CS251 Discrete Mathematics

Weekly Tutorial Notes

Based on Epp, *Discrete Mathematics with Applications*

Abstract

These notes provide supplementary material for CS251 Discrete Mathematics. Each week covers key definitions, theorems, worked examples, and practice problems aligned with the Epp textbook. The emphasis is on developing proof techniques and problem-solving skills essential for computer science.

Contents

1 Chapter 0: Proof Refresher

Or: "How to convince someone you're right, rigorously"

This chapter reviews the proof techniques and logical foundations from the first quarter. Think of it as a reference card you can flip back to throughout the course. If anything here feels unfamiliar rather than "oh right, that," you may want to spend extra time with the readings.

Propositional Logic Review

Let's start with the building blocks. A **proposition** is a statement that is either true or false—no maybes, no "it depends," no quantum superpositions. "It's raining" is a proposition. "Is it raining?" is not (it's a question). "This statement is false" is not (it's a paradox, and we don't allow those).

Definition 1.1 (Logical connectives)**.** Let P and Q be propositions.

Symbol	Name	True when...
$\neg P$	Negation (NOT)	P is false
$P \wedge Q$	Conjunction (AND)	Both P and Q are true
$P \vee Q$	Disjunction (OR)	At least one of P, Q is true
$P \to Q$	Implication (IF-THEN)	P is false, or Q is true
$P \leftrightarrow Q$	Biconditional (IFF)	P and Q have the same truth value

The one that trips everyone up is implication. "$P \to Q$" is true whenever P is false *or* Q is true. This means "if pigs fly, then I'm the queen of England" is a *true* statement (because pigs don't fly). This feels weird, but it's the definition that makes mathematics work. We'll just have to live with it.

Theorem 1.1 (Key equivalences)**.** *The following are logically equivalent (have the same truth table):*

$$\begin{aligned} P \to Q &\equiv \neg P \vee Q \equiv \neg Q \to \neg P \quad \textit{(contrapositive)} \\ \neg(P \wedge Q) &\equiv \neg P \vee \neg Q \quad \textit{(De Morgan)} \\ \neg(P \vee Q) &\equiv \neg P \wedge \neg Q \quad \textit{(De Morgan)} \\ P \to Q &\not\equiv Q \to P \quad \textit{(converse is NOT equivalent!)} \end{aligned}$$

That last line deserves emphasis. "If it's raining, the ground is wet" does *not* mean "if the ground is wet, it's raining." Someone could have turned on the sprinklers. The converse of a true statement can be false. This mistake shows up constantly in student proofs.

Predicate Logic Review

Propositional logic lets us talk about specific statements. But what about "all even numbers are divisible by 2" or "there exists a prime greater than 100"? For that, we need **quantifiers**.

Definition 1.2 (Quantifiers)**.** Let $P(x)$ be a **predicate**—a statement that depends on a variable x and becomes true or false once you plug in a specific value.

Universal quantifier: $\forall x \in D.\, P(x)$ means "for all x in domain D, $P(x)$ holds." To prove this, you need to show $P(x)$ for *every* possible x. No exceptions.

Existential quantifier: $\exists x \in D.\, P(x)$ means "there exists some x in D such that $P(x)$ holds." To prove this, you need to find *one* example.

Theorem 1.2 (Negating quantifiers)**.** *Here's how negation interacts with quantifiers:*

$$\neg(\forall x.\, P(x)) \equiv \exists x.\, \neg P(x)$$
$$\neg(\exists x.\, P(x)) \equiv \forall x.\, \neg P(x)$$

In English: the negation of "all cats are black" isn't "no cats are black"—it's "there exists a cat that isn't black." You only need one counterexample to disprove a universal claim.

Theorem 1.3 (Quantifier order matters)**.**

$$\forall x.\, \exists y.\, P(x, y) \quad \not\equiv \quad \exists y.\, \forall x.\, P(x, y)$$

This is subtle but crucial. Let $P(x, y)$ mean "$y > x$" over $\mathbb{R}$:

- $\forall x.\, \exists y.\, (y > x)$*: "For every number, there's a larger one."* ***True.*** *(Given any x, take $y = x + 1$.)*
- $\exists y.\, \forall x.\, (y > x)$*: "There's a number larger than all others."* ***False.*** *(No largest real number exists.)*

The difference: in the first, y can depend on x (different x, different y). In the second, you must pick a single y that works for all *x simultaneously. That's a much stronger requirement.*

Proof Techniques

Now for the main event: how to actually prove things. There are several standard approaches, and part of mathematical maturity is recognizing which technique fits which situation.

Proof Strategy

[Direct proof] To prove $P \to Q$: Assume P is true, then show Q follows.
This is the most straightforward approach. You start with what you're given and push forward until you reach what you want.
Template:

> *Assume P. [Reasoning...] Therefore Q.*

Example. Prove: If n is even, then n^2 is even.
Proof. Assume n is even. Then $n = 2k$ for some integer k. (This is what "even" means—we're unpacking the definition.) So $n^2 = (2k)^2 = 4k^2 = 2(2k^2)$. Since $2k^2$ is an integer, n^2 has the form $2 \cdot (\text{integer})$, which is the definition of even. □

Proof Strategy

[Proof by contrapositive] To prove $P \to Q$: Prove $\neg Q \to \neg P$ instead (they're logically equivalent).
Why would you do this? Sometimes $\neg Q$ gives you more to work with than P does. If you're stuck on a direct proof, try the contrapositive.
Template:

> *We prove the contrapositive. Assume $\neg Q$. [Reasoning...] Therefore $\neg P$.*

Example. Prove: If n^2 is odd, then n is odd.

