus avoid the infinite loop problem from earlier!

- If you're designing a train line for a large city, you'll likely want to connect as many major areas as possible with a minimum of railroad track (to keep costs down.) To do this, you'll often want to take your city graph and construct a **spanning tree**, and then build rail on the spanning tree! Because a tree has the smallest number of edges amongst any connected graph, this should keep costs down while still connecting all of your major areas (because the tree is spanning.)

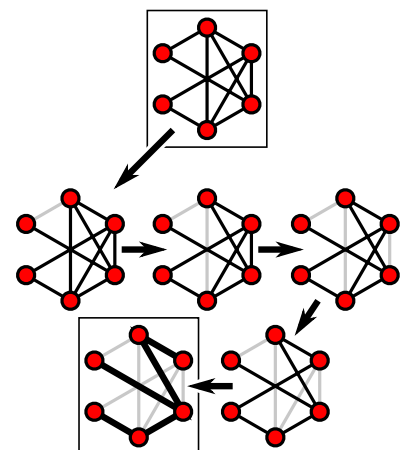
Luckily for us, spanning trees are easy to find:

Theorem. If G is a connected graph, then G has a spanning tree.

Proof. We give a process for finding a spanning tree of any connected graph G here:

- If G is not a tree, then as shown before G must contain a cycle subgraph. Pick any such cycle subgraph and delete an edge from that cycle; as shown before, this does not change the fact that G is connected.
- Repeatedly do this until G no longer contains any cycle subgraphs.

The resulting graph is a tree, as it's still connected and contains no cycles. In particular, because we never deleted any vertices from G , this is a **spanning tree**, as desired. □



This is great for the internet problem we mentioned earlier! However, it can leave something to be desired in other applications. Consider the transit problem, where we wanted to construct a rail network for a large city.

While it is true that a spanning tree will mean we're minimizing the number of connections that we're drawing, some connections (e.g. train lines that require us to make a tunnel, or a bridge) are much more expensive than others. As such, we'll often not just want to make a spanning tree, but a spanning tree with the "smallest possible cost" amongst all possible spanning trees!

To make this rigorous, we have a pair of definitions:

Definition. Given a graph G , we can turn G into a **edge-weighted** graph by labeling each of its edges with real numbers.

For example, we could take a map of the South Island (subdivided into regions corresponding to rugby teams) and turn this into a graph: our regions are the vertices, and we connect two regions when they're adjacent.

We can label each of these connections with estimated travel times, that give us an idea of how long it would take to travel from one region to the next. Doing this gives us a **edge-weighted** graph!

Given an edge-weighted graph G and a spanning tree T of G , we say that the **weight** of T is the sum of the weights of all of the edges in T . We say that T is a **minimum spanning tree** for G if the weight of T is the smallest possible amongst all spanning trees of G ; that is, if $\text{weight}(T) \leq \text{weight}(T')$, for any other spanning tree T' .

Notice that a graph G may have more than one minimum spanning tree; ties for the smallest possible weight are entirely possible.

Finding minimum spanning trees is (for the reasons stated above) an incredibly practical task in computer science! There are several algorithms that construct minimum spanning trees, each with their own pros and cons; in other CompSci papers you will study several of these and their associated run-time/pros/cons.

For now, I want to pick just one famous one (specifically, Prim's algorithm) and **prove** that it finds a minimum spanning tree. We do this to both practice our proof techniques, and to illustrate how we can use our algorithm techniques to study and prove the correctness of famous results in computer science!

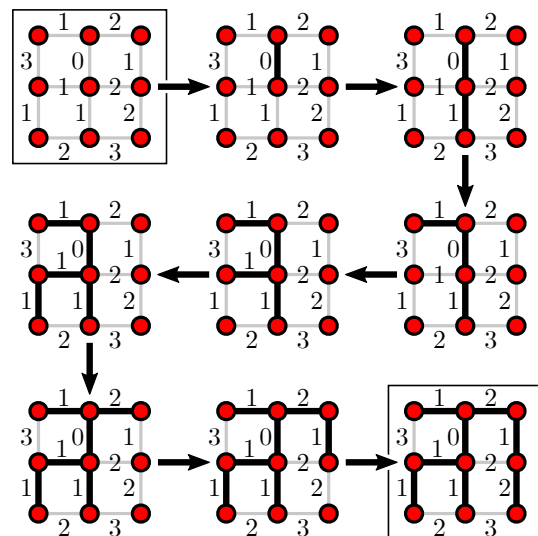
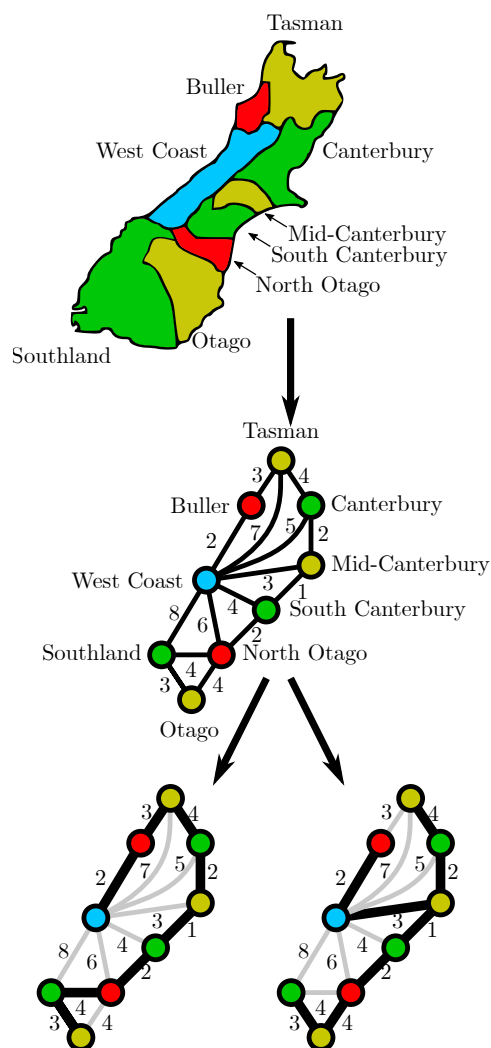
Theorem. (Prim:) Take any connected graph G with edge weights. Consider the following process:

- Init: Let e be an edge with the smallest possible weight in G . (If there is a tie, pick one such edge at random.) Initialize T to be the edge e and its two endpoints.
1. If T is a spanning tree, stop. Otherwise, look at the collection of all edges with exactly one endpoint in T . Pick the one with the smallest possible weight and add it to T .

This generates a minimal spanning tree.

Proof. To see why this process generates a spanning tree, notice the following:

- In the process above, note that if T is not a spanning tree there will always be edges to choose from. This is because if T is not spanning, then by definition there are vertices in G that are not in T . Because G is connected, there are walks from these vertices to the vertices in T , and thus in particular there are edges with exactly one vertex in T and



one not in T . (Specifically, we have at the least the first edge on our walk where we go from a non- T vertex to a T vertex!)

- T is also connected, because we only add edges connected to T at each step.
- As well, each step of this process increases the number of edges and vertices in T by 1 (because exactly one endpoint of e was in T .) Because T starts with 2 vertices and 1 edge, this means that at each step T will always have one more vertex than it has edges.
- Therefore, our results from before tell us that T must be a tree; this is because T is connected and has one less edge than it has vertices!

To see that Prim generates a **minimal** spanning tree takes a bit more work:

- Take a minimal spanning tree S of G .
- At the same time, run Prim's algorithm on G to try to make a spanning tree T , edge-by-edge. Stop this algorithm if it ever picks an edge that's not in S .
- If we never stop, then at the end $S = T$ and we're done, as S was minimal!
- Otherwise, at some point in time Prim picked an edge $e_1 = \{x, y\}$ to add to T that is not in S . Let x be the vertex that was already in T and y be the new vertex to T .
- Because S is spanning and a tree, there's a unique path in S from x to y ; because $e_1 \notin S$, this path plus e_1 forms a cycle. Call it C .
- On this cycle, start from y and travel away from x around the cycle until you first return to T on some other edge e_2 . (You'll eventually do this, because you started at a vertex y not in T and will eventually get to a vertex x that's in T .)
- This edge e_2 was a choice we had for T , but didn't make. As a result its weight is no less than the weight of e_1 .
- As well, if we add e_1 to S and delete e_2 the resulting graph is still a tree. This is because it's still connected (any path that used e_2 can just go the other way around the cycle C) and we didn't change the number of vertices or edges.
- Therefore, because S was minimal and we traded e_2 for an edge of no greater weight, S is still minimal after this change. Also, S now agrees with our choice of e_1 !
- Do this **every time** S and T disagree, until they no longer disagree. At this point in time S and T will now be equal, which proves that T was indeed a minimal spanning tree!

□

Chapter 6: Sets, Relations and Functions

Weeks 7-9

UoA 2018

In the preceding pages of this coursebook, we've used sets to describe and work with many mathematically useful objects! Motivated by this, we're starting this chapter off by studying sets *formally*: here, we will define many of the set-theoretic ideas we've been using throughout this course, prove a few set properties to illustrate how these sorts of arguments work, and proceed from there to introduce some new ways of combining and creating sets that will come in handy later.

From there, we'll use these ideas to study the mathematical concepts of **equivalence relations**, **posets**, and **functions**, a trio of concepts that are ubiquitous in computer science and mathematics! We'll study several examples of each of these here, and write some straightforward proofs involving each concept to help us get a grasp on them.

6.1 Sets

Our first definition is pretty straightforward:

Definition. A **set** is just a collection²⁷ of things. We typically define a set by listing or describing the objects it contains, separating objects by commas and surrounding the entire collection with a pair of curly braces $\{$ and $\}$. We call the things inside a set the **elements** of the set, and use the symbol \in to say that an element is in a set.

For example, $\{1, 2, 3\}$ is a set containing the three elements 1, 2 and 3. Similarly, $\{\text{Bulbasaur}, \text{Charmander}, \text{Squirtle}\}$ is a set, containing the three starter Pokémon from the original games! Sets can contain all sorts of things, and are not restricted to just “numbers” or “mathematical objects.” For these sets, we would say that $2 \in \{1, 2, 3\}$, but $\pi \notin \{\text{Bulbasaur}, \text{Charmander}, \text{Squirtle}\}$.

In a set, we do not care about the ordering of the elements in a set: that is, we think that $\{1, 2, 3\}$, $\{2, 3, 1\}$ and $\{3, 2, 1\}$ are all the same set! We also do not list elements multiple times, and ignore repeated elements if this were to happen: that is, we would regard the sets $\{1, 1, 2, 3, 3, 3, 2, 1\}$ and $\{3, 3, 2, 1\}$ as both describing the “same” set $\{1, 2, 3\}$, though we would never write out $\{1, 2, 3\}$ in such a form because it's weird/misleading.

We say that two sets A, B are equal precisely when they contain the exact same elements, and say that they are not equal if either of A, B contains an element not present in the other.

To describe a set with infinitely many elements (in which case it is impractical to list all of the elements,) we can give a **rule** that defines all of the elements in the set. For example,

$$\{x \mid x = 2k + 1, k \in \mathbb{Z}\}$$

means “The set of all values of x such that $x = 2k + 1$, where k is an integer;” in other words, this defines the set of all odd

²⁷This is 99% the true and correct definition for what a set is! However, in the event that you let your sets start using recursive definitions to define their “collection,” it is possible that you'll encounter some fun paradoxes, the resolution of which requires some remarkably powerful mathematics! See the [Zermelo-Fraenkel set theory axioms](#) for the full-on formal definition of what counts as a set. In the meantime, if you'd like a paradox to help motivate *why* this naive definition might not be enough, consider the following (very strange!) set:

$$A = \{B \mid B \text{ is any set such that } B \notin B\}$$

Is $A \in A$? Or is $A \notin A$?

numbers! Similarly, one could define such a set by describing the **form** of all of its elements: i.e.

$$\{2k + 1 \mid k \in \mathbb{Z}\}$$

is another way we could define the set of all odd integers.

Finally, one can sometimes define a set by listing its first few elements and hoping that someone sees the pattern. For instance, one could define the odd numbers by writing

$$\{\dots - 5, -3, -1, 1, 3, 5, \dots\}$$

It's worth noting that this can backfire sometimes: for instance, if you were trying to describe the set of all integers of the form $n^6 - 35n^4 + 259n^2 + n - 225$, you would **also** write down the numbers $-5, -3, -1, 1, 3, 5$ (but definitely not ± 7 , or indeed many other odd numbers!) Only use this if you think your pattern is very clear and obvious.

6.1.1 Commonly Used Sets and Operations

Here are a handful of frequently used sets:

- \mathbb{N} denotes the set of all **natural**²⁸ **numbers**, that is all nonnegative whole numbers. In symbols, we can write $\mathbb{N} = \{0, 1, 2, 3, \dots\}$
- \mathbb{Z} denotes the set of all **integers**; in other words, all whole numbers. In symbols, we could write $\mathbb{Z} = \{\dots - 2, -1, 0, 1, 2, \dots\}$
- \mathbb{Q} denotes the set of all rational numbers; i.e. all numbers that we can express as a ratio of whole numbers. In symbols, we can write $\mathbb{Q} = \{\frac{m}{n} \mid m, n \in \mathbb{Z}, n \neq 0\}$.
- \mathbb{R} denotes the set of all real numbers; that is, all numbers that we can express via a (possibly infinite) decimal expansion. For example, things like π and $\sqrt{2}$ are real numbers.
- The empty set, denoted by the symbol \emptyset , is the set containing nothing: that is, $\emptyset = \{\}$.

Given a pair of sets A, B , there are several ways in which we can “combine” A and B into a new set:

- We define the **intersection** of A and B , denoted by $A \cap B$, be the set of all elements in A that are also in B . For example, $\{1, 2, 3\} \cap \{2, \text{a sandwich}, \pi\} = \{2\}$.

We think of the intersection of two sets as listing all of the elements that both sets have “in common.” Formally, we define $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$.

- We define the **union** of A and B , denoted by $A \cup B$, be the set of all elements in either A or B . For example, $\{1, 2\} \cup \{2, \pi\} = \{1, 2, \pi\}$.

You can think of this as “merging” the two sets A, B together into one set that contains anything that either original set had. Formally, we define $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.

- We define the **set difference** of A and B , denoted by $A \setminus B$, be the set of all elements in A that are not in B . For example, $\{1, 2, 3\} \setminus \{1, 2\} = \{3\}$, $\{1, 2, 3\} \setminus \{4, \text{a sandwich}\} = \{1, 2, 3\}$, and $\{1, 2\} \setminus \{2, \pi\} = \{1\}$. Notice in particular that B can contain elements that aren't in A : if it does, we just

²⁸People in the fields of computer science, combinatorics, logic, and a few other fields will also sometimes say that 0 is a natural number. This is one of those weird inconsistencies that crop up in a language when enough people use it, like how Americans don't use ‘u’ when spelling ‘colour’.

ignore those elements, and only remove the elements in B that *are* also in A from A to get the resulting set $A \setminus B$.

Formally, we define $A \setminus B = \{x \mid x \in A \text{ and } x \notin B\}$.

If you want to perform multiple operations on a collection of sets, there is no presumed “PEMDAS/BEDMAS” order-of-operations; instead, simply group operations using parentheses so that the order is unambiguous. For example:

$$(\{1, 2\} \cup \{2, 3\}) \cap (\{1, 2, 3, 4\} \setminus \{2, 3\}) = \{1, 2, 3\} \cap \{1, 4\} = \{1\}.$$

When we defined sets earlier, we said that our sets were “un-ordered:” that is, that $\{1, 2\}$ and $\{2, 1\}$ shouldn’t be regarded as different sets. Sometimes it is useful to refer to sets in which the elements occur in a fixed order, though: to do this, we use the following notation.

Definition. An **ordered set** is a list of objects, separated by commas and surrounded by parentheses (and). For example, $(1, 2, 3)$ is the ordered set of the three numbers 1, 2, 3 in that order, while $(2, 3, 1)$ is a different ordered set containing the same elements.

You’ve likely seen ordered sets when working with mathematical objects like $\mathbb{R}^2 = \{(x, y) \mid x, y \in \mathbb{R}\}$, i.e. the Euclidean plane!

This notion of an “ordered set” lets us define a new way to combine two sets: the **Cartesian product**!

Definition. Take any two sets A, B . The **Cartesian product** $A \times B$ is the set of all ordered pairs of elements where the first coördinate comes from A and the second from B : that is, $A \times B = \{(a, b) \mid a \in A, b \in B\}$.

For example, if we let $A = \{1, 2\}$ and $B = \{1, 3, 4\}$, we would have

$$A \times B = \{(1, 1), (1, 3), (1, 4), (2, 1), (2, 3), (2, 4)\}.$$

You can of course generalize this to products of more than two sets; i.e. you could define $A \times B \times C = \{(a, b, c) \mid a \in A, b \in B, c \in C\}$.

There is one last commonly used way to construct a new set from a pre-existing one: the **power set**! To define it, we first define the concept of a **subset**:

Definition. Given any two sets A, B , we say that B is a **subset** of A , and write $B \subseteq A$, if every element of B is also an element of A . For example, $\{1, 2, 3\} \subseteq \mathbb{Z}$, $\mathbb{Q} \subseteq \mathbb{R}$, and $\emptyset \subseteq \emptyset$.

Notice that under this definition every set is a subset of itself; if you want your reader to not count this possibility, you would say that B is a **proper subset** of A , and write $B \subsetneq A$.

Definition. Given a set A , we define the **power set** of A , denoted $\mathcal{P}(A)$, as the collection of all subsets of A . In other words, $\mathcal{P}(A) = \{B \mid B \subseteq A\}$.

For example,

$$\mathcal{P}(\{1, 2, 3\}) = \left\{ \emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\} \right\}$$

Notice that in particular a set can contain **other sets**! This can lead to some fairly surprising results if you don’t think carefully about your definitions.

For example, I claim that $1 \notin \mathcal{P}(\{1, 2, 3\})$. This might seem somewhat counterintuitive: after all, when we wrote $\mathcal{P}(\{1, 2, 3\})$ above,

we said that it contains $\{1\}$, and surely that means that 1's an element of this set, right?

Not quite. You see, the objects $\{1\}$ and 1 are different things: 1 is a number (and thus is something we know how to multiply and add and divide and...), while $\{1\}$ is a *set*: that is, it's something we don't know how to add or multiply or divide by, but it **is** something we can take unions and intersections and other such things with!

From a computer science perspective, this is like the difference between 1 thought of as a number and 1 thought of as an ASCII character; while they look the same, they are stored as very different objects and interact quite differently when you try to perform certain operations! If computer science isn't your cup of tea, though, here's a more natural-language explanation for the difference: think of 1 as an object, and as the set $\{1\}$ as "the object 1 wrapped up in some packaging." In this sense, these are very different things: the "packaging" of the set notation means that we can do set things to this object, but if we want to perform arithmetic we need to "remove" the packaging first.

6.1.2 Sets: Two Example Proofs

To give us some practice with set theory arguments, we prove a few

Claim. If A, B are two sets, then $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$.

Proof. We proceed by expanding our definitions:

- On the left-hand-side, we have $A \setminus (B \cup C)$. By definition, $A \setminus (B \cup C) = \{x \mid x \in A \text{ and } x \notin (B \cup C)\}$; as well, by definition we have, $B \cup C = \{x \mid x \in B \text{ or } x \in C\}$.

By plugging this second definition into our first definition, we have that $A \setminus (B \cup C) = \{x \mid x \in A \text{ and } \neg(x \in B \text{ or } x \in C)\}$. In our first chapter, we saw how to negate statements, and in particular that $\neg(p \vee q) = \neg p \wedge \neg q$; as a result, we can simplify this to $\{x \mid x \in A \text{ and } x \notin B \text{ and } x \notin C\}$.

- On the right-hand-side, we can similarly use our definitions to notice that $A \setminus B = \{x \mid x \in A \text{ and } x \notin B\}$ and $A \setminus C = \{x \mid x \in A \text{ and } x \notin C\}$.

By definition, then, the intersection $(A \setminus B) \cap (A \setminus C)$ is just $\{x \mid (x \in A \text{ and } x \notin B) \wedge (x \in A \text{ and } x \notin C)\}$. Logically, we can simplify this down to $\{x \mid x \in A \text{ and } x \notin B \text{ and } x \notin C\}$, which is the same expression that the left-hand-side reduced to. So these two sets are indeed equal!

□

Claim. If A is a finite²⁹ set, then $\mathcal{P}(A)$ will always contain more elements than A .

Proof. Take any element $a \in A$, and notice that by definition $\{a\} \subseteq A$. Therefore $\mathcal{P}(A)$ has one element $\{a\}$ for every element $a \in A$. Also notice that for any $a \neq b$, the sets $\{a\}, \{b\}$ are distinct because they contain different elements; therefore $\mathcal{P}(A)$ has at least as many elements as A itself does.

Also notice that for any set A , $\emptyset \subseteq A$, and so $\emptyset \in \mathcal{P}(A)$! This is by definition: recall that $X \subseteq A$ holds if and only if " $\forall x$, if $x \in X$

²⁹The above claim holds even if A is not a finite set! To make sense of this, though, we'll need a rigorous notion for what it would mean for two infinite sets to have "different sizes." When we get to functions, we'll talk about how to do this!

then $x \in A$ is a true statement. If $X = \emptyset$, then X contains no elements, and so the “if” part of this claim must always be false; therefore the entire compound if-then statement is vacuously true (as discussed earlier in our propositional logic section!)

Because $\emptyset \neq \{a\}$ for any $a \in A$ (as it contains no elements, while $\{a\}$ always contains one element a), this means that we’ve found another element in $\mathcal{P}(A)$.

Therefore $\mathcal{P}(A)$ has strictly more elements than A does; it has one element $\{a\}$ for every $a \in A$, and also has an additional element \emptyset on top of all of the others. \square

6.2 Relations

With the above set-theoretic language in place, we can now define the idea of a **relation**:

Definition. A **relation** on a set A is any subset T of $A \times A$. Perhaps more instructively, we can think of a **relation** as any way to assign every ordered pair of elements in A to either the values “true” or “false.”

To see why these two definitions of a relation are equivalent, simply think of the ordered pairs in T as being the pairs we think our relation “holds” for (i.e. that we assign to “true,”) and all of the others as being the ones we think our relation fails on (i.e. that we assign to “false.”)

This definition is pretty abstract, so we make it concrete with the following list of examples:

- **Equality ($=$):** on any set S , you can define a relation “ $=$ ” by saying that $x = y$ is true if x and y are the same object, and false if x and y are different objects. In other words, the equals sign you’ve been using your whole life is a relation!
- **Mod n :** “ $\equiv \text{ mod } n$ ” is a relation on the set of integers \mathbb{Z} , defined as follows: we say that $x \equiv y \text{ mod } n$ is true whenever $x - y$ is a multiple of n , and say that it is false otherwise.
- **Less than:** the symbol “ $<$ ” is a relation on the real numbers \mathbb{R} . We say that $x < y$ is true whenever x is a smaller number than y (i.e. when $y - x$ is positive,) and say that it is false otherwise.
- **Factor:** On the set of positive integers, “is a factor of” , i.e. the symbol $|$, is a relation! For any two positive integers d, n , we say that $d|n$ holds if d is a factor of n , and say that $d|n$ fails if d is not a factor of n .
- **Beats:** this is a relation on the set $\{\text{rock, paper, scissors}\}$ in the game Rock-Paper-Scissors! We can define it by saying that the three statements “Rock beats scissors,” “Scissors beats paper,” and “Paper beats rock” are all true, and that all other pairings of these symbols are false.

There are some properties of relations that make them particularly nice to work with:

- **Reflexivity.** We say that a relation R on a set S is **reflexive** if the following holds: for any $x \in S$, xRx is true.
- **Symmetry.** We say that a relation R on a set S is **symmetric** if the following holds: for any $x, y \in S$, if xRy holds, then yRx is also true.
- **Transitivity.** We say that a relation R on a set S is **transitive** if the following holds: for any $x, y, z \in S$, if xRy and yRz both hold, then xRz is also true.

To get a handle on these properties, let’s check out some of the examples we gave above and see what properties they satisfy:

- Equality ($=$), on any set S , satisfies all three properties:
 - **Reflexivity**: for any $x \in S$, we have that $x = x$, by the definition of equality.
 - **Symmetry**: for any $x, y \in S$, if $s = t$, then x and y are the same; so $y = x$ also holds.
 - **Transitivity**: for any $x, y, z \in S$, if $x = y$ and $y = z$, then x, y, z are all the same; as a consequence, $x = z$ must hold.
- “Mod n ” ($\equiv \pmod{n}$) satisfies all three of these properties, as well:
 - **Reflexivity**: for any $x \in \mathbb{Z}$, $x - x = 0$ is a multiple of n ; therefore $x \equiv x \pmod{n}$.
 - **Symmetry**: for any $x, y \in S$, if $x \equiv y \pmod{n}$, then $x - y$ is a multiple of n ; consequently $y - x$ is also a multiple of n , and thus $y \equiv x \pmod{n}$.
 - **Transitivity**: for any $x, y, z \in S$, if $x \equiv y \pmod{n}$ and $y \equiv z \pmod{n}$, then $x - y$, $y - z$ are all multiples of n ; therefore $(x - y) + (y - z) = x - y + y - z = x - z$ is also a multiple of n , and thus $x \equiv z \pmod{n}$.
- “Less than” ($<$), however, does not satisfy all of these properties:
 - **Reflexivity**: it is not true that for any $x \in \mathbb{R}$, that $x < x$. To give a counterexample: if we let $x = 0$, we can see that $0 \not< 0$. So this property fails to hold for all $x \in \mathbb{R}$.
(Indeed, it fails to hold for *any* $x \in \mathbb{R}$, but we don’t need to show this to break the property: to disprove a “for all” statement, we just need to find **one** example!)
 - **Symmetry**: it is also not true that for any $x, y \in \mathbb{R}$, if $x < y$, then $y < x$. For example, it is true that $1 < 2$, and yet $2 \not< 1$.
 - **Transitivity**: This one is true! For any $x, y, z \in \mathbb{R}$ such that $x < y$ and $y < z$, we can see that z is the largest of the three, and so conclude that $x < z$.
- “Beats” fails to satisfy any of these properties:
 - **Reflexivity**: it is not true that for any $x \in \{\text{paper, scissors, rock}\}$ that “ x beats x ” holds. For instance, “paper beats paper” is false.
 - **Symmetry**: it is also not true that for any $x, y \in \{\text{paper, scissors, rock}\}$, if “ x beats y ,” then “ y beats x .” For example, while “paper beats rock” holds, “rock beats paper” does not hold.
 - **Transitivity**: it is also not true that for any $x, y, z \in \{\text{paper, scissors, rock}\}$ such that “ x beats y ” and “ y beats z ,” that “ x beats z ” must follow. For instance, while “paper beats rock” and “rock beats scissors,” it is not true that “paper beats scissors!”

6.2.1 Equivalence Relations

Definition. We call any relation that is both reflexive, symmetric and transitive an **equivalence relation**.

Given any set S with an equivalence relation R and an element $x \in S$, define the **equivalence class** corresponding to x , denoted by $[x]$, as the set $\{s \in S \mid sRx\}$.

We have worked with equivalence classes before! Let’s consider mod 3 arithmetic on the integers, as an example. If we look at the set $[0]$, then, by definition, this is the set of all numbers that

are congruent to 0 modulo 3; that is,

$$\begin{aligned}[0] &= \{s \in \mathbb{Z} \mid s \equiv 0 \pmod{3}\} \\ &= \{\dots - 6, -3, 0, 3, 6, \dots\}\end{aligned}$$

We can calculate $[1]$ and $[2]$ in the same way:

$$\begin{aligned}[1] &= \{s \in \mathbb{Z} \mid s \equiv 1 \pmod{3}\} = \{\dots - 5, -2, 1, 4, 7, \dots\}, \\ [2] &= \{s \in \mathbb{Z} \mid s \equiv 2 \pmod{3}\} = \{\dots - 4, -1, 2, 5, 8, \dots\}\end{aligned}$$

Notice that every integer in \mathbb{Z} is in exactly one of these three sets! This isn't an accident, as the following theorem states:

Theorem. Take any set S with an equivalence relation R , and any $x, y \in S$. Then the following statements are equivalent:

1. xRy	2. $[x] = [y]$.	3. $[x] \cap [y] \neq \emptyset$
----------	------------------	----------------------------------

Proof. To prove that three things A, B, C are equivalent, we usually do something like prove $A \Rightarrow B, B \Rightarrow C$ and $C \Rightarrow A$; that way, if any of A, B, C are true, we'll get that all of them are true (and in particular that there's never a case where they have different truth values, which is what equivalence means!)

Take any set S with an equivalence relation R , and any two elements $x, y \in S$. Let's write a proof like the above here!

- (1) \Rightarrow (2) : We assume that xRy , and seek to show that $[x] = [y]$ (by showing every member of $[x]$ is in $[y]$, and vice-versa.)
We know that $[x] = \{s \in S \mid sRx\}$. For any $s \in [x]$, because sRx , we can use transitivity plus the fact that xRy to conclude that sRy ; as a result, we have $s \in [y] = \{t \in S \mid tRy\}$. Similarly, we can use symmetry to note that because xRy , we have yRx . As a result, for any $t \in [y]$, because tRy and yRx , we have by transitivity that tRx , and so $t \in [x]$.
- (2) \Rightarrow (3) : We assume that $[x] = [y]$, and seek to show that $[x] \cap [y] \neq \emptyset$. This is easy! If $[x] = [y]$, then their intersection $[x] \cap [y]$ is just the original set $[x]$. This always contains at least the element x , because reflexivity says that for any $x \in S$, xRx ; so this set is nonempty.
- (3) \Rightarrow (1) : We assume that $[x] \cap [y] \neq \emptyset$, and seek to show that xRy . To do this, notice that if $[x] \cap [y] \neq \emptyset$, there must be some element s in both $[x]$ and $[y]$. By definition, this means that sRx and sRy ; by symmetry, we can rewrite this as xRs and sRy , which becomes xRy by transitivity and finishes our proof! \square

This result gives us a nice way to think of equivalence classes:

Definition. A **finite partition** of a set S is any collection of sets X_1, \dots, X_n such that

1. The union $X_1 \cup \dots \cup X_n$ of these sets is S .
2. The intersection of any two different sets X_i, X_j is \emptyset

Corollary. If S is a set with an equivalence relation R , then the collection of distinct equivalence classes given by R is a partition of S .

Proof. If $[x] \neq [y]$, then by the theorem above $[x] \cap [y]$ must be the empty set. As a result, the collection of distinct equivalence classes satisfy the second property of a partition. To see that their union is all of S , simply note that by reflexivity, we know that each $x \in S$ is in $[x]$, and so the union of all equivalence classes will give us all of the elements in S ! \square

6.3 Orderings and Posets

Equivalence relations can be thought of as generalizing of the idea of “equality”: pretty much every time you have an equivalence relation R , you can phrase the rule R describes as saying “is the same up to *blah* as” or “has the same *blah* as.” For instance, saying $a \equiv b \pmod n$ is just saying that a and b are the same “up to a multiple of n ,” that is, that their difference is just a multiple of n . Less numerically, if we let C be the set of people in CS225, the relation “has the same color shoes as” is an equivalence relation (check it!), and is just the idea of “we can group people together based on their shoe’s colors.”

In this section, we look at a way to generalize a different sort of relation: the idea of “comparing” things, i.e. \leq ! We start with a definition:

Definition. We say that a relation R on a set S is **antisymmetric** if for any distinct $x, y \in S$, if xRy holds, then yRx does not hold.

We say that a relation R is a **partial ordering** if it is reflexive, antisymmetric, and transitive. We call a set S with a partial ordering R a **poset**³⁰.

You’ve seen many examples of sets with orderings before:

- The relation \leq on the real numbers is an ordering! It’s reflexive (for any $x \in \mathbb{R}$, $x \leq x$), it’s antisymmetric (for any $x \neq y \in \mathbb{R}$, if $x \leq y$, then $y \not\leq x$), and it’s transitive (for any $x, y, z \in \mathbb{R}$, if $x \leq y$ and $y \leq z$, then $x \leq z$!)
- The relation “occurs earlier (or at the same place) in the dictionary than”, which for brevity we’ll call the **lexicographical order** is an ordering on the set of all words! It’s reflexive: any word occurs at the same place as itself. It’s antisymmetric: if one word occurs before another, then it definitely doesn’t also occur afterwards³¹; that is, because “cat” occurs before “cataclysm,” we know that “cataclysm” cannot occur before “cat.” Finally it’s transitive; if a word w_1 occurs before a word w_2 , which occurs before a word w_3 , we can conclude that w_1 itself occurs before w_3 .
- The relation “is a factor of” is an ordering on the positive integers! Given any integer n , it’s a factor of itself, so we’re reflexive; if n and m are distinct integers and n ’s a factor of m , then n is smaller than m , and so m cannot be a factor of n (so we’re antisymmetric), and finally if n ’s a factor of m and m ’s a factor of l , then n ’s a factor of l , so we’re transitive.

(Proof of that last claim: by definition, this “is a factor of” stuff means that $an = m$, $bm = l$ for two integers a, b . Combining gives you $abn = l$, i.e. that n ’s a factor of l .)

- Consider the set P of all breakfast foods, with the relation \geq defined by “is either equal to or tastier than.” For instance, we have

(delicious perfect pancakes) \geq (horribly burnt pancakes).

This is a poset! The “either equal to” part of our relation gives us reflexivity for free; antisymmetry and transitivity follow from the same arguments we used for \leq and the real numbers (except, y’know, with waffles instead of π .)

In a poset, when two objects x, y are such that either xRy or yRx , we call them **comparable**. Not all objects are comparable in a poset! For instance, the two numbers 2 and 3 are not comparable

³⁰Poset is a funny word. If you do not believe me, say it out loud a few times.

³¹Assuming you have a not-really-weird dictionary.

in our “is a factor of” poset: 2 is not a factor of 3, and 3 is not a factor of 2. Similarly, in our breakfast food poset,

(delicious perfect pancakes) , (delicious perfect french toast)

are two different objects such that neither are really obviously “tastier” than the other. This is OK, because in a poset we do not know that any two elements are comparable!

If we have a poset in which **every two elements** x, y are comparable, we call it a **totally ordered set**, because everything in it is comparable! Words under the dictionary order, and \leq on the real numbers, are examples of totally ordered sets, while the factors and breakfast food posets are not examples of totally ordered sets. We call a totally ordered set a **toset**³² for short.

Given a poset $P = (X, \leq)$, it can be very useful to visualize P by drawing it as a diagram! We do this as follows:

- Let $M_0 = \{x \in P \mid \text{there is no } y \neq x \in P \text{ with } x \leq y\}$. We call M_0 the collection of all **maximal** elements, and say that any element of M_0 is a maximal element in P .

At the top of a piece of paper, draw vertices in a row, one for each element of M_0 .

- Now, take the collection M_1 of all of the elements “directly beneath” M_0 ; that is, form the set

$$M_1 = \{x \in P \mid \exists y_1 \neq x \in P \text{ such that } x \leq y_1, \text{ but } \neg(\exists \text{ distinct } y_1, y_2 \neq x \in P \text{ such that } x \leq y_1 \leq y_2)\}$$

of all elements with exactly one object greater than them under our relation. Draw these elements in a row beneath the M_0 vertices, and draw a line from any element in M_0 to any element in M_1 whenever they are comparable.

- Now, take the collection

$$M_2 = \{x \in P \mid \exists \text{ distinct } y_1, y_2 \neq x \in P \text{ such that } x \leq y_1 \leq y_2, \text{ but } \neg(\exists \text{ distinct } y_1, y_2, y_3 \neq x \in P \text{ such that } x \leq y_1 \leq y_2 \leq y_3)\}$$

of all points with only two things greater than them under our relation. Draw these points beneath the M_1 points, and connect points in M_1 to points in M_2 if they are comparable.

- Repeat this until you’ve drawn all of P !

We call this diagram the **lattice diagram**, or **Hasse diagram**, for our poset.

This is perhaps overly abstract / best understood with an example. Consider the “is a divisor of” relation from before, except instead of working with all of \mathbb{Z} , let’s just let our set S be the set of all divisors of some number. Say, for instance, 90:

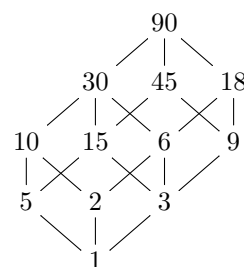
$$90, 45, 30, 18, 15, 10, 9, 6, 5, 3, 2, 1.$$

If we group elements as suggested above, we get

- $M_0 = 90$,
- $M_1 = 45, 30, 18$,
- $M_2 = 15, 10, 9, 6$,
- $M_3 = 5, 3, 2$,
- $M_4 = 1$.

If we draw the diagram as described, we get the lattice at right!

Notice that the lattice above actually contains all of the information about our poset: if we want to know if any two elements x, y



³²Toset is another funny word.

are comparable, we just need to find them in the diagram above, and see if there is a strictly ascending path from the lower of our elements to the higher; the existence of any such path will guarantee (by transitivity) that those elements are comparable! So, for example, $2 \leq 18$, as the ascending path $2 \rightarrow 6 \rightarrow 18$ demonstrates. However, 45 and 2 are incomparable, as we cannot find any path from 2 to 45 that is going up. This makes sense; because $2 \not\leq 45$ and $45 \not\leq 2$, we know that these two elements are incomparable!

There are some beautiful open questions we can talk about right now with these diagrams. In particular, consider the following two-player game:

- Player 1 picks out an element on the lattice, and deletes it, along with every element “beneath” that element (i.e. if player 1 pick out x , they delete x and every $y \leq x$ from our lattice.)
- Player 2 then picks out a remaining element in the lattice, and deletes it along with all elements beneath it.
- Players repeat this process until there are no elements left in the lattice. The player who chose the last element loses.

Determining who wins this game on many families of posets is often an open problem!

6.3.1 Maximal, minimal, greatest, and least

A useful idea when making the poset earlier was the idea of a **maximal** element: we generalize this idea here!

Definition. Given a poset (P, \leq) , we say that an element x is a **maximal** element if there is no other element $y \in P$ such that $x \leq y$; similarly, we say that an element x is a **minimal** element if there is no other $y \in P$ such that $y \leq x$.

Note that a poset can have many maximal and minimal elements, or none at all! For instance, the real numbers \mathbb{R} under the ordering \leq do not have any maximal or minimal elements: for any $x \in \mathbb{R}$, we can always find other real numbers that are bigger and smaller than x . As well, the breakfast food poset from before arguably has many maximal elements: “perfect pancakes” and “perfect waffles” and “perfect scones” are all breakfast foods that by definition don’t have things better than them, even if there isn’t necessarily a single one that beats the others!

If a poset P has exactly one maximal element, we call that element the **greatest** element in the poset; similarly, if P has exactly one minimal element, we call that element the **least** element in the poset. For instance, 90 is the greatest element in the divisor poset we drew above, and 1 is the least element.

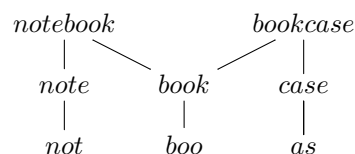
To help us make sense of this, we draw a second example here:

Example. Let \preceq be a relation on strings defined as follows: we say that string A is \preceq to string B if A is a “substring” of B : that is, if you can make B by taking A and adding some characters to either the start or end (or both!) of A . For example, “cat” is a substring of “concatenate”, so “cat” \preceq “concatenate;” similarly “and” is a substring of “sandwich,” so we have “and” \preceq “sandwich.”

If we do this, then \preceq forms a poset relation on any collection of strings (prove this, if you want an exercise!) In particular, if we look at $\{\text{bookcase, book, case, as, note, notebook, not, boo}\}$, we get the following poset:

If we do this, then the process described above generates the Hasse diagram at right:

This poset has two maximal elements, “notebook” and “bookcase,” and three minimal elements: “not,” “boo,” and “as.”



See tutorial 8 for more examples!

With relations now (hopefully) well-understood, we turn to the last subject of this chapter: **functions**!

6.4 Functions

6.4.1 The Definition of a Function

Definition. A function $f : A \rightarrow B$ consists of three parts:

- A set A , called the **domain** of f ; we think of A as the set of all valid inputs to the function f .
- A set B , called the **codomain** of f ; we think of B as the set of all valid potential outputs of the function f .
- A rule f , that matches (or sends, or maps) elements in A to elements in B . This rule satisfies the following property:

For every $a \in A$, there is exactly one $b \in B$ such that $f(a) = b$.

In other words, we never have a value $a \in A$ for which $f(a)$ is undefined, nor do we ever have a value $a \in A$ where there are multiple outputs (i.e. we never have $f(a) = b$ and $f(a) = c$ for different values a, c .)

Typically, to define a function we'll write something like "Consider the function $f : (0, \infty) \rightarrow \mathbb{R}$, defined by the rule $f(x) = \frac{1}{x}$ ". This definition tells you three things: what the domain is (the set the arrow starts from, which is $(0, \infty)$ in this case), what the codomain is (the set the arrow points to, which is \mathbb{R} in this case), and the rule used to define f .

Example. The function $f : \mathbb{R} \rightarrow \mathbb{R}$ defined by the rule $f(x) = x^2$ is a function, with domain and codomain both equal to \mathbb{R} .

To see why, note that by definition f is a function if for every x in the domain, there is exactly one value y in the codomain that x is sent to. But for any $x \in \mathbb{R}$, there is exactly one possible number that is x^2 , and so there is exactly one possible element $y = x^2$ that our function sends x to.

Note that the only condition we were concerned about was ensuring that for every value x in the domain, there is exactly one value in the codomain that our function sends x to. A common misconception that people have about functions is that we must also ensure that every value in the codomain is mapped to. This is not at all necessary! Even though there is no x in the domain such that $f(x) = x^2 = -1$, we still think that $f : \mathbb{R} \rightarrow \mathbb{R}$, $f(x) = x^2$ is a function.

Example. The relation $f : \mathbb{R} \rightarrow \mathbb{R}$ defined by the rule " $f(x) = y$ if and only if $x = y^2$ " is not a function. There are many reasons for this:

- There are values in the domain that do not get mapped to any values in the codomain by our rule. For instance, consider $x = -1 \in \mathbb{R}$. There is no value $y \in \mathbb{R}$ such that $-1 = y^2$; therefore, x is not mapped to any value y in the codomain, and so we do not regard f as a function.
- There are also values in the domain that get mapped to multiple values in the codomain by our rule. For instance, consider $x = 1 \in \mathbb{R}$. Because $1 = y^2$ has the two solutions $y = \pm 1$, this rule maps $x = 1$ to the two values $y = \pm 1$; this is another reason why f is not a function.

Example. The relation $f : [0, \infty) \rightarrow [0, \infty)$ defined by the rule $f(x) = y$ if and only if $x = y^2$ is a function. To see why, take any

x in the domain $[0, \infty)$; if f is a function, there should be exactly one value y in the codomain $[0, \infty)$ that gets matched to x . This is indeed true; if $x \geq 0$ then we can simplify $x = y^2$ to $\sqrt{x} = |y|$, and if we know that $y \in [0, \infty)$ we can remove the absolute value signs to get $\sqrt{x} = y$, and in particular see that there is exactly one value that x is mapped to.

Typically, with functions like the first and third examples we've studied where the rule is relatively simple to write down, we will write something like " $f : \mathbb{R} \rightarrow \mathbb{R}, f(x) = x^2$ " or " $f : [0, \infty) \rightarrow [0, \infty), f(x) = \sqrt{x}$."

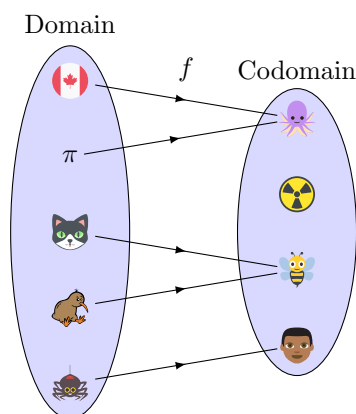
With that said, not all functions can be written down this nicely. Consider the following example:

Example. The map $f : \{(\text{🇨🇦}), \pi, (\text{🐱}), (\text{🐘}), (\text{🦀})\} \rightarrow \{(\text{🦋}), (\text{☢}), (\text{🐝}), (\text{👤})\}$ defined by the rules

$$f(\text{🇨🇦}) = (\text{🦋}), \quad f(\pi) = (\text{🦋}), \quad f(\text{🐱}) = (\text{🐝}), \quad f(\text{🐘}) = (\text{🐝}), \quad f(\text{🦀}) = (\text{👤})$$

is a function, because every element in the domain is sent to exactly one element in the codomain.

A useful way to visualize functions defined in this piece-by-piece fashion is with a diagram: we draw the domain at left, the codomain at right, and connect an element x in the domain to an element y in the codomain when our function maps x to y :



6.4.2 Useful Function Properties: Range

Now that we know what a function is, we now turn our attention to various sorts of **properties** that functions can have that may make them interesting or useful to study. We start with one that you've likely encountered before: the idea of the **range** of a function.

Definition. Take any function $f : A \rightarrow B$. We define the **range** of f as the set of all values in the codomain that our function sends values in the domain to. Formally, we express this as follows:

$$\text{range}(f) = \{y \in B \mid \text{there is some } x \in A \text{ such that } f(x) = y\}.$$

Example. Consider the function $f : \mathbb{R} \rightarrow \mathbb{R}$ given by the rule $f(x) = x^2 - 1$. This function has range $[-1, \infty)$.

To see why, simply notice that for any $y \in \mathbb{R}$, we can express $y = x^2$ if and only if $y \in [0, \infty)$; if y is indeed nonnegative then we can set $x = \sqrt{y}$ to get a value of x such that $x^2 = y$, while if y is negative it is impossible to square any real number and get y as the output. Therefore, we can express $y = x^2 - 1$ if and only if $y \in [-1, \infty)$, as subtracting 1 from the right-hand-side just increases the range of possible y -values by 1.

Example. Consider the function $f : \mathbb{Z} \rightarrow \mathbb{R}$ given by the rule $f(x) = 2x - 1$. This function has range equal to all of the odd numbers.

This is because for any odd number n , we can write $n = 2k - 1$ for some $k \in \mathbb{Z}$ by definition; this gives us $f(k) = n$, and shows us that our range contains all odd numbers. Conversely, for any $k \in \mathbb{Z}$, $f(k) = 2k - 1$ is an odd number, so we know that odd numbers are the only possible things we can get in our range; as a result we have that the set of all odd numbers **is** our range.

6.4.3 Useful Function Properties: Injective and Surjective

The idea of **range** above lets us talk about the values that our function can output. Relatedly, the definitions of **injective** and **surjective** functions let us talk about “how often” our function outputs a given value:

Definition. Take any function $f : A \rightarrow B$. We say that f is:

- **injective**, or one-to-one, if for every $b \in B$, there is **at most one** $a \in A$ such that $f(a) = b$.
- **surjective**, or onto, if for every $b \in B$, there is **at least one** $a \in A$ such that $f(a) = b$.
- **bijective** if for every $b \in B$, there is **exactly one** $a \in A$ such that $f(a) = b$.

Note that bijective literally just means “injective and surjective.”

Example. Consider the function $f : \mathbb{R} \rightarrow \mathbb{R}$ given by the rule $f(x) = x^2$.

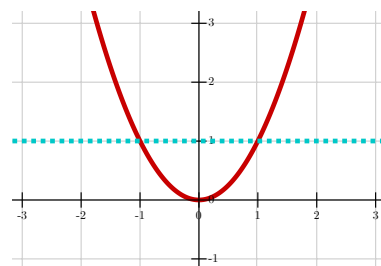
This function is not injective. To see why, we just need to show that the injective condition fails somewhere; that is, that there is some value y in the codomain with more than one corresponding value x in the domain. This is not hard: if we look at $y = 1$, for instance, we can see that $x = 1$ and $x = -1$ both get mapped to y , as $f(1) = (1)^2 = 1$ and $f(-1) = (-1)^2 = 1$. So the injective condition fails.

This function is also not surjective. To see why, we just need to show that the surjective condition fails somewhere; that is, that there is some y in the codomain that no values in the domain map to. This is also straightforward; if we pick $y = -2$, for instance, we know that there is no x such that $f(x) = x^2 = -2$, and so have shown that the surjective condition fails.

Example. Consider the function $f : \mathbb{N} \rightarrow \mathbb{N}$ given by the rule $f(n) = 2n$.

This function is injective. There are several ways to show this. One technique, that you’ll see in many places, is the following:

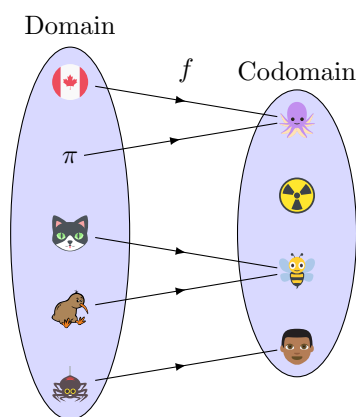
- Take any value y in the codomain.
- Let x_1, x_2 be any two values in the domain such that $f(x_1) = y = f(x_2)$.
- From here, the statement that f is one-to-one is equivalent to saying that x_1 and x_2 are equal (as if they must be equal then we’ve proven that there’s no way to have multiple different values in the domain mapping to the same y in the codomain.) So, try to prove that x_1 must be equal to x_2 . If you succeed, you’ve proven that f is one-to-one; if you cannot, then perhaps f is not injective!



In this specific case, take any y in the codomain, and suppose that x_1, x_2 are two values in our domain such that $f(x_1) = y = f(x_2)$. But $f(x_1) = 2x_1 = f(x_2) = 2x_2$ implies that $x_1 = x_2$! As a consequence, we cannot have two different values of x that map to the same y , and so our function is injective. (This method is overkill here and you can show this directly with less work, but it's worth seeing this proof method in a simple case before using it on something trickier.)

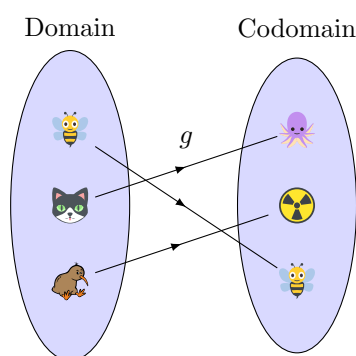
This function is not surjective, as there are values in the codomain (for example, any odd number in \mathbb{N}) that are not mapped to by any element in the domain.

Example. Consider the map $f : \{\text{🇨🇦}, \pi, \text{🐱}, \text{🦘}\} \rightarrow \{\text{🦋}, \text{☢️}, \text{🐝}, \text{👤}\}$ that we looked at earlier.



This function is not injective, as there is an element in the codomain (say, 🦋) that is mapped to by more than one element in the domain. It's also not surjective, as there are elements in the codomain (say, ☢️) that are not mapped to by any elements in the domain.

Example. Consider the function $g : \{\text{🐝}, \text{🐱}, \text{🦘}\} \rightarrow \{\text{🦋}, \text{☢️}, \text{🐝}\}$ with rules given by the diagram



This function is injective and surjective, as every element in the codomain is mapped to by exactly one element in the range.

Sometimes we look at more theoretical sorts of questions around injective and surjective functions:

Claim. If $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ are both surjective functions, then $g \circ f : X \rightarrow Z$ is also a surjective function.

Proof. The first step in a proof like this, as with many proofs, is to carefully write out what we know:

1. We know that $f : X \rightarrow Y$ is . In other words: for any $y \in Y$, there is an $x \in X$ such that $f(x) = y$.

2. We also know that $g : Y \rightarrow Z$ is surjective; that is, for any $z \in Z$, there is a $y \in Y$ such that $g(y) = z$.

We want to show that $g \circ f$ is surjective as well: that is, that for any $z \in Z$, there is some $x \in X$ with $g(f(x)) = z$. How can we do this?

Well: take any $z \in Z$. We want to find a value of x such that $g(f(x)) = z$, and all we have are properties (1) and (2) above.

None of them immediately give us a value of x that does this, but point (2) *does* tell us that for our value $z \in Z$, we can always find a value $y \in Y$ such that $g(y) = z$.

If we take that value of y and apply point (1) above, it tells us that we can find a value of $x \in X$ such that $f(x) = y$.

Together, then, this means that for any value of z , we can find corresponding values of y, x such that $g(f(x)) = g(y) = z$. But this is what we wanted: for any $z \in Z$, we found an $x \in X$ such that $(g \circ f)(x) = z$! So we've proven our claim. \square

6.4.4 Cardinality

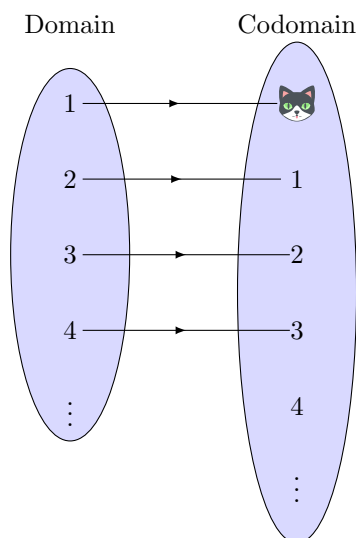
One thing we noticed in class is that we could make a bijection $f : A \rightarrow B$ if and only if the two sets A, B had the same number of elements. With this idea in mind, we make the following definition

Definition. We say that two sets A, B are the same size (formally, we say that they are of the same **cardinality**), and write $|A| = |B|$, if and only if there is a bijection $f : A \rightarrow B$.

The nice thing about this definition is that it lets us talk about **infinite sets**, not just finite ones! To test these ideas out, let's start with some calculations to build our intuition:

Question. Are the sets \mathbb{N} and $\mathbb{N} \cup \{\text{cat}\}$ the same size?

Answer. Well: we know that they can be the same size if and only if there is a bijection between one and the other. So: let's try to make a bijection! In the typed notes, the suspense is somewhat gone, but (at home) imagine yourself taking a piece of paper, and writing out the first few elements of \mathbb{N} on one side and of $\mathbb{N} \cup \{\text{cat}\}$ on the other side. After some experimentation, you might eventually find yourself with the following map:



Formally, this is a function defined by the rule $f(1) = \text{cat}$ and $f(n) = n - 1$ for any $n \geq 2$.

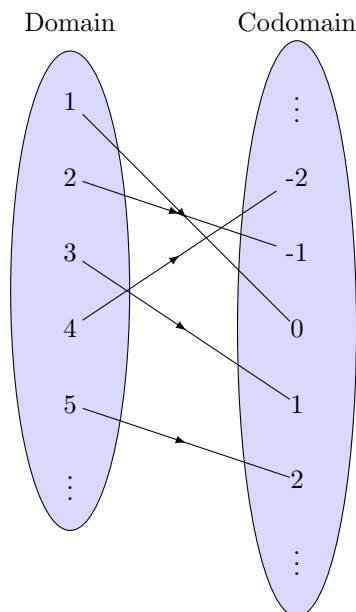
This is clearly a bijection; so these sets are the same size!

In a rather crude way, we have shown that adding one more element to a set as “infinitely large” as the natural numbers doesn’t do anything to it! – the extra element just gets lost amongst all of the others.

This trick worked for one additional element. Can it work for infinitely many? Consider the next proposition:

Proposition. The sets \mathbb{N} and \mathbb{Z} are the same cardinality.

Proof. Consider the following map:



Formally, this function can be defined as follows:

$$f(n) = \begin{cases} \frac{n-1}{2}, & \text{if } n \text{ is odd, or} \\ -\frac{n}{2}, & \text{if } n \text{ is even.} \end{cases}$$

This is a bijection (justify this to yourself if you don’t see why!), so these sets are the same cardinality. \square

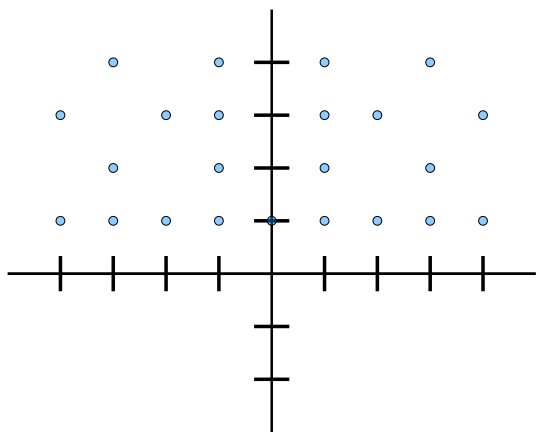
So: we can in some sense “double” infinity! Strange, right? Yet, if you think about it for a while, it kind of makes sense: after all, don’t the natural numbers contain two copies of themselves (i.e. the even and odd numbers?) And isn’t that observation what we just used to turn \mathbb{N} into \mathbb{Z} ?

After these last two results, you might be beginning to feel like all of our infinite sets are the same size. In that case, the next result will hardly surprise you:

Proposition. The sets \mathbb{N} and \mathbb{Q} are the same cardinality.

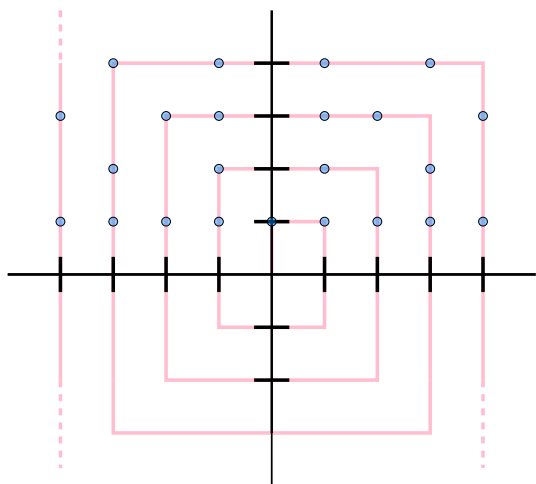
Proof. First, take any rational number $x \in \mathbb{Q}$. By definition, note that we can write x as a fraction $\frac{p}{q}$, where p and q are integers and $q > 0$; moreover, by dividing through by common multiples, we can make it so that p and q have no factors in common.

This process assigns each rational number to exactly one such fraction $\frac{p}{q}$. For each such fraction, draw a point at (p, q) in \mathbb{R}^2 :



In the picture above, every rational number is represented by exactly one blue dot!

Now, on this picture, draw a spiral that starts at $(0,0)$ and goes through every point of $\mathbb{Z} \times \mathbb{Z}$, as depicted below:



We use this spiral to define our bijection from \mathbb{N} to \mathbb{Q} as follows:

$f(n)$ = the n -th rational number whose point our spiral has crossed, found by starting at $(0,0)$ and walking out along our spiral.

This function hits every rational number exactly once by construction; thus, it is a bijection from \mathbb{N} to \mathbb{Q} . Consequently, \mathbb{N} and \mathbb{Q} are the same size. \square

6.4.5 The Reals

At this point, it almost seems inevitable that **every** infinite set will wind up having the same size!

This is false.

Theorem. The sets \mathbb{N} and \mathbb{R} have different cardinalities.

Proof. (This is **Cantor's famous diagonalization argument**.) Suppose not – that they were the same cardinalities. As a result, there is a bijection between these two sets! Pick such a bijection $f : \mathbb{N} \rightarrow \mathbb{R}$.

For every $n \in \mathbb{N}$, look at the number $f(n)$. It has a decimal representation. Pick a number $a_{n,\text{trash}}$ corresponding to the integer part of $f(n)$, and $a_{n,1}, a_{n,2}, a_{n,3}, \dots$ that correspond to the digits after the decimal place of this decimal representation – i.e. pick numbers $a_{n,i}$ such that

$$f(n) = a_{n,\text{trash}}.a_{n,1}a_{n,2}a_{n,3} \dots$$

For example, if $f(4) = 31.125$, we would pick $a_{4,\text{trash}} = 31, a_{4,1} = 1, a_{4,2} = 2, a_{4,3} = 5$, and $0 = a_{4,4} = a_{4,5} = a_{4,6} = \dots$, because the integer part of $f(4)$ is 31, its first three digits after the decimal place are 1, 2, and 5, and the rest of them are zeroes.

Now, get rid of the $a_{n,\text{trash}}$ parts, and write the rest of these numbers in a table, as below:

$f(1)$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	\dots
$f(2)$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	
$f(3)$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	
$f(4)$	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	
\vdots	\vdots				\ddots

In particular, look at the entries $a_{1,1}a_{2,2}a_{3,3}\dots$ on the diagonal.

Let's make a number that disagrees with $0.a_{1,1}a_{2,2}a_{3,3}\dots$ in every possible decimal place! We define such a number B as follows:

- Define $b_i = 2$ if $a_{i,i} \neq 2$, and $b_i = 8$ if $a_{i,i} = 2$. (There is nothing special about 2 and 8 here; the relevant property is that $b_i \neq a_{i,i}$ for every i ! Feel free to pick your own values here.)
- Define B to be the number with digits given by the b_i – i.e.

$$B = .b_1b_2b_3b_4\dots$$

Because B has a decimal representation, it's a real number! So, because our function f is a bijection, it must have some value of n such that $f(n) = B$. But the n -th digit of $f(n)$ is $a_{n,n}$ by construction, and the n -th digit of B is b_n – by construction, these are different numbers! So $f(n) \neq B$, because they disagree at their n -th decimal place!

This is a contradiction to our original assumption that such a bijection existed. Therefore, we know that no such bijection can exist: as a result, we've shown that the natural numbers are of a strictly “different” size of infinity than the real numbers. \square

Crazy! This is how in a sense about how half of the mathematics we do as researchers goes: by combining relatively mundane ideas (the concepts of bijection and proof by contradiction) we can get to remarkably strange results (there are different sizes of infinity!)

6.4.6 Combining Functions and Inverse Functions

Given any of these building block functions, we can combine them in several ways:

- Given any two functions $f, g : A \rightarrow \mathbb{R}$, for $A \subset \mathbb{R}$, we can combine these functions via **arithmetic**: that is, we can define the functions $f + g$, $f \cdot g$, and $f - g$, all of which have domain A and codomain \mathbb{R} . If $g(x) \neq 0$ for all $x \in A$, we can also form the function $\frac{f}{g}$, which also has domain A and codomain \mathbb{R} .
- Given any two functions $f : A \rightarrow B$, $g : C \rightarrow D$ where $\text{range}(f) \subseteq C$, we can form the **composition** $g \circ f : A \rightarrow D$ of these two functions, defined by the rule $g \circ f(a) = g(f(a))$ for any $a \in A$.

For example, if $f, g : \mathbb{R} \rightarrow \mathbb{R}$ and $f(x) = x+1, g(x) = x^2-1$, we would have $g \circ f(x) = g(f(x)) = g(x+1) = (x+1)^2-1 = x^2+2x$. Notice that this is different to $f \circ g(x) = f(g(x)) = f(x^2-1) = (x^2-1)+1 = x^2$; in general, $f \circ g$ and $g \circ f$ are usually different functions.

- Finally, we can define a function in a **piecewise** manner by giving different rules for different parts of its domain: e.g. we could define the absolute value $f : \mathbb{R} \rightarrow \mathbb{R}, f(x) = |x|$ by

$$|x| = \begin{cases} x, & \text{if } x \geq 0 \\ -x, & \text{if } x < 0 \end{cases}$$

By combining “basic” functions like $\sin(x), \cos(x), \tan(x), e^x, \ln(x), |x|, \sqrt{x}, \frac{1}{x}, x$ using these tools you can create almost every function that we study in mathematics!

There is one exception, though: we’re missing the idea of an **inverse** function. We define this here:

Definition. Take a function $f : A \rightarrow B$. We say that $g : B \rightarrow A$ is an **inverse** of f if the following two properties hold:

- For all $a \in A$, we have $g(f(a)) = a$.
- For all $b \in B$, we have $f(g(b)) = b$.

The idea here is that g “undoes” whatever it is that f does, returning the same input that was entered. If such an inverse function g exists, we denote it by writing f^{-1} .

Equivalently, we can define the inverse by defining the **identity function** i_X of any set X as follows: for any $x \in X$, $i_X(x) = x$. In other words, the identity function is the function whose input is its output; it “does nothing.”

A function $f : X \rightarrow Y$ has another function $f^{-1} : Y \rightarrow X$ as its inverse if $f \circ f^{-1} = i_Y$ and $f^{-1} \circ f = i_X$. (This is the exact same as above, except we’re talking about just two equations of functions, as opposed to a ton of equations with values plugged in!)

Example. The function $f : \mathbb{R} \rightarrow \mathbb{R}, f(x) = 3x - 1$ has an inverse; namely, it has the inverse $f^{-1} : \mathbb{R} \rightarrow \mathbb{R}$ defined by $f^{-1}(x) = \frac{x+1}{3}$. To prove this, we simply check the two properties listed above for being an inverse:

- For all $a \in \mathbb{R}$, we have $f^{-1}(f(a)) = a$. We check this, and indeed: $f^{-1}(f(a)) = f^{-1}(3a - 1) = \frac{(3a - 1) + 1}{3} = a$.
- For all $b \in \mathbb{R}$, we have $f(f^{-1}(b)) = b$. We check this as well, and it works: $f(f^{-1}(b)) = f\left(\frac{b+1}{3}\right) = 3 \cdot \frac{b+1}{3} - 1 = b$.

A process that can sometimes find an inverse (not always, but sometimes) is the following:

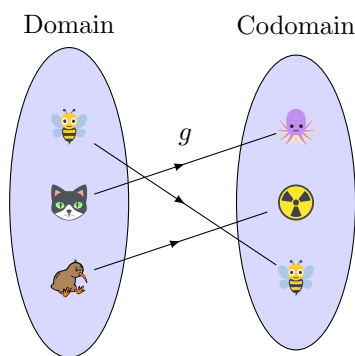
- Take any function rule $f(x)$ that you want to find an inverse for.
- Look at the equation $f(x) = y$. This is currently an equation for y in terms of x .
- Try to “reverse” this: i.e. solve this equation for x in terms of y ! You should get some expression of the form $g(y) = x$, if this works.
- If you succeed, then this g is *probably* f^{-1} (though you need to check whether $g \circ f(x) = x$ for all x and $f \circ g(y) = y$ for all y first.)

For example, if we took the rule $f(x) = 3x - 1$ from above, we could actually find its inverse as follows:

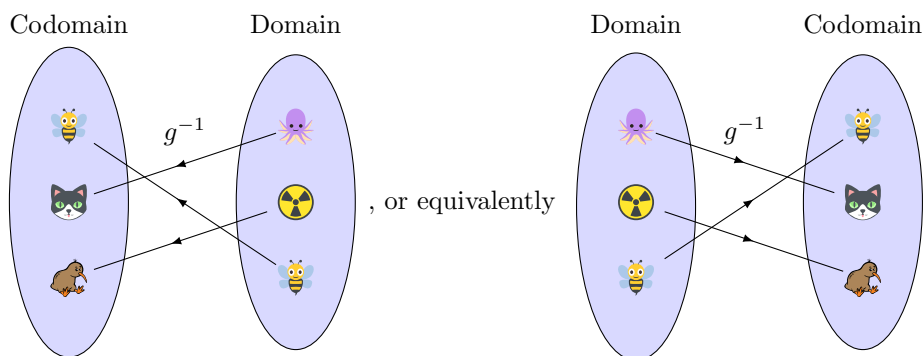
- Set $3x - 1 = y$.
- Try to solve for x ; you can do this by adding 1 to both sides to get $3x = y + 1$, and thus $x = \frac{y+1}{3}$.
- This gives us a function $g(y) = \frac{y+1}{3}$. This is the inverse of f , as we proved earlier!

For functions described by diagrams, inverses are remarkably straightforward to find:

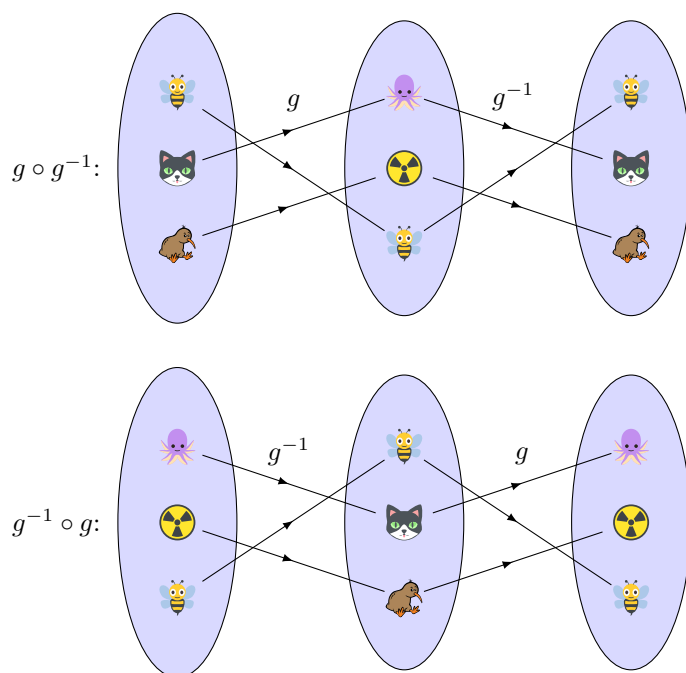
Example. The function $g : \{\text{bee}, \text{cat}, \text{dino}\} \rightarrow \{\text{spider}, \text{radio}, \text{bee}\}$ that we studied before, with diagram



has an inverse given by “reversing” all of these arrows: that is, if we define $g^{-1} : \{\text{spider}, \text{radio}, \text{bee}\} \rightarrow \{\text{bee}, \text{cat}, \text{dino}\}$ by the diagram



we have



Checking the pictures above verifies that $g \circ g^{-1}$ and $g^{-1} \circ g$ satisfy the inverse property.

Inverses are closely tied to the injective and surjective properties we studied earlier, as the following theorem

Theorem. If a function $f : X \rightarrow Y$ is a bijection, then it has an inverse $f^{-1} : Y \rightarrow X$. Conversely, if a function has an inverse, then it must be a bijection.

Proof. We start by proving the first half of this result (that bijection \Rightarrow an inverse exists.) To do this, start by noting that by definition, if f is a bijective function, then for every $y \in Y$ there is exactly one $x \in X$ such that $f(x) = y$.

Define the function $g : Y \rightarrow X$ as follows: for any $y \in Y$, let $g(y)$ be defined as the unique $x \in X$ such that $f(x) = y$. This is a function, as this definition gives us exactly one output for every $y \in Y$ because f is a bijection.

I claim that $g = f^{-1}$; i.e. that $f \circ g(y) = y$ for all $y \in Y$, and that $g \circ f(x) = x$ for all $x \in X$.

To see this, take any $y \in Y$. Notice that by definition $g(y)$ is the unique x such that $f(x) = y$; therefore $f(g(y)) = f(x) = y$, as desired.

Now, take any $x \in X$, and let $y = f(x)$. By definition, $g(y)$ is the unique x such that $f(x) = y$: therefore $g(f(x)) = g(y) = x$, as desired!

This establishes the first half of our proof. To see the second half (that any invertible function must be a bijection,) simply notice the following: if $f : X \rightarrow Y$ has inverse $f^{-1} : Y \rightarrow X$, then for any $y \in Y$ there is exactly one output $f^{-1}(y)$, because f^{-1} is a function and thus has precisely one output for any input.

As a consequence, there must exist at least one x in X , namely $x = f^{-1}(y)$, such that $f(x) = y$; this is because $f(f^{-1}(y)) = y$ by definition. As well, there cannot exist distinct values $x_1, x_2 \in X$ such that $f(x_1) = f(x_2) = y$; this is because $f^{-1}(y) = f^{-1}(f(x_1)) = x_1$ and $f^{-1}(y) = f^{-1}(f(x_2)) = x_2$ must be equal, as noted above.

In other words, f is a bijection, as claimed. □

This property comes in quite handy! In general, it can be hard to see whether a function has an inverse directly; it is often easier to check the injective and surjective properties, and conclude via the above that our function has an inverse.

Chapter 7: Combinatorics

Week 10

UoA 2018

7.1 How to Count

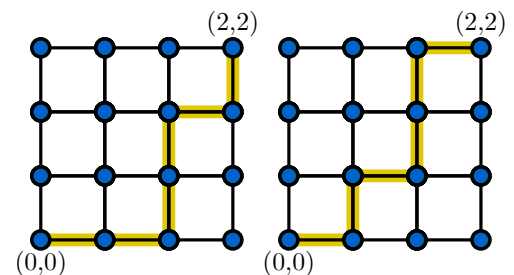
This might seem like a silly section title; counting, after all, is something that you learned how to do at a very young age! So let's clarify what we mean by "counting."

On one hand, it is very easy to see that there are four elements in a set like

$$A = \{3, 5, 7, \text{Snape}\}.$$

Things can get trickier than this, however. Consider the following five problems, in which you are asked to "count" some object:

- How many strings of five letters are **palindromes** (i.e. can be read the same way forwards and backwards?)
- **Pizza My Heart** is a great pizza place that I loved back when I was a lecturer in California. Suppose that I went there and they had the following deal on pizzas: for 13\$, they'd sell you a pizza with any two different vegetable toppings, or any one meat topping. If there are m meat choices and v vegetable choices, and for each pizza I also got to choose one of s sauces, how many different pizzas could I order under this deal?
- How many injective functions exist from $\{0, 1, 2, 3, 4\}$ to $\{0, 1, 2, 3, 4, 5\}$?
- A **lattice path** in the plane \mathbb{R}^2 is a path joining integer points via steps of length 1 either upward or rightward. For any two natural numbers $a, b \in \mathbb{N}$, how many lattice paths are there from $(0, 0)$ to (a, b) ?
- How many seven-digit phone numbers exist in which the digits are all nondecreasing?
- What is the coefficient of $x^8 y^7$ in $(x + y)^{15}$?
- How many positive integers between 1 and 10,000 are not divisible by 2, 3 or 5?



All of these are "counting" problems, in that they're asking you to figure out how many objects of a specific kind exist. However, because the sets in question are trickily defined, these problems are much harder than our "how many elements are in $\{3, 5, 7, \text{Snape}\}$ " question.

To approach them, we'll need some new counting techniques! This is the goal of this chapter: we're going to study **combinatorics**, the art of counting, and develop techniques for solving problems like the ones above.

7.1.1 How to Count: Multiplication

We start by considering a new problem:

Problem. Suppose that we have k different postcards and n friends. In how many ways can we mail out all of our postcards to our friends?

Solution. In the setup above, a valid “way” to mail postcards to friends is some way to assign each postcard to a friend (because we’re mailing out all of our postcards.) In other words, a “way” to mail postcards is just a function from the set³³ $[k] = \{1, 2, 3, \dots, k\}$ of postcards to our set $[n] = \{1, 2, 3, \dots, n\}$ of friends!

In other words, we want to find the size of the following set:

$$A = \left\{ f \mid f \text{ is a function with domain } [k] \text{ and codomain } [n] \right\}$$

We can do this! Think about how any function $f : [k] \rightarrow [n]$ is constructed. For each value in $[k] = \{1, 2, \dots, k\}$, we have to pick exactly one value from $[n]$. Doing this for each value in $[k]$ completely determines our function; furthermore, any two functions f, g are different if and only if there is some value $i \in [k]$ at which we made a different choice (i.e. where $f(i) \neq g(i)$.)

$$\underbrace{\boxed{n \text{ choices}} \cdot \boxed{n \text{ choices}} \cdot \dots \cdot \boxed{n \text{ choices}}}_{k \text{ total slots}}$$

Consequently, we have

$$\underbrace{n \cdot n \cdot \dots \cdot n}_{k \text{ } n\text{'s}} = n^k$$

total ways in which we can construct distinct functions. This gives us this answer n^k to our problem!

This looks like an excellent sort of answer to a counting problem: given a set defined by parameters n, k , we created a closed-form algebraic expression n^k for the number of elements in that set! Again, under any theory of counting that we come up with, this should count as a pretty good answer.

Alongside our answer, we also came up with a fairly interesting **method** for counting at the same time. Specifically, we had a set A of the following form:

1. Each element f of A could be constructed by making k choices in a row.
2. Each time we made one of those choices, we had n total possibilities to pick from. Also, none of those choices affected the possibilities for our other choices; that is, we could make any choice we wanted for a value that f would output, and it would not affect the possibilities for the values f could output on any other values.
3. Therefore, we had $\underbrace{n \cdot n \cdot \dots \cdot n}_{k \text{ } n\text{'s}} = n^k$ total elements in A .

This can be generalized as follows:

Observation. (Multiplication principle.) Suppose that you have a set A , with the following properties:

- Each element of A can be thought of as the consequence of making k consecutive choices.
- There is a fixed³⁴ number n_i of possibilities for the i -th choice made in constructing any such element of A .

³³Some useful notation: $[n]$ denotes the collection of all integers from 1 to n , i.e. $\{1, 2, \dots, n\}$.

³⁴By “fixed,” we mean the following: the number of such choices is not affected by our other choices. That is, we’ll always have n_i options for our i -th choice, no matter what our earlier choices actually were.

Then there are

$$n_1 \cdot n_2 \cdot \dots \cdot n_k = \prod_{i=1}^k n_i$$

total elements in A .

A useful special case of this principle is the following:

Observation. (Ordered choice with repetition.) Suppose that you are choosing k objects from a set of n things, where you care about the order in which you choose your objects and can repeatedly pick the same thing if desired. There are $\boxed{n^k}$ many ways to make such a choice.

This principle is remarkably handy! With it, we can already answer one of our five problems from earlier:

Problem. How many strings of five letters are **palindromes** (i.e. can be read the same way forwards and backwards?)

Solution. First, notice that any five-letter palindrome can be constructed by taking an arbitrary three-letter string and sticking its second and first characters at the end of the string: e.g. you can transform “rad” to “radar,” “ten” to “tenet” and “eev” to “eevee.” This process is clearly reversible: i.e. you can take any five-letter palindrome and cut off its last two letters to get a 3-letter string. Therefore this process is a bijection, and so the set we are counting is equal in size to the set of all three-letter strings.

This can be counted by our multiplication principle! Any such three-letter string is formed by making three choices in a row, and we have 26 choices for each letter; this gives us $\boxed{26^3}$ many such strings, and thus 26^3 many five-letter palindromes.

This can get a bit more complex, however. Let’s try changing our postcard problem a bit from before:

Problem. Suppose that we still have n different kinds of postcards, k friends, and that we still want to mail these postcards to our friends. Last time, however, it was possible that we just mailed all of our postcards to the same friend. That’s a bit silly, so let’s add in a new restriction: let’s never send any friend more than one postcard.

In how many ways can we mail out postcards now?

We can still describe each way of sending postcards as a sequence of choices:

$$\underbrace{\boxed{? \text{ choices}} \cdot \boxed{? \text{ choices}} \cdot \dots \cdot \boxed{? \text{ choices}}}_{k \text{ total slots}}$$

As before, we still have n possibilities for who we can send our first card to. However, the “ordered choice with repetition” principle doesn’t immediately apply here: because we don’t want to repeat any of our friends, we only have $n - 1$ choices for our second slot, instead of n as before! In general, we have the following sequence of choices:

$$\underbrace{\boxed{n \text{ choices}} \cdot \boxed{n-1 \text{ choices}} \cdot \boxed{n-2 \text{ choices}} \cdot \dots \cdot \boxed{n-(k-1) \text{ choices}}}_{k \text{ total slots}},$$

which translates into

$$n \cdot (n - 1) \cdot \dots \cdot (n - (k - 1))$$

many choices in total.

(A common question I get here: why do we go to $n - 1$ and not n in the product above? Well: we have k total slots. In the first slot, none of our choices were eliminated yet! In the second slot, however, we've eliminated one choice with our first slot. By the third slot we've eliminated two possibilities, by the fourth we've eliminated three possibilities, and in general in the i -th slot we've eliminated $i - 1$ possibilities. This leaves us with $k - 1$ possibilities eliminated by the time we get to the k -th slot!)

Notice that if $k > n$, the above product contains a $n - n$ term and is 0. Otherwise, a convenient way to describe the above for $k \leq n$ is as the following quantity:

$$\begin{aligned} n \cdot (n - 1) \cdot \dots \cdot (n - (k - 1)) &= \frac{(n \cdot (n - 1) \cdot \dots \cdot (n - (k - 1))) \cdot ((n - k) \cdot (n - (k + 1)) \cdot \dots \cdot 3 \cdot 2 \cdot 1)}{((n - k) \cdot (n - (k + 1)) \cdot \dots \cdot 3 \cdot 2 \cdot 1)} \\ &= \frac{n!}{(n - k)!}. \end{aligned}$$

Here, by $n!$ we mean **n “factorial”**³⁵ the product of all of the natural numbers between 1 and n inclusive. For example, $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$.

We summarize this result in the following observation:

Observation. (Ordered choice without repetition.) Suppose that you are choosing k objects from a set of n things, where you care about the order in which you choose your objects, but can only pick an object at most once. There are $\frac{n!}{(n - k)!}$ many ways to make such a choice if $k \leq n$, and 0 ways otherwise (as we'll run out of choices!)

We can use this idea to answer another one of our problems from before:

Problem. How many injective functions exist from $\{0, 1, 2, 3, 4\}$ to $\{0, 1, 2, 3, 4, 5\}$?

Solution. Think of defining a function $f : \{0, 1, 2, 3, 4\} \rightarrow \{0, 1, 2, 3, 4, 5\}$ as just making five choices for where to send each of $f(0), f(1), f(2), f(3)$, and $f(4)$ in the codomain set $\{0, 1, 2, 3, 4, 5\}$. Because our function is injective, we never send any two elements to the same value in the codomain; i.e. we never repeat any choice! Therefore, by our logic above, because the domain contains 5 elements and the codomain contains 6 values to choose from, there are $\frac{6!}{(6 - 5)!} = \frac{6!}{1} = 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 = 720$ many such injective functions.

Let's tweak the postcard problem again!³⁶

Problem. Suppose that we have just one friend that we want to send postcards to. We still have n different kinds of postcards,

³⁵Note that by convention, we define $0! = 1$. There are three reasons commonly given for this:

- Another definition for $n!$ is the number of ways of ordering a list of n objects. There is exactly one way to order an empty list, so $0! = 1$.
- You can interpret $0!$ as the “empty product;” if you think of $n!$ as the product of the first n positive integers, then $0!$ is the product of nothing. Because 1 is the multiplicative identity, it makes sense to return 1 here, by the same reasoning that we use to say that $n^0 = 1$ for any nonzero n .
- It makes all of our formulas work! I.e. if you didn't have $0! = 1$, you'd have to have a ton of special cases in many of the formulas in this class.

³⁶In mathematics: whenever you have a problem at hand, constantly look for modifications like these to make to the problem! If you're stuck, it can give you different avenues to approach or think about the problem; conversely, if you think you understand the problem, this can be a way to test and deepen that understanding.

but now want to send that one friend k different postcards in a bundle (say as a gift!) In how many ways can we pick out a set of k cards to send our friend?

In this problem, we have n different kinds of postcards, and we want to find out how many ways to send k different cards to a given friend. At first glance, you might think that this is the same as the answer to our second puzzle: i.e. we have k slots, and we clearly have n choices for the first slot, $n - 1$ choices for the second slot, and so on/so forth until we have $n - (k - 1)$ choices for our last slot.

This would certainly seem to indicate that there are $\frac{n!}{(n-k)!}$ many ways to assign cards. However, our situation from before is not quite the same as the one we have now! In particular: notice that the order in which we pick our postcards to send to this one friend does not matter to our friend, as they will receive them all at once anyways! Therefore, our process above is **over-counting** the total number of ways to send out postcards: it would think that sending card X and Y is a different action to sending card Y and card X !

To fix this, we need to **correct** for our over-counting errors above. Notice that for any given set of k distinct cards, there are $k!$ different ways to order that set: this is because in ordering a set of k things, you make k choices for where to place the first element, $k - 1$ choices for where to place the 2nd element, and so on / so forth until you have just 1 choice for the k -th element.

Therefore, if we are looking at the collection of ordered length- k sequences of cards, each unordered sequence of k cards corresponds to $k!$ elements in this ordered sequence! That is, we have the following equality:

$$\begin{aligned} (\text{unordered ways to pick } k \text{ cards from } n \text{ choices}) \cdot k! &= (\text{ordered ways to pick } k \text{ cards from } n \text{ choices}) \\ &= \frac{n!}{(n-k)!} \end{aligned}$$

Therefore, if we want to only count the number of unordered ways to pick k cards from n choices, we can simply divide both sides of the above equation by $k!$, to get

$$(\text{unordered ways to pick } k \text{ cards from } n \text{ choices}) = \frac{n!}{k!(n-k)!}$$

This concept — given a set of k things, in how many ways can we pick m of them, if we don't care about the order in which we pick those elements — is an incredibly useful one, and as such leads itself to the following definition:

Definition. (Unordered choice without repetition.) The **binomial coefficient** $\binom{n}{k}$ is the number of ways to choose k things from n choices, if repeated choices are not allowed and the order of those choices does not matter.

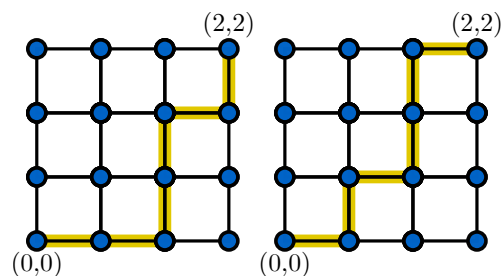
Observation. By the working above, we can see that $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ for any natural numbers k, n with $k \leq n$. (For $k > n$, we have $\binom{n}{k} = 0$ by the same reasoning as before: we cannot choose more than n distinct things from a set of n possibilities!)

This observation lets us solve two more of our problems:

Problem. What is the coefficient of x^8y^7 in $(x + y)^{15}$?

Solution. This might not seem like a choice-related problem at first. To make it a choice problem, though, let's think about a simpler case: $(x + y)^2$!

In particular, notice the following:



- Every term in $(x + y)(x + y)$ is generated by picking an element in the first $(x + y)$, picking an element in the second $(x + y)$, and then multiplying them together. That is: if we are FOILING the product of these polynomials together, we could either
 - pick the x from the first and the second,
 - the x from the first and the y from the second,
 - the y from the first and the x from the second, or
 - the y from the first and the second.

This leaves us with $x^2 + xy + xy + y^2$, which is correct!

You can do this in general to expand $(x + y)^n$ for any n : if you

expand $(x + y)^n$ into $\overbrace{(x + y)(x + y) \dots (x + y)}^{n \text{ times}}$, then we can calculate the product by just thinking of all of the ways in which these terms can combine! That is, every term in this product corresponds to exactly one of the following: either

- pick x from all n of the $(x + y)$ terms, or
- choose exactly $n - 1$ of the n different $(x + y)$ terms to pick a x from, and choose a y from the other $(x + y)$, or
- choose exactly $n - 2$ of the n different $(x + y)$ terms to pick a x from, and choose a y from the other two $(x + y)$'s, or
- ...
- choose exactly one of the n different $(x + y)$ terms to pick a x from, and choose a y from the other $n - 1$ $(x + y)$'s, or
- choose a y from every term.

Notice that doing this tells us that

$$(x + y)^n = \binom{n}{0}x^n + \binom{n}{1}x^{n-1}y + \binom{n}{2}x^{n-2}y^2 + \dots + \binom{n}{n-1}xy^{n-1} + \binom{n}{n}y^n.$$

We can simplify this with summation notation³⁷ to the following:

$$(x + y)^n = \sum_{i=0}^n \binom{n}{i} x^{n-i} y^i$$

In particular, in $(x + y)^{15}$, this tells us that the coefficient of x^8y^7 in $(x + y)^{15}$ is $\binom{15}{7} = \boxed{6435}$. Success!

Problem. A **lattice path** in the plane \mathbb{R}^2 is a path joining integer points via steps of length 1 either upward or rightward. For any two natural numbers $a, b \in \mathbb{N}$, how many lattice paths are there from $(0, 0)$ to (a, b) ?

Solution. Notice that any path from $(0, 0)$ to (a, b) will need to take $a + b$ steps. Of those $a + b$ steps, precisely a must be to the right and the remaining ones must be upward. Therefore, we can create any such path by just “choosing” a out of our $a + b$ steps to be the rightward steps! To be precise:

- Let s_1, s_2, \dots, s_{a+b} be $a + b$ placeholders for the steps we will take.
- Choose any set R of a of these steps, without any repeats or caring about the order in which R 's elements are chosen.

³⁷If you haven't seen this before: writing $\sum_{i=1}^n a_i$ is just shorthand for the

sum $a_1 + a_2 + \dots + a_n$. For example, $1 + 2 + 3 + \dots + n = \sum_{i=1}^n i$.

- Turn this into a path by making it so that we step right on any step $s_i \in R$, and step upwards if $s_i \notin R$.

There are $\binom{a+b}{a}$ many ways to choose such a set R , and therefore $\binom{a+b}{a}$ many paths from $(0,0)$ to (a,b) as well. Success!

Notice that if we had instead chosen the set U of moves to go “up” by, we would have gotten the answer $\binom{a+b}{b}$ to this problem instead. This might look worrisome (it looks like we got two different answers!), but is actually expected:

Claim. For any $n \geq k$, we have $\binom{n}{k} = \binom{n}{n-k}$.

Proof. Simply use our formula from above, which says that

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

□

As a result, $\binom{a+b}{a} = \binom{a+b}{(a+b)-a} = \binom{a+b}{b}$, as expected.

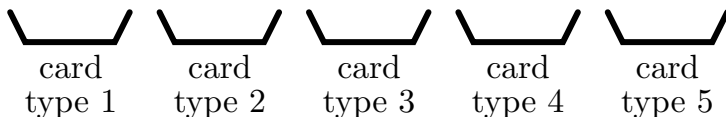
To illustrate one last variation on the multiplication principle, we return to our postcard problem:

Problem. Suppose that we are at the shops and want to buy a bunch of postcards to send out to our friends. The shop sells n different kinds of postcards, and has tons of each kind. We want to buy k cards (possibly with repetitions, if there’s a specific card design we like and want to send to many people.) In how many ways can we pick out a set of k cards to buy?

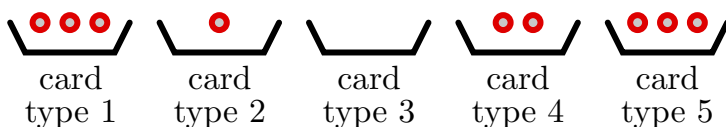
It first bears noting that this problem does not fall under the situations of our earlier problems. In this problem, our choices are unordered: i.e. we’re just picking out a bundle of cards to buy, and the order in which they’re bought is irrelevant. Therefore, we cannot use the “ordered choice with repetitions” observation we made earlier, as this would massively overcount things (i.e. we’d count orders of the same cards as different if the cashier rang things up in a different order, which is silly.)

However, unlike our two “ordered/unordered choice without repeats” situations, we can repeat choices! This means that this is not at all like those situations: in particular, k can be larger than n and we will still have lots of possibilities here, whereas in the “without repeats” situations this was always impossible. So we need a new method!

To develop this method, think of the n different kinds of postcards as n “bins.” Here’s a visualization for when $n = 5$:



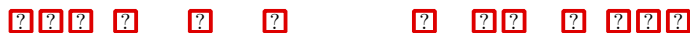
Picking out k cards to buy, then, can be thought of as pulling a few cards from the first bin, a few from the second, and so on/so forth until we’ve pulled out k cards in total. In other words, this is the **same problem** as distributing k balls amongst n bins:



To do *this* task, replace the n bins with $n - 1$ “dividers” between our choices. This separates our choices just as well as the bins did, so this is still the same problem.



Now, **forget the difference between objects and dividers!** That is, take the diagram above and suppose that you cannot tell the difference between an object and a divider between our choices.



How can we return this back to a way to choose k things from n choices? Well: take the set of $k + (n - 1)$ objects, of which k used to be things and $n - 1$ were dividers. Now choose $n - 1$ of them to be dividers! This returns us back to a way to pick out k things from n choices.



In particular, note that given any set of $k + (n - 1)$ placeholders, we can turn it into a way to choose k things from n choices with repetition by performing such a choice! Therefore, there are as many ways to make such choices as there are ways to choose $n - 1$ things from a set of $k + (n - 1)$ options to be placeholders. This second choice is unordered and without repetition (we want all of the placeholders to be different, and don't care about the order in which we pick the placeholders: just the elements that are chosen!) Therefore, we can use our “unordered choice without repetition” principle to see that there are $\binom{k+n-1}{n-1}$ many ways to do this!

In other words, we have the following observation:

Observation. (Unordered choice with repetition.) The number of ways to choose k things from n choices, where we do not care about the order in which we make our choices but allow choices to be repeated, is $\binom{k+n-1}{n-1}$.

We can use this to solve another one of our problems:

Problem. How many seven-digit phone numbers exist in which the digits are all nondecreasing?

Solution. By “nondecreasing” here, we just mean that each digit is at least as big to the digit to its left; i.e. 122-2559 is a valid phone number, but 321-1234 would not be.

With this understood, I claim that our problem can be reduced to an “unordered choice with repetition” task as follows: consider any way to choose seven numbers from the set of digits $\{0, 1, \dots, 9\}$, without caring about the order and with repetition allowed.

On one hand, I claim that any such choice can be turned into a nondecreasing phone number! Just list the digits here in order of their size; i.e. if you picked three 1's, a 2, a 3, and two 5's, write down 111-2355. This process also clearly generates any such phone number (just pick its digits!), and so the number of seven-digit nondecreasing phone numbers is just the number of unordered ways to choose seven things from the set of digits $\{0, 1, \dots, 9\}$ with repetition.

By our above formula, there are $\binom{7+10-1}{10-1} = \binom{16}{9} = 11440$ many such numbers. Success!

7.2 How to Count: Addition

Our “multiplication principle” is not the only tool we have for counting things. Consider the following common-sense idea for how to count the elements in a set:

Observation. (Summation principle.) Suppose that you have a set A that you can write as the **union** of several smaller disjoint³⁸ sets A_1, \dots, A_n .

Then the number of elements in A is just the summed number of elements in the A_i sets. If we let $|S|$ denote the number of elements in a set S , then we can express this in a formula:

$$|A| = |A_1| + |A_2| + \dots + |A_n|.$$

We can use this to solve our pizza problem from earlier:

Problem. Pizza My Heart is a great pizza place that I loved back when I was a lecturer in California. Suppose that I went there and they had the following deal on pizzas: for 13\$, they'd sell you a pizza with any two different vegetable toppings, or any one meat topping. If there are m meat choices and v vegetable choices, and for each pizza I also got to choose one of s sauces, how many different pizzas could I order under this deal?

Solution. Using the summation principle, we can break our pizzas into two types: pizzas with one meat topping, or pizzas with two vegetable toppings.

For the meat pizzas, we have $m \cdot s$ possible pizzas, by the multiplication principle (we pick one of m meats and one of s sauces.)

For the vegetable pizzas, we have $\binom{v}{2} \cdot s$ possible pizzas (we pick two different vegetables out of v vegetable choices, and the order doesn't matter in which we choose them; we also choose one of s sauces.)

Therefore, in total, we have $s \cdot (m + \binom{v}{2})$ possible pizzas!

We can also prove some interesting identities with this principle:

Problem. Demonstrate the following equality:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}.$$

Solution. While it is not particularly obvious, we can do this with the summation principle!

We do this as follows: take the set $\{1, \dots, n\}$, and consider all of the k -element subsets³⁹ of this set. On one hand, there are $\binom{n}{k}$ many such subsets — this is because there are precisely $\binom{n}{k}$ many ways to pick out k things from a set of n things if we don't care about the order in which we pick things, and that is precisely what we are doing when we are finding k -element subsets.

On the other hand, let's break our subsets of $\{1, \dots, n\}$ into two cases:

1. The subsets that contain the element n . How many such subsets are there? Well: to create any such subset, we have to pick the element n , and then we have to pick $k-1$ more elements out of a set of $n-1$ total possible objects (to fill in the rest of the set.) But this just means that there are $\binom{n-1}{k-1}$ many such sets!
2. The subsets that **do not** contain the element n . How many such subsets are there? Well: to create any such subset, we have to pick k elements out of a set of $n-1$ total possible objects (because we need k things that are not n .) But this just means that there are $\binom{n-1}{k}$ many such sets!

³⁸Sets are called **disjoint** if they have no elements in common. For example, $\{2\}$ and $\{\text{lemur}\}$ are disjoint, while $\{1, \alpha\}$ and $\{\alpha, \text{lemur}\}$ are not disjoint.

³⁹We say that a set A is a subset of a set B , and write $A \subseteq B$, if every element of A is an element of B . For example, $\{1, 2, \text{Batman}\}$ is a subset of $\{1, 2, 3, 4, 5, \text{Batman}, 7\}$.

By the rule of sum, because each subset of $\{1, \dots, n\}$ falls into exactly one of the two cases above, we can conclude that the total number of k -element-sized subsets of $\{1, \dots, n\}$ is just

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

Combining our two observations above gives us that

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}.$$

With these fundamental principles of counting at hand, we now turn to a pair of more subtle counting-based arguments: the **pigeonhole principle** and the **inclusion-exclusion principle**.

7.3 The Pigeonhole Principle

Proposition. (Pigeonhole principle): Suppose that $kn + 1$ pigeons are placed into n pigeonholes. Then some hole has at least $k + 1$ pigeons in it.

Proof. The most pigeons that can be put into n holes so that no hole has more than k pigeons is kn ; just put k pigeons in each hole. Therefore, if we have more than kn pigeons, at least one hole must contain at least $k + 1$ pigeons. \square

The applications of this property are where it really shines! We look at a few examples here:

Claim. Suppose that “friendship” is⁴⁰ a symmetric relation: i.e. that whenever a person A is friends with a person B , B is also friends with A . Also, suppose that you are never friends with yourself (i.e. that friendship is antireflexive.) Then, in any set S of greater than two people, there are at least two people with the same number of friends in S .

Proof. Let $|S| = n$. Then every person in S has between 0 and $n - 1$ friends in S . Also notice that we can never simultaneously have one person with 0 friends and one person with $n - 1$ friends at the same time, because if someone has $n - 1$ friends in S , they must be friends with everyone besides themselves.

Therefore, each person has at most $n - 1$ possible numbers of friends, and there are n people total: by the pigeonhole principle, there must be some pair of people whose friendship numbers are equal. \square

(This should remind you of problem 1(b) on assignment 3; it’s basically the same problem!)

Claim. Suppose that p is a prime number, and a is any number in $\{1, 2, \dots, p - 1\}$. Then there is some $b \in \{1, 2, \dots, p - 1\}$ such that $ab \equiv 1 \pmod{p}$.

Proof. Look at the $p - 1$ different products

$$1 \cdot a, 2 \cdot a, 3 \cdot a, \dots, (p - 1) \cdot a.$$

Specifically, look at their remainders after division by p .

Because no number in $\{1, 2, \dots, p - 1\}$ contains p as a factor, none of these products is a multiple of p ; therefore, all of these numbers

⁴⁰Magic!

are not congruent to 0 mod p , and their remainder on division by p is thus not equal to 0.

Assume for a contradiction that all of these products are also not congruent to 1 mod p ; i.e. their remainder on division by p is also not equal to 1.

Because every number has some remainder in $\{0, 1, 2, \dots, p-1\}$ on division by p , as proven in chapter 2, this means that we have $p-1$ different products that are distributed across $p-2$ different remainders (we originally had p options, but the 0 and 1 possibilities were eliminated.)

By the pigeonhole principle, some remainder must be repeated! That is, there are two distinct products ab, ac such that ab and ac have the same remainder on division by p ; that is, $ab \equiv ac \pmod{p}$.

But this means that by definition $ab - ac$ is a multiple of p . $b - c$ is a difference of numbers from $\{1, 2, \dots, p-1\}$; the largest this difference can be is $(p-1) - 1 = p-2$ and the smallest it can be is $1 - (p-1) = -(p-2)$, so the only multiple of p possible is if this is 0 (which it's not, because we said ab was distinct from ac .) But a is not a multiple of p as well, because $a \in \{1, 2, \dots, p-1\}$.

Therefore their product cannot be a multiple of p , and we have a contradiction to our assumption! That is, there must have been some b such that $ab \equiv 1 \pmod{p}$, as desired. \square

Claim. Take a sphere, and draw five different points on the sphere. Then there is some way to cut the sphere in half such that one half contains at least four of the points (if we consider points on the boundary of the hemisphere to count as “in” the sphere.)

Proof. Take any two points on the sphere. Cut the sphere in half through those two points.

By the pigeonhole principle, at least two of the remaining points must be on one half or the other half. That half now contains four points: the two on the boundary that we cut through, plus the two on that half! \square

Claim. Take any set of ten two-digit positive integers A . Then A has two disjoint subsets B, C such that the sum of the elements in B is equal to the sum of the elements in C .

This claim is maybe a bit abstract, so let's illustrate it with an example before proving it properly: suppose that our set A was

$$\{5, 9, 10, 17, 25, 31, 41, 72, 89, 93\}$$

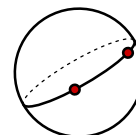
Then if we let $B = \{5, 10, 17, 25, 72\}$ and $C = \{9, 31, 89\}$, we have that the sum of elements in B is $5 + 10 + 17 + 25 + 72 = 129$ and the sum of elements in C is $9 + 31 + 89 = 129$, which are equal as desired!

This is not a proof, however; this is just an example. Let's prove that this happens for **every** set A :

Proof. Notice that there are $2^{10} = 1024$ subsets of A . This is because we can think of making a subset of A as deciding for each element in A whether to include it or not; this gives us 2 possibilities for each element, and thus 2^n total ways to make such a choice for a n -element set in general.

Now, notice that the largest sum that any subset B of A can be is if $B = A = \{90, 91, 92, 93, \dots, 99\}$, in which case the sum⁴¹ is

⁴¹A useful identity that we just used here is the fact that $1 + 2 + \dots + n = \frac{n(n+1)}{2}$. This is worth knowing!



$90 + \dots + 99 = 10 \cdot 90 + (1 + 2 + \dots + 9) = 900 + \frac{9 \cdot 10}{2} = 945$. Similarly, the smallest sum possible for any subset B is if $B = \emptyset$, in which case the sum is 0.

Think of the numbers $0, 1, \dots, 945$ as our pigeonholes, and the subsets of A as our pigeons. Because there are 1024 pigeons and 946 pigeonholes, there must be at least two sets A, B in the same pigeonhole; that is, there are two different sets A, B such that their sums are the same.

These sets A, B may have some overlap, however! To fix that, though, just delete the elements they have in common: i.e. replace A and B with $A \setminus (A \cap B)$ and $B \setminus (A \cap B)$. Because this removes the same elements from each set, it decreases the sum of each set by the same amount, and does not change the property that the sum of the elements in each of these sets are equal. \square

7.4 The Inclusion-Exclusion Principle

We motivate our last counting principle by considering the last problem from our introduction:

Problem. How many positive integers between 1 and 10,000 are not divisible by 2, 3 or 5?

Solution. Directly solving this by writing down all of the numbers from 1 to 10,000 sounds like... not a great plan. Instead, let's try a more indirect approach! That is: notice the following:

- It is easy to count all of the integers between 1 and 10,000 that are multiples of 2; these are just the even numbers, and there are $\frac{10000}{2} = 5000$ of these.
- Similarly, it is easy to count all of the integers that are multiples of 3; there⁴² are $\lfloor \frac{10000}{3} \rfloor = 3333$ of these integers.
- As well, there are also $\frac{10000}{5} = 2000$ integers that are multiples of 5.

At first, it's tempting to say that we can just count all of the integers between 1 and 10,000 by taking the 10000 possible values and subtracting the count of the multiples of 2, 3 and 5; this would give us $10000 - 5000 - 3333 - 2000 = -333$ many such numbers. ...um. That's not right. What went wrong?

Well: the main error we made here is that we **overcounted** our numbers! That is: when we subtracted off the multiples of 2 and the multiples of 3, the multiples of 6 were counted **twice**. Therefore, we subtracted these numbers **twice** (and similarly for the multiples of 10 and 15,) which is why we got a ridiculous answer.

To correct for this, we need to fix our mistake! That seems easy enough to do: we just need to undo the double-subtraction of the multiples of 6, 10 and 15 by adding these back in! There are $\lfloor \frac{10000}{6} \rfloor = 1666$, $\lfloor \frac{10000}{10} \rfloor = 1000$ and $\lfloor \frac{10000}{15} \rfloor = 666$ many such values, which suggests an answer of $10000 - 5000 - 3333 - 2000 + 1666 + 1000 + 666 = 2999$ many such values.

...except if you calculate this on a computer, you'll get 2666 many such numbers. What went wrong?

Well: when we added back in these multiples of 6, 10 and 15, numbers that were in the overlap of these sets (namely, multiples of 30) were added back in too many times! To be precise, any number that was a multiple of 30 has gone through the following process:

⁴²Note: we write $\lfloor x \rfloor$ to denote the "round down" function: i.e. $\lfloor \pi \rfloor = 3$, $\lfloor 4 \rfloor = 4$ and $\lfloor -0.9 \rfloor = -1$.

- We subtracted it out three times, as it was a multiple of 2, 3 and also 5 individually,
- We then added it back in three times, as it was a multiple of 6, 10 and 15 individually.

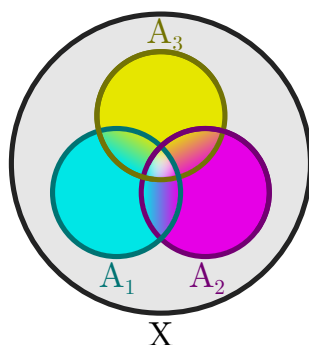
To get the right count, we need to subtract this back out again! That is, we need to subtract off $\lfloor \frac{10000}{30} \rfloor = 333$ to get to the right answer. Doing this to our second mistaken answer 2999 gives us the correct answer of 2666, as desired.

The idea here, roughly speaking, was the following:

- We wanted to calculate the size of $X \setminus (A_1 \cup A_2 \cup \dots A_k)$, for some set X and subsets $A_1, A_2 \dots A_k$. Here, in particular, we wanted to count all of the numbers from 1 to 10,000 (our set X) that aren't multiples of 2 (our set A_1), 3 (our set A_2) or 5 (our set A_3 .)
- Doing this directly was hard!
- Instead, we calculated the size of the sets $A_1, A_2, A_3, A_1 \cap A_2, A_1 \cap A_3$ and $A_2 \cap A_3$.
- Then, we used the following⁴³ observation:

$$|X \setminus (A_1 \cup A_2 \cup A_3)| = |X| - |A_1| - |A_2| - |A_3| + |A_1 \cap A_2| + |A_1 \cap A_3| + |A_2 \cap A_3| - |A_1 \cap A_2 \cap A_3|.$$

While this formula might look scary, if you represent your sets with a Venn diagram it should make things clearer. Consider the drawing below, that depicts a set X containing three sets A_1, A_2, A_3 with a Venn diagram:



If we wanted to count all of the elements in X that aren't in either of the sets A_1, A_2, A_3 , the above process says we'd do this as follows:

- First, count all of the elements in X .
- Now, count all of the elements in each of A_1, A_2, A_3 , and subtract those counts from the number we counted earlier.
- Notice that when you did this, the elements in $A_1 \cap A_2, A_1 \cap A_3$ and $A_2 \cap A_3$ were deleted “twice:” that is, when we deleted the blue elements and then deleted the yellow elements, the blue-yellow elements were removed twice (and similarly for the other colors!)
- To fix this, we need to add back in another copy of these deleted elements. That is: count the number of elements in each of $A_1 \cap A_2, A_1 \cap A_3$ and $A_2 \cap A_3$, and add those counts back into our sum.

⁴³Given a set X , we use the notation $|X|$ to denote the number of elements in X . For example, $|\{1, 4, 5\}| = 3$.

- Almost done! The only issue here is that the elements in $A_1 \cap A_2 \cap A_3$ (the rainbow elements in the middle) have now been added back in “too many times:” we counted them four times (once when we counted X , and once each when we counted $A_1 \cap A_2$, $A_1 \cap A_3$ and $A_2 \cap A_3$), and only deleted them three times (when we subtracted $|A_1|$, $|A_2|$, $|A_3|$.)

We want to have these elements not counted at all (we want to count $X \setminus (A_1 \cup A_2 \cup A_3)$), so we need to take away one more copy of these elements: namely $|A_1 \cap A_2 \cap A_3|$!

This fact let us calculate the answer to our problem!

In general, this is called the **inclusion-exclusion principle**:

Claim. Given a finite set X , along with n subsets A_1, A_2, \dots, A_n of X , we have

$$\left| X \setminus \left(\bigcup_{k=1}^n A_k \right) \right| = |X| + \sum_{k=1}^n \left((-1)^k \cdot \sum_{\substack{\text{all unordered subsets} \\ \{i_1, i_2, \dots, i_k\} \\ \text{of size } k \text{ of } \{1, \dots, n\}}} |A_{i_1} \cap \dots \cap A_{i_k}| \right).$$

First, notice that this somewhat scary-looking sum is indeed the same thing as our result from earlier: if $n = 3$, the sum at right is just

$$|X| + (-1)(|A_1| + |A_2| + |A_3|) + (-1)^2(|A_1 \cap A_2| + |A_1 \cap A_3| + |A_2 \cap A_3|) + (-1)^3(|A_1 \cap A_2 \cap A_3|)$$

Proof. To see that this counting process actually works, take any $x \in X$, and consider cases:

1. x is not in $\bigcup_{k=1}^n A_k$. In this case, then x is “counted” once in our big sum: it contributes 1 to the $|X|$ count, and never shows up in any of the other terms, as it is not in any of A, B, C or their intersections.
2. x is in l of the A_i sets: to be specific, let’s say that $x \in A_{i_1}, \dots, A_{i_l}$ for some indices $i_1 < \dots < i_l$, and that x is not in any of the other A_j sets.

In this setting, then x shows up in the following:

- x contributes a 1 to the $|X|$ term’s size.
- x is in l of the A_i terms, and thus contributes $(-1) \cdot l$ from the $\sum_{1 \leq i_1 \leq n} |A_{i_1}|$ terms.
- x is in $\binom{l}{2}$ of the $A_i \cap A_j$ terms, because it’s in l sets total and there are $\binom{l}{2}$ many ways to pick out pairs of these sets to intersect. Therefore, it contributes $(+1) \cdot \binom{l}{2}$ from the $\sum_{1 \leq i_1 < i_2 \leq n} |A_{i_1} \cap A_{i_2}|$ terms.
- Similarly, x is in $\binom{l}{3}$ of the $A_i \cap A_j \cap A_k$ terms, because it’s in l sets total and there are $\binom{l}{3}$ many ways to pick out triples of these sets to intersect. Therefore, it contributes $(-1) \cdot \binom{l}{3}$ from the $\sum_{1 \leq i_1 < i_2 < i_3 \leq n} |A_{i_1} \cap A_{i_2} \cap A_{i_3}|$ terms.
- ...
- In general, for any $k \leq l$, x is in $\binom{l}{k}$ of the $A_{i_1} \cap \dots \cap A_{i_k}$ terms, as it is in l sets total and there are $\binom{l}{k}$

many ways to pick out k sets from the l sets in total. Therefore, in total, it contributes $(-1)^k \cdot \binom{l}{k}$ from the $(-1)^k \cdot \sum_{1 \leq i_1 < \dots < i_k \leq n} |A_{i_1} \cap \dots \cap A_{i_k}|$ terms.

So, in total, x contributes

$$1 - l + \binom{l}{2} - \binom{l}{3} + \dots + (-1)^k \binom{l}{k} \\ = \sum_{k=0}^l \binom{l}{k} (-1)^k$$

to the right-hand side.

If we remember the binomial theorem, which says that $(x + y)^l = \sum_{k=0}^l \binom{l}{k} x^k y^{l-k}$, and plug in $x = -1, y = 1$, we get

$$(-1 + 1)^l = \sum_{k=0}^l \binom{l}{k} (-1)^k.$$

But $(-1 + 1)^l = 0$; so we actually have that x contributes 0 to the sum!

So, our sum adds up a 1 for every $x \in X \setminus (\bigcup_{k=1}^n A_k)$ and a 0 for every other x ; in other words, the right hand side gives us the size of $X \setminus (\bigcup_{k=1}^n A_k)$, as claimed! \square

Roughly speaking, our strategy here was the following:

- To find the size of the union of a bunch of sets, we first added up the size of all of these sets; this overcounts, however, because elements in more than one set get counted multiple times!
- To correct for this overcount, we then “fix” things by subtracting off the size of all of our pairwise intersections. But this results in an undercount, as elements in more than two sets get counted by these pairwise intersections too many times!
- To correct for **this** undercount, we then add in the size of all of our triple-intersections; but this too results in an overcount . . .
- But if we keep doing this, at the end we will have (magically) counted everything correctly!

This idea is an incredibly useful one — to get an exact count, we can just use “at least” counts and repeatedly add and subtract off errors to get the right result at the end!

We use this to solve one last problem:

Problem. Suppose that you’re at a graduation with n people, each of which has their own cap. Suppose that everyone throws their cap in the air when they graduate, and that the caps are randomly handed back to everyone.

What is the probability that **no one** gets their own cap back?

Solution. There are n people and n caps, and therefore $n!$ ways for people to be assigned to caps: this is choosing n things from n choices, where repeats aren’t allowed but we care about order.

So, if we can just count all of the ways in which everyone can get a cap back that is not their own, we’ll have an answer here!

To do this, let A_i denote the set of all ways to match people to caps such that person i gets their own cap back. If we do this, then we can make the following observations:

- $n! - |A_1 \cup A_2 \cup \dots A_n|$ is what we want to count: we want all of the ways to assign people to caps minus all of the ways in which someone can get their own cap back.
- $|A_{i_1} \cap A_{i_2} \cap \dots A_{i_k}|$ is just the number of permutations that makes sure that people $i_1, \dots i_k$ all get their own cap back.
- $|A_{i_1} \cap A_{i_2} \cap \dots A_{i_k}|$ is just $(n - k)!$; we just assign people $i_1, \dots i_k$ their own cap, and choose any way to assign the remaining $n - k$ people the remaining $n - k$ caps.
- There are $\binom{n}{k}$ ways to choose unordered value $i_1, \dots i_k$ from $\{1, 2, \dots n\}$.

Therefore, the inclusion-exclusion formula says that there are

$$\begin{aligned}
 & n! - \sum_{i=1}^n |A_i| + \sum_{\substack{\text{all unordered subsets} \\ \{i_1, i_2\} \text{ of } \{1, \dots n\}}} |A_{i_1} \cap A_{i_2}| + \dots + |A_1 \cap \dots \cap A_n| \\
 &= n! - \binom{n}{1}(n-1)! + \binom{n}{2}(n-2)! + \dots + (-1)^n \binom{n}{n}(n-n)! \\
 &= n! - \frac{n!}{1!(n-1)!}(n-1)! + \frac{n!}{2!(n-2)!}(n-2)! + \dots + (-1)^n \frac{n!}{n!0!}0! \\
 &= n! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + (-1)^n \frac{1}{n!} \right) \\
 &= n! \sum_{k=0}^n \frac{(-1)^k}{k!}
 \end{aligned}$$

many ways to hand back caps without anyone getting their own cap back.

Expressed as a ratio, this tells us that the probability of choosing such a way is just the above quantity divided by $n!$, i.e.

$$\boxed{\sum_{k=0}^n \frac{(-1)^k}{k!}}.$$

People with some calculus knowledge will recognize that $\sum_{k=0}^{\infty} \frac{(x)^k}{k!} = e^x$, and therefore that for large values of n this probability is just $\frac{1}{e}$. Neat!

Chapter 8: Automata

Week 11

UoA 2018

In this section, we combine the skills we developed on graph theory and proofs here to study a simple form of computation: namely, that of a **deterministic finite-state automata!**

To do this, we first start off by defining some preliminary notation:

8.1 Strings, Languages, and Automata

Definition. A **alphabet** is just any set. Alphabets usually consist of single characters, like $\{a\}$ or $\{a, b, c\}$ or $\{0, 1, \dots, 9\}$, and are usually denoted by the symbol Σ ; for example, a common phrase you'll see when working with automata is "Let $\sigma = \{a, b\}$ be an alphabet."

Given an alphabet Σ , a **word** or **string** of length n over Σ is any ordered list of n symbols from σ . For example, 011 and 101 are both length-3 strings over the alphabet $\{0, 1\}$, $aaaa$ is the only length-4 string over the alphabet $\{a\}$, and $ababab, abcabc$ are two length-6 strings over the alphabet $\{a, b, c\}$. Words are often denoted by writing lower-case letters: i.e. v, u, w are common variables used for words, much like how x, y are common variables used for real numbers.

Often, it is useful to be able to discuss the length-0 string that contains no symbols. We denote this string by writing λ !

Given this notion of alphabets and words, we say that the **set of all finite length words** Σ^* is the collection of all finite-length words in the alphabet Σ : i.e.

$$\Sigma^* = \{a_0 a_1 \dots a_{n-1} \mid n \in \mathbb{N}, a_0 \dots a_{n-1} \in \Sigma\}$$

Finally, we define a **language** to be any subset of Σ^* itself. In other words, a language is just a collection of things we consider to be words (which should make intuitive sense!) For example, the following things are languages over the alphabet $\{a, b\}$:

- $\{a, b, ab, aba\}$
- $\{\overbrace{abab \dots ab}^{n \text{ times}} \mid n \in \mathbb{N}\}$
- Σ^*
- $\{w \mid w \in \Sigma^*, w \text{ contains a prime number of } a's\}$

Languages are often denoted with capital letters, like L .

Given an alphabet Σ and two strings $v, w \in \Sigma^*$ we can define a pair of operations on how to compare and combine v, w :

- Given two strings v, w , we can define the **concatentation** of v and w as the string that is formed by first writing down v and then w in that order. For example, the concatentation of 01010 and 111 is just 01010111, and the concatentation of aaa and λ , the empty string, is aaa .
- Given two strings v, w , we say that v is a **substring**⁴⁴ of w if we can delete some symbols from the start and end of w to get the string v . For example, the following four strings are each substrings of $abbaaaa$:

⁴⁴Good exercise: show that "is a substring of" is a poset relation on any language!

Notice that the empty string λ is a substring of any string w ; just delete all of w , and you'll have the empty string left over!

Strings and languages, in of themselves, are not particularly interesting. However, the notion of an **automaton** is where things get more compelling:

Definition. Take an alphabet Σ . A **deterministic finite-state automaton** over the alphabet Σ consists of four things:

- A finite set of states S .
- A **transition function** $T : S \times \Sigma \rightarrow S$. Recall that this notation is just a fancy way of saying that the transition function T takes in as its **inputs** a state $q \in S$ and a letter $\sigma \in \Sigma$, and uses this to **output** another state $q' \in S$.
- A subset $F \subseteq S$ of **accepting states**.
- An element $q_0 \in S$, that we think of as the **initial state**.

Much like how graphs had a very abstract definition that was made significantly better by visualizing things, deterministic finite-state automata can be made much clearer with diagrams! In particular, we can visualize any deterministic finite-state automaton as a *directed multigraph with edge labels from Σ* as follows:

- First, make a vertex for every state in S .
- Now, draw an edge from vertex q to vertex q' labeled with the letter σ if $T(q, \sigma) = q'$. This lets us easily visualize the function T : to see where T sends any state-letter pair (q, σ) , just find the vertex corresponding to state q and look for the edge leaving q labeled σ !

Note that this edge can be a self-loop if $T(q, \sigma) = q$.

- Finally, label the starting vertex as “start” and the accepting vertices as “accepting.” Many people indicate the starting vertex by drawing a partial edge pointing to the starting vertex, and by shading in the accepting vertices.

Notice that in any such visualization, every vertex has precisely $|\Sigma|$ edges leaving it: one for every element of Σ . This is because T is a function, and so must be defined for every state q and letter σ !

We give an example of an automaton and its associated visualization below:

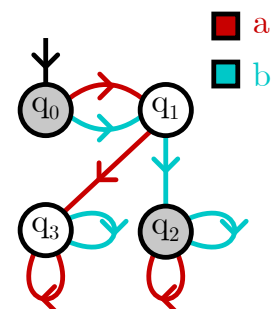
Example. Here's an automaton M over the language $\{a, b\}$:

- States: we have four states, q_0, q_1, q_2 and q_3 . We set q_0 to be the **initial** state, and q_0, q_2 as its **accepting states**.
- We define our transition function T by cases:

$$\begin{array}{ll} - T(q_0, a) = T(q_0, b) = q_1. & - T(q_2, a) = T(q_2, b) = q_2. \\ - T(q_1, a) = q_3, T(q_1, b) = q_2. & - T(q_3, a) = T(q_3, b) = q_3. \end{array}$$

Just like with graphs, we will typically just give the visualization for an automaton instead of its formal definition, as the picture at the right is typically much easier to understand!

Given an automaton, the main thing we will want to do is **run** it on a string w ! We do this as follows:



Definition. Given an automaton M over an alphabet Σ and a string $w = \sigma_0\sigma_1 \dots \sigma_{n-1}$ in Σ^* , we can **run** M on w by doing the following:

- Start at q_0 , the initial state.
- Travel from q_0 to whichever state is connected to q_0 by an edge labeled σ_0 .
- Then, travel from that state by using whichever edge is labeled σ_1 .
- Then, travel from that state by using whichever edge is labeled σ_2 .
- ...repeat this process, until you've ran out of letters in w !

For example, if we ran our automaton M from before on the word *abba*, we'd do the following:

- We'd start at q_0 .
- Then, our rules say we'd take the *a*-edge to go to q_1 .
- Then our rules would tell us to take the *b*-edge to go to q_2 .
- From here, our rules will tell us to return to q_2 no matter whether we take the *a* or *b* edges (i.e. we're stuck!)

We say that an automaton M **accepts** a word w if when we run M on w , we end on one of our accepting states; otherwise, we say that M **rejects** w .

Given an automaton M , a common problem that we'll try to solve is the following: what is the language of all strings that M will accept? In other words, what's the collection of all inputs to our automaton that will end in an accepting state?

If you think of our automaton as modeling a very simple notion of modelling what a computer program can do (i.e. it takes in inputs, uses those inputs to shuffle between states, and uses this to output either "accept" or "reject") this is a very natural question to ask. In computer programming, we often want to determine the behavior of a program: i.e we'll want to figure out if a given program can recognize all strings that look like valid ID numbers, or all numbers that are primes, or all orders that can be fulfilled through the Auckland branch of your company.

So, let's try to answer this problem for a few simple automata!

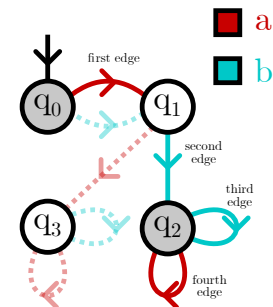
Problem. Let M be the automaton we defined before. What is the language of all strings that M accepts?

Solution. First off, we note that q_0 is an accepting state, so we accept λ , the empty string.

Secondly, we can notice that if we run M on any string w with at least one letter, after reading off our first character we're at q_1 , a nonaccepting state. So M does not accept either of the one-letter words *a* or *b*.

Now, notice that if our second letter is *a*, we go to q_3 . q_3 is something that we call an "absorbing" state: i.e. all of the edges connected to q_3 are self-loops, and so once we go to this vertex we stay here forever. Therefore, any string whose second letter is *a* will go to q_3 and stay there! Because q_3 is a nonaccepting state, this means that we do not accept any string whose second letter is an *a*.

Finally, notice that if our second letter is a *b*, we go to q_2 . This is another "absorbing" state, but it is also an accepting state: therefore we can conclude that we **accept** any string whose second letter is a *b*.



In conclusion, then, we have that M accepts the language L of all strings that are either empty or contain at least two letters, with a b as their second letter. Success!

Problem. Let M be the automaton over the language $\{0, 1, 2\}$ defined at right. Completely describe the language of all strings accepted by M .

Solution. Notice that the letter 0 never changes the state we're in; on every vertex, the edge labeled 0 returns to itself. So we can safely "ignore" those symbols; that is, if we take any word w and delete all of its 0 letters, this won't change the behavior of our automaton.

As well, notice that the symbols 1 and 2 "undo" each other; that is, every edge labeled 1 going in one direction is paired with an edge labeled 2 going in the other direction. Therefore, if we ever have a 1 followed by a 2, or vice-versa, these symbols will "cancel out;" in other words, if we take any word w and one-by-one delete "12" pairs, and then delete "21" pairs, the behavior of our automaton will still not change.

As a result, we can reduce any word to just a string of 1's or a string of 2's. Finally, notice that in our automaton, repeating 1 three times just returns us to where we start, because this forms a cycle; similarly, repeating 2 three times does the same thing. Therefore, we can just delete any triple 1's or triple 2's and still not change our automaton's behavior!

This leaves us with just a handful of options: after the reductions given above, we must be one of $\{\lambda, 1, 2, 11, 22\}$. Checking by hand will show you that we accept precisely the string 1 and 22 from that list, and reject the others. So, we accept the language of all strings that can be reduced via the above process to just a 1 or two 2's!

This is certainly an acceptable answer! However, it is worth noting that we can find a shorter description of our language if we examine our reductions. Notice that our reductions can be collectively described as "if a collection of symbols in your string sum to 3, get rid of them." Therefore, at the end, we've reduced our string to just its sum modulo 3! From looking at the diagram, we accept precisely those strings whose sum is 1 modulo 3, and reject all others. This is a nice description of our language to have!

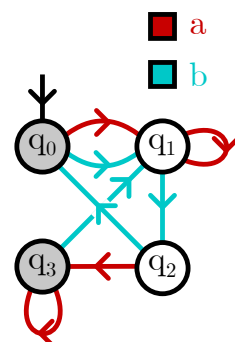
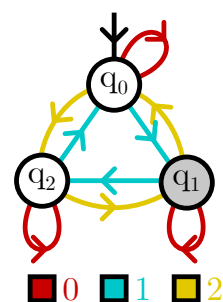
Not every automaton accepts such a clean description of its accepting states! Consider the following automaton at right, generated by you all in class on Friday. What strings does this automata accept?

Solution. This is a lot messier than our earlier automata. I claim that it's probably easier to solve this problem by describing all strings that our automaton does not accept. To do so, let's consider a number of smaller problems:

- Suppose that we started at q_1 , and chose to ignore the q_3 state entirely (i.e. we pretend it's been deleted.) What kinds of strings would let us return to q_1 ?

Well, by looking at our automaton, we can see that the following strings completely describe all of the ways to stay at q_1 :

1. We have any string of a 's of any length: $\overbrace{aaa \dots a}^n$, for $n \geq 0$.
2. We have the above string followed by a $bb\star$, where \star denotes a wildcard representing any one character (i.e. either a or b .)



3. We have any number of strings of the type listed in (2) repeated after each other.

Call such words γ -words, for shorthand.

- Now, let's consider all of the ways to start at q_1 and stay at q_1 if we're allowed to also consider the q_3 state! In this world, we've got the following possibilities:

1. We still have the γ -words from before.
2. We also have any γ -word followed by a $b \overbrace{aaa \dots a}^{m \text{ } a's} b$, where m is any integer that's at least 1, as that's the only path that uses the q_3 state once.
3. We have any number of strings of the type listed in (2) repeated after each other.

Call such words δ -words, again for shorthand.

- Now, consider all ways of starting at q_0 and ending at q_1 ! Because any single-character word \star goes from q_0 to q_1 , this is just the collection of all words of the form $\star w_\delta$, where w_δ denotes an arbitrary δ -word as defined above.
- Now, consider all ways of starting at q_0 and ending at q_2 ; this is just any word of the form $\star w_\delta b$.

This completely characterizes all of the strings we do not accept! That is: let Σ^* denotes all of the possible finite strings on the alphabet $\{a, b\}$, and let $D = \{v \in \Sigma^* \mid v = \star w_\delta \text{ or } v = \star w_\delta b, \text{ for some } \delta\text{-word } w_\delta\}$.

Then our automata accepts precisely all words of the form $\Sigma^* \setminus D$.

Not the prettiest solution, but that's OK! Not every problem has a pretty solution. The point is that it *is* a solution, and we found it by reasoning our way through our automaton's behavior.

Another common kind of question we'll encounter is trying to "reverse" this process: that is, given some language L , we'll want to design an automaton M that accepts precisely L . This can be a little tricky to do in practice, but we can describe a rough framework that will usually get the job done.

To do this, we first introduce a useful definition:

Definition. We say that an automaton M **cannot distinguish** between two strings w_1, w_2 if when M is ran on the string w_1 , it winds up in the same state as when it runs on the string w_2 .

For example, our second automaton could not distinguish between strings whose sums were the same modulo 3; any string whose sum was 0 modulo 3 wound up at q_0 , any string whose sum was 1 modulo 3 wound up at q_1 , and any string whose sum was 2 modulo 3 wound up at q_2 .

Conversely, our first automaton could distinguish between the empty string and all other strings; this is because after we read our first symbol we left q_0 , and no edges let us return to q_0 .

With this definition at hand, here's a strategy for designing an automaton to recognize a language L :

1. Start by drawing a single initial vertex q_0 .
2. Let a be the first letter in the alphabet for our language L . Consider the following question: does our language L "care" about whether a word starts with a ? That is:
 - The language $\{aba, aaa\}$ definitely cares if a word starts with a ; it can distinguish between the words aba (which it accepts) and ba (which it rejects.)

- Similarly, the language $\{\overbrace{ababab \dots ab}^{n \text{ pairs}} \mid n \in \mathbb{N}\}$ cares if a word starts with the letter a , as it can tell the difference between words like $ab, abab, ababab, \dots$ (which it accepts) and words like b, ba, bbb, \dots (all of which it rejects).
- However, the language $\{w \in \Sigma^* \mid w \text{ contains exactly one } b\}$ does **not** care about whether a word starts with a or not! All it cares about is whether the word has a b in it or not.

If our language cares about whether a word starts with a or not, we will need to build an automata that can distinguish λ from a . If so, we will need to have an edge leaving q_0 to some new state; create such a state and edge. If not, however, then our automata does not need to create a new state; add a self-loop from q_0 to itself labeled a , because we want a machine M that cannot distinguish between λ and a .

3. Do this for every letter in our alphabet! This completes all of the edges that we need to draw leaving q_0 .
4. Go on to any one of the new states we've made; call it q_1 . For this state and for any letter σ in our alphabet, consider all of the possible ways to connect q_1 to any other state q_2 by an edge labeled σ .

For each such way of making a connection, we are in essence telling our automaton that any word that leads us to q_1 , followed by a σ , should be indistinguishable from any word that leaves us at q_2 .

If this is ever true, create such an edge! However, if it is never true and our language demands that we can distinguish between a q_1 word followed by a σ and all other words that lead us to other vertices, create a new state and connect q_1 to that state by a σ edge.

5. Repeat step (4) until q_1 now has edges leaving it for every letter in our alphabet.
6. Now, repeat steps (4+5) until we are out of new states! This leaves us with an automaton M that distinguishes precisely between the words that L cares about; if we simply mark all of the states in M with "accepting" when the corresponding words are in L , we're done!

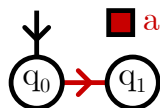
This is pretty wordy for a relatively straightforward process, so we give a few examples here to illustrate how this goes in practice:

Problem. Let L be the language over the alphabet $\{a\}$ consisting precisely of the strings $\{a, aaa\}$. What is an automaton whose set of accepting strings is precisely L ?

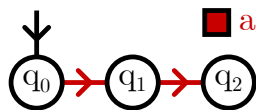
Solution. At first, we draw our initial state:



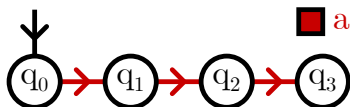
Our language cares about whether a word starts with a , as it can tell the difference between a and λ . So we add a new state:



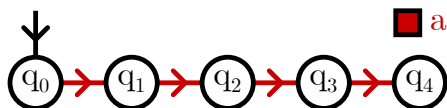
Our language also cares about the difference between a (which it accepts) and aa (which it rejects,) so we do not add a self-loop from a to itself. We also do not add an edge back to q_0 , as this would tell us that we think that λ and aa are indistinguishable (and thereby think that aaa and $aaaaa$ are indistinguishable, which is wrong.) So we add an edge to a new state:



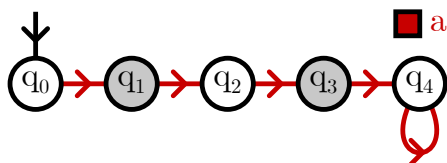
Once again, we cannot add an edge back to any of our existing states, as this would create a loop and likely force us to accept infinitely many words, when we want just the words a and aaa ! So we make another new state:



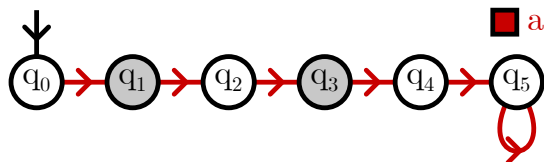
Again, we cannot add an edge back to any earlier states as this would tell us that aaa (which is the longest string we accept) is equivalent to some earlier string, and thereby generate even longer / infinitely many strings that we accept. So we add one last state:



From this state, we can safely add a self-loop: our language does not care about the number of a 's in a word once they exceed 3, as we reject all of these words. Shading in q_1 and q_3 to indicate that they're the states we accept (as we accept precisely the words a , aaa) then finishes our construction:



Notice that there are multiple different automata that can generate the same language: for instance, the automaton below will also accept precisely $\{a, aaa\}$!



This is because it is functionally the same as our automaton as before, except we've just made it take a bit longer to get to the "absorbing state" self-loop.

To illustrate this process further, we calculate one last example:

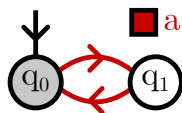
Problem. Let L be the language over the alphabet $\{a, b\}$ consisting precisely of the strings

$$\{w \mid w \text{ contains an even number of } a\text{'s and exactly one } b\}.$$

What is an automaton whose set of accepting strings is precisely L ?

Solution. While the case-by-case process would work here as well, let's be a bit more clever this time!

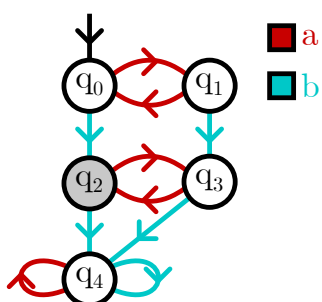
Specifically: notice that something like



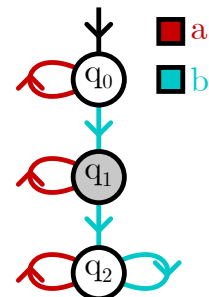
is going to be great for detecting whether we have an even number of a 's in it, as it just bounces back and forth, landing on the gray vertex for any even number of a 's.

Similarly, we can borrow an idea from the automaton we designed earlier and notice that something like the automata at right should be great at detecting whether or not we have exactly one b : the red self-loops mean that we ignore a 's, while the blue edges ensure that we only accept a word with one b and reject any with none or more than one.

With some cleverness, we can “combine” these into a single automata: behold!



This automata clearly rejects any word with no b 's (as we never get to the middle level) or with more than one b (as we are sent to the bottom level, which is an absorbing state.) As well, it keeps track of whether we've seen an even or odd number of a 's so far, and only accepts strings with an even number. Success!



Chapter 9: Codes

Week 12

UoA 2018

To open our class, we're going to study the following problem:

9.1 A Motivational Example

Problem. Suppose that you are the Voyager 1 probe. You are currently on the outer limits of the solar system, and about to leave the solar system forever! Consequently, you want to call your family to say goodbye. However, you are currently separated from your parents by the vast interstellar void of SPAAAAAAACE.

The vast interstellar void of space has an annoying habit of occasionally containing stray electromagnetic waves that interfere with your communications back home; when you send a signal back home (in binary, naturally), occasionally one of your 1's will be switched into a 0, or vice-versa. Assume that no more than one out of three consecutive bits in any message you send will be scrambled.

How can you call home?

One solution to this problem you might come up with is to simply "build redundancy" into the signals you send home, by sending (say) six 1's every time you want to send a 1, and six 0's every time you want to send a 0. For example, to send the message "0101," you'd send

000000 111111 000000 111111.

Then, even if some of the bits are flipped, your parents back on earth could still decode your message. In particular, if at most one bit out of any consecutive three is flipped, each of your all-0's blocks will have at most two 1's in them after errors are introduced, while each of your all-1's blocks will have at most two 0's in them after errors. In either case, we would never confuse one of these blocks with the other: if your parents received the signal

010010 001111 100001 101011,

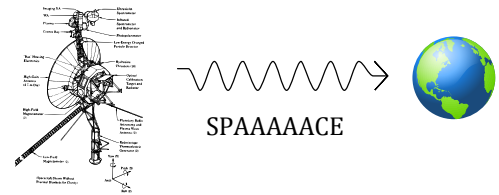
they could "correct" the errors above to get the signal

000000 111111 000000 111111,

because we're assuming that there's at most two errors in each block! This means that they've successfully received the signal 0101 that you sent back home.

This code can correct for the presence of one error out of any three consecutive blocks, but no better. That is, if we could have more than two errors in a block of six, we might have three errors in a string of six: in this case it would be impossible to tell what our string was intended to be. For example, the string 000111 could have resulted from three errors on the signal 000000, or three errors on the signal 111111. The cost for this "error-correction" ability is that we have to send $6k$ bits whenever we want to transmit k bits of information.

Can we do better? In specific, can we make a code that is more efficient (i.e. needs less bits to transmit the same information,) or can correct for more errors? With a little thought, it's easy to improve our code above: if we instead simply replace each 0 with just 000 and each 1 with 111, our code can still correct for the presence of at most one error in any three consecutive blocks (for



example, 101 is unambiguously the result of one error to 111,) and now needs to send just $3k$ bits to transmit k bits of information.

There are more interesting codes than just these repetition codes, though! Consider, for example, the codeword table at right.

In this code, we encode messages by breaking them into groups of three, and then replacing each string of three with the corresponding group of six. For example, the message “010 101 111” would become

010101 101101 111000

In this code, every word in the table above differs from any other word in at least three spots (check this!) Therefore, if we have at most 1 error in any six consecutive bits, we would never confuse a word here with any other word: changing at most one bit in any block of six would still make it completely unambiguous what word we started with.

Therefore, if we sent the string that we described above, and people on Earth received

010111 101111 111000,

they would look through our codeword table for what words these strings of six could possibly be, if at most one error in every six consecutive bits could occur.

The first signal 010111 isn’t in our table, so we can’t immediately tell what 010111 was supposed to mean. However, 010111 differs from each of the words $\{000000, 100011, 001110, 011011, 101101, 110110, 111000\}$ in at least two spots: so if only one error could have crept in, 010111 can’t be any of these strings! So the only string it **could** be is the one that differs from it in at most one place: 010101.

Similarly, we can look at 101111; this again isn’t in our table, but we can look through the table and see that there’s only one word that differs from it in at most one place: 101101. So that must have been what was originally sent!

Finally, 111000 is in our table, so we don’t have to correct anything here. So, our corrected message is

010101 101101 111000.

We can translate this message back to “010 101 111,” which is the message we sent.

This code can correct for at most one error in any six consecutive bits (worse than our earlier code,) but does so much more efficiently: it only needs to send $2k$ bits to transmit a signal with k bits of information in it.

So: suppose we know ahead of time the maximum number of errors in any consecutive string of symbols. What is the most efficient code we can make to transmit our signals?

At this time, it makes sense to try to formalize these notions of “maximum number of errors” and “efficiency.” Here are a series of definitions, that formalize the words and ideas we’ve been playing with in this talk:

9.2 Codes: Definitions and Ideas

Definition. A q -ary word of length n is any string of n numbers, each one of which is an element of $\{0, 1, 2, \dots, q-1\}$. We denote the collection of all such words by writing $(\mathbb{Z}_q)^n$.

For instance, 011101 is a binary string of length 6, and 123 is a 4-ary string of length 3.

Definition. A q -ary block code C of length n is a set C of words of length n , written in base q . In other words, it is any subset C of $(\mathbb{Z}_q)^n$.

word	signal to transmit
000	000000
100	100011
010	010101
001	001110
011	011011
101	101101
110	110110
111	111000

Example. The “repeat three times” code we described earlier is a 2-ary code of length 3, consisting of the two elements $\{(000), (111)\}$. We used it to encode a language with two symbols, specifically 0 and 1.

The second code we made is a 2-ary code of length 6, consisting of the 8 elements we wrote down in our table.

Definition. Given a q -ary code C of length n , we define its **information rate** as the quantity

$$\frac{\log_q(\# \text{ of elements in } C)}{n}$$

This, roughly speaking, captures the idea of how “efficient” a code is.

Example. The “repeat three times” code we described earlier contains two codewords of length 3; therefore, its information rate is $\frac{\log_2(2)}{3} = \frac{1}{3}$. This captures the idea that this code needed to transmit three bits to send any one bit of information.

Similarly, the second code we made contains 8 codewords of length six, and therefore has information rate $\frac{\log_2(8)}{6} = \frac{3}{6} = \frac{1}{2}$. Again, this captures the idea that this code needed to transmit two bits in order to send any one bit of information.

Definition. The **Hamming distance** $d_H(\mathbf{x}, \mathbf{y})$ between any two q -ary strings of length n is simply the number of places where these two elements disagree.

Given a code C , we say that the minimum distance of C , $d(C)$, is the smallest possible value of $d_H(\mathbf{x}, \mathbf{y})$ taken over all distinct \mathbf{x}, \mathbf{y} within the code. If $d(C) \geq k$, we will call such a code a distance- k code.

Example. The Hamming distance between the two words

12213, 13211

is 2, because they disagree in precisely two places. Similarly, the Hamming distance between the two words

TOMATO, POTATO

is 2, because these two words again disagree in precisely two places.

The “repeat three times” code from earlier has minimum distance 3, because the Hamming distance between 000 and 111 is 3.

Similarly, the second code we described from earlier has minimum distance 3, because every two words in our list disagreed in at least 3 places.

The following theorem explains why we care about this concept of distance:

Theorem. A code C can detect up to s errors in any received codeword as long as $d(C) \geq s + 1$. Similarly, a code C can correct up to t errors in any received codeword to the correct codeword as long as $d(C) \geq 2t + 1$.

Proof. If $d(C) \geq s + 1$, then making s changes to any codeword cannot change it into any other codeword, as every pair of codewords differ in at least $s + 1$ places. Therefore, our code will detect an error as long as at most s changes occur in any codeword.

Similarly, if $d(C) \geq 2t + 1$, then changing t entries in any codeword still means that it differs from any other codeword in at least $t + 1$ many places; therefore, the codeword we started from is completely unambiguous, and we can correct these errors. \square

Example. Using this theorem, we can see that both of our codewords can correct at most one error in any codeword, because their Hamming distances were both three.

9.3 Finding Codes

Now that we've made this formal, we can now state our question rigorously:

Problem. Let $A_q(n, d)$ denote the maximum number of elements in a block-length n q -ary code C with $d(C) \geq d$. What is $A_q(n, d)$ for various values of q, n, d ?

This problem is **open** for most values of n, q, d ! To give examples,

$$2720 \leq A_2(16, 3) \leq 3276,$$

are the best bounds currently known⁴⁵ for these parameters, and in general $A_2(n, k)$ is open for $n \geq 17, k \in \{3, 4, 5, 6\}$.

So: we have a mathematical object that on one hand seems incredibly practical (almost all modern electronical objects need some automated way to correct for errors) and also shockingly open (we don't know what the most efficient codes are for even very small block lengths n and distances d !) This, for me, is the sign of an exciting area of research: we both know nothing and want to be able to know everything!

We do know **some** values of $A_q(n, d)$, though! We examine a handful of such values here, through the following theorems:

Theorem. $A_q(n, 1) = q^n$, for any q, n .

Proof. First, notice that any code C has distance at least 1; this is by definition, as

$$d(C) = \min_{c_1 \neq c_2 \in C} d(c_1, c_2),$$

and the smallest distance between any two nonequal words c_1, c_2 is at least 1 (because if they were distance 0, then they would be equal!)

Consequently, $A_q(n, 1)$ is just asking us for the maximum number of elements in a q -ary block-length n code, as the "distance 1" property is trivial to satisfy! Consequently, the code with the maximum number of elements is simply the code given by taking **all** of the possible code words: that is, it is the set of all q -ary strings of length n . This set has q^n elements (we're choosing n things from the set $\{0, 1, \dots, q-1\}$ with replacement where we care about the order!) so we've proven our claim. \square

Theorem. $A_q(n, n) = q$, for any q, n .

Proof. First, notice that if we take the code

$$\overbrace{000 \dots 0}^{n \text{ zeroes}}, \overbrace{111 \dots 1}^{n \text{ ones}}, \overbrace{222 \dots 2}^{n \text{ twos}}, \dots, \overbrace{q-1 \dots q-1}^{n \text{ copies of } (q-1)}$$

this code has q elements, all of which are distance n apart. So we have shown that $A_q(n, n) \geq q$ for any q, n .

So it suffices to prove that q is an upper bound for the size of any such code C , as well! To do this: take any q -ary block-length n code C with $d(C) \geq n$. Because C is a block-length n code, it is impossible for any two words to disagree in more than n places (as each word is length n); therefore, we can actually say that $d(C) = n$.

⁴⁵ See <http://www.win.tue.nl/~aeb/codes/binary-1.html> for the specific bounds stated here. For general information on what we know and do not know, check out <http://www.codetables.de/>!

Suppose, for contradiction, that C has $q+1$ or more words. Then, because there are q choices of symbol for the first symbol in each of the words of C , and there are $q+1$ words in total, there must be two words $c_1, c_2 \in C$ that start with the same symbol. But this means that $d(c_1, c_2) \leq n-1$, because they can disagree in at most $n-1$ places; in other words $d(C) \leq n-1$, a contradiction! Therefore, we have shown that $|C| \leq q$, as claimed. \square

Working with other cases, however, gets harder in a hurry. Consider the following problem:

Problem. What is $A_2(n, 2)$ for arbitrary n ?

Answer. We start answering this problem by gathering some data first. For $n = 2$, this is a pretty simple problem; we've already shown above that $A_2(2, 2) = 2$.

Let's try $n = 3$ now. Assume without loss of generality that the all-zero codeword 000 is in our binary block-length-3 distance 2 code C . What else can we put in this code?

Well, we could add in 111, to get the code

$$\{000, 111\};$$

by inspection, we cannot add any more words to this code, as any other word would either have one 1 (and thus be distance 1 from 000) or two 1's (and thus be distance 1 from 111.)

Alternately, instead of adding in 111, we could add in the three words 011, 101, 110, to get the code

$$\{000, 011, 101, 110\}.$$

This is a distance-2 code; all of the "two-1's" words are distance 2 from 000, and also distance 2 from each other (prove this if you don't see why!) We cannot add any more words to this code, as all remaining words have either one 1 or three 1's, and in either case are distance 1 from one of our "two 1's" words.

By exhaustion, then, we seem to have shown that $A_2(3, 2) = 4$. By using a similar technique, we can see that it looks likely that $A_2(4, 2) = 8$; if we start with the all-zero string 0000 and add in the "two-1's" words, we get

$$\{0000, 0011, 0101, 0110, 1001, 1010, 1100\},$$

to which we can also add the "four-1's" word 1111 to get

$$\{0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111\}.$$

We clearly cannot add more words to this code, as any other word would have either one or three ones, and thus be within distance 1 of one of our existing codewords!

Note that we haven't proven that this is the largest code possible; unlike our earlier work, we haven't considered other cases like adding in the "three-1's" words instead of the "two-1's" words. You can try (check it!) using those codewords, and verify that the resulting code is smaller than what we have here; however, it also seems likely that going through **all** of the other ways to grow this code would be painful / not a good proof method for us to pursue.

Instead, we'll stick with intuition for now; it "looks like" 8 is the size of the largest code for $n = 4$, and hopefully we'll come up with a proof of this later.

Similarly, you could study $A_2(5, 2)$. To continue the pattern, you'd want the all-zero word 00000, all of the "two 1's" words, of which there are $\binom{5}{2} = 10$ (because you have five slots to put 1's in, and you're placing two 1's,) and all of the "four 1's" words, of which

there are $\binom{5}{4} = 5$ (same reasoning.) This gives us $1 + 10 + 5 = 16$, which gives us an interesting pattern: we have

	$A_2(2, 2)$	$A_2(3, 2)$	$A_2(4, 2)$	$A_2(5, 2)$
Guesses	2	4	8	16

which might point to a conjecture of $A_2(n, 2) = 2^{n-1}$!

With this data collected, we transition from our guesses above to some actual proofs:

Theorem. $A_2(n, 2) = 2^{n-1}$.

Proof. As before, we prove this in two stages. First, we establish a lower bound of 2^{n-1} by giving a concrete example that shows this is possible.

To do this, consider the binary block-length n code C formed by taking all of the code words with an even number of 1's. I claim that this code has distance 2. To see why, consider any two words w_1, w_2 with $d(w_1, w_2) = 1$. These two words disagree in exactly one place; so, outside of that one place, they have the same number of 1's, and at that place one is 1 and the other is 0. But this means that one of w_1, w_2 has exactly one more 1 than the other; in other words, it is impossible for both w_1, w_2 to have an even number of 1's!

So, we have shown that C is a distance-2 block-length n binary code. How many elements are in C ? I claim that it is 2^{n-1} , or in other words that **half** of the total number of possible codewords⁴⁶ $(\mathbb{Z}_2)^n$ have an even number of 1's, and prove this by induction on n .

Our base case is immediate: if $n = 1$, then there is exactly one code word, 0, that has an even number of 1's.

For our inductive step: assume that half of the elements of $(\mathbb{Z}_2)^n$ have an even number of 1's, and the other half have an odd number of 1's. Look at the elements of $(\mathbb{Z}_2)^{n+1}$, and divide them into two groups: those whose last element is a 0, and those whose last element is a 1.

If you ignore the last element, then exactly half of the “last-element-zero” words have an even number of 1's, and half have an odd number of 1's; similarly, exactly half of the “last-element-one” words have an even number of 1's, and half have an odd number of 1's, by induction! Therefore, by taking the even-number-1's half from the last-element-zero words, and the odd-number-1's half from the “last-element-one” words, we have on one hand all of the “total number of 1's is even” numbers, and on the other hand half of the elements in total from $(\mathbb{Z}_2)^{n+1}$! So we've proven our claim by induction, and therefore proved that $|C|$ is precisely half of $|(\mathbb{Z}_2)^n| = 2^n$; i.e. $|C| = 2^{n-1}$, as claimed.

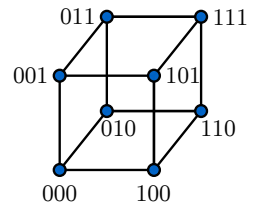
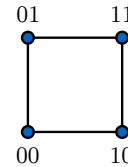
This does the first half of our proof; that 2^{n-1} is a lower bound. We now seek to prove that it is an upper bound, as well!

To see this, consider the following graph:

- **Vertices:** the set $(\mathbb{Z}_2)^n$.
- **Edges:** connect two words w_1, w_2 in our set with an edge if and only if $d(w_1, w_2) = 1$.

We draw a few examples of these graphs at right.

Notice that these graphs are precisely just n -dimensional cubes! This shouldn't be too surprising — after all, the vertices of the unit cube in n dimensions are precisely all of the points in \mathbb{R}^n whose coördinates are either 0 or 1 — but it's pretty nonetheless.



⁴⁶Because there are 2^n many words in $(\mathbb{Z}_2)^n$, as you've seen in the enumeration part of this paper!

What I want to consider instead, here, is what **codes** correspond to on these graphs. In particular, suppose we take the graph above, and draw in the two distance-2 block-length-3 binary codes we came up with earlier.

Notice how in both cases, we never had any edges connecting two codewords! If you think about this for a moment, this makes sense: if two words are connected by an edge, then they are distance 1 from each other, which would cause a problem for any code in which all of our words are distance at least 2 from each other.

Also, notice that in the graph above, each vertex has degree n in the n -dimensional cube! This is again not hard to see: if we have any word w in $(\mathbb{Z}_2)^n$, there are exactly n words at distance 1 from w , as we get exactly one such word for each bit in w that we can change.

By combining these observations, we can prove our claim! To do this, take any block-length n distance-2 binary code C , and label it on the n -dimensional cube graphs above. Redraw our graphs by “clumping together” all of the codewords into one cluster, and the “non-codewords” into another cluster:

In the drawing above, the edges going from one cluster to the other are bolded, while the edges within a cluster (where they exist; they exist on the LHS, but not on the RHS) are dashed.

If you look at the “codeword” cluster, you can immediately see that the total number of edges connecting the codeword cluster to the non-codeword cluster is just $|C| \cdot n$.

This is because there are $|C|$ vertices in the codeword cluster, each has n edges leaving it by our earlier observation, and all of those edges must go to non-codewords (again, by our earlier observations!)

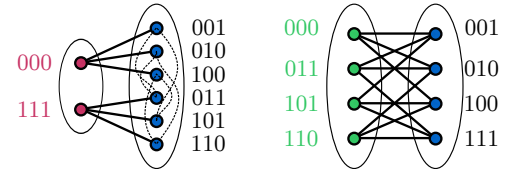
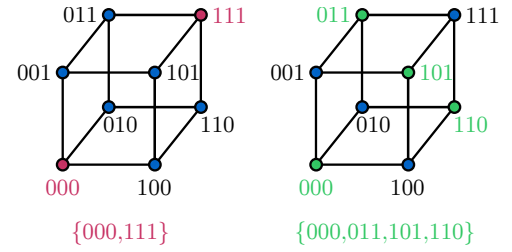
Conversely, if you look at the “non-codeword” cluster and count edges there, you can see that the total number of edges from it to the codeword cluster is at most $(\# \text{ non-codewords}) \cdot n$.

This is because (again) each vertex has degree n ! Notice that this time we don’t know the exact number of edges going from non-codewords to codewords; not every edge from a non-codeword needs to go to a codeword, as our examples earlier demonstrate! However, we know that there are **at most** as many edges from non-codewords to codewords as our bound above, which is enough for our claim.

Finally: we know that these two quantities are equal, because they’re both counting the same thing: the edges from the codewords to the non-codewords!

Therefore, we have $|C| \cdot n \leq (\# \text{ non-codewords}) \cdot n$. Dividing by n gives us $|C| \leq (\# \text{ non-codewords})$.

But this means that there must be at least as many non-codewords as codewords; in other words, that C can contain at most half of the elements of $(\mathbb{Z}_2)^n$! In other words, we’ve proven that $A_2(n, 2) \leq 2^{n-1}$, which finishes the second part of our proof! \square



Credits

UoA 2018

Thanks go to the following students who've caught typos on earlier versions of these notes:

- Steve Quindo
-
-

(If you spot a typo, email me to have your name added here!
Also, if you stop by my office I am happy to pay a bounty of one chocolate fish per typo found :D)