Proof. We prove the contrapositive: if n is even, then n^2 is even.
Assume n is even. Then $n = 2k$ for some integer k. So $n^2 = 4k^2 = 2(2k^2)$, which is even. □
(Notice this is almost the same as the previous example. The contrapositive often turns a "weird" statement into a more natural one.)

Proof Strategy

[Proof by contradiction] To prove P: Assume $\neg P$ and derive a contradiction.
This is the "suppose not" approach. You assume the opposite of what you want to prove, then show this leads somewhere impossible. The logic is: if assuming $\neg P$ leads to nonsense, then P must be true.
Template:

> *Suppose for contradiction that $\neg P$. [Reasoning...] This contradicts [known fact]. Therefore P.*

Example. Prove: $\sqrt{2}$ is irrational.
Proof. Suppose for contradiction that $\sqrt{2}$ is rational. Then $\sqrt{2} = a/b$ where a, b are integers with no common factors (we've reduced to lowest terms).
Squaring both sides: $2 = a^2/b^2$, so $a^2 = 2b^2$. This means a^2 is even, so a is even (we proved this above). Write $a = 2c$ for some integer c.
Substituting: $(2c)^2 = 2b^2$, so $4c^2 = 2b^2$, so $b^2 = 2c^2$. This means b^2 is even, so b is even.
But now both a and b are even—they share a factor of 2. This contradicts our assumption that a/b was in lowest terms. □
(This is one of the oldest proofs in mathematics, attributed to the ancient Greeks. Legend has it that Hippasus was drowned for revealing it.)

Proof Strategy

[Proof by cases] To prove P: Partition into exhaustive cases and prove P in each one.
Sometimes a statement is true for different reasons in different situations. Rather than finding one unified argument, you can just handle each situation separately.
Template:

> *Case 1: [Condition]. [Proof of P in this case.]*
> *Case 2: [Condition]. [Proof of P in this case.]*
> *These cases are exhaustive, so P holds.*

Example. Prove: For any integer n, $n^2 + n$ is even.
Proof. Every integer is either even or odd. We handle each case.
Case 1: n is even. Then n^2 is even (proved earlier). So $n^2 + n$ is even + even = even. ✓
Case 2: n is odd. Then n^2 is odd (similar argument). So $n^2 + n$ is odd + odd = even. ✓
Since every integer is either even or odd, we've covered all cases. □

Mathematical Induction

Induction is how we prove statements about all natural numbers (or anything else we can line up in a sequence). The idea is beautifully simple: prove the first case, then prove that each case implies

the next. Like dominoes.

Proof Strategy

[Weak induction] To prove $\forall n \geq n_0.\, P(n)$:

1. **Base case:** Prove $P(n_0)$.
2. **Inductive step:** Prove that $P(k) \to P(k+1)$ for an arbitrary $k \geq n_0$.

Why it works: The base case establishes $P(n_0)$. The inductive step gives us $P(n_0) \to P(n_0+1)$, so $P(n_0+1)$ is true. The inductive step again gives $P(n_0+1) \to P(n_0+2)$, so $P(n_0+2)$ is true. And so on, forever. We've built an infinite chain of implications, starting from a known truth.

Example 1.1. Prove: $\displaystyle\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ for all $n \geq 1$.

Proof. By induction on n.
Base case ($n = 1$): The left side is $\sum_{i=1}^{1} i = 1$. The right side is $\frac{1 \cdot 2}{2} = 1$. They match. ✓
Inductive step: Assume the formula holds for some $k \geq 1$:

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2} \quad \text{(this is our \textbf{inductive hypothesis})}$$

We need to prove it holds for $k+1$:

$$\begin{aligned}
\sum_{i=1}^{k+1} i &= \left(\sum_{i=1}^{k} i\right) + (k+1) && \text{(peel off the last term)} \\
&= \frac{k(k+1)}{2} + (k+1) && \text{(apply inductive hypothesis!)} \\
&= \frac{k(k+1) + 2(k+1)}{2} && \text{(common denominator)} \\
&= \frac{(k+1)(k+2)}{2} && \text{(factor)}
\end{aligned}$$

This is exactly the formula with $n = k+1$. ✓
By induction, the formula holds for all $n \geq 1$. □

The key move in the inductive step is the second line: we *used* the inductive hypothesis. If your inductive step doesn't reference the assumption $P(k)$, something's wrong—either your proof has a bug, or you didn't need induction in the first place.

Proof Strategy

[Strong induction] To prove $\forall n \geq n_0.\, P(n)$:

1. **Base case(s):** Prove $P(n_0)$ (and possibly several more).
2. **Inductive step:** Assume $P(n_0), P(n_0+1), \ldots, P(k)$ all hold. Prove $P(k+1)$.

The difference from weak induction: you get to assume P holds for *all* values up to k, not

just k itself. Use strong induction when proving $P(k+1)$ requires jumping back further than just one step.

Example 1.2. Prove: Every integer $n \geq 2$ can be written as a product of primes.

Proof. By strong induction on n.

Base case ($n = 2$): 2 is prime, so it's a "product of primes" (a product with one factor). ✓

Inductive step: Assume every integer from 2 to k can be written as a product of primes. We need to show $k + 1$ can too.

Case 1: $k + 1$ is prime. Then $k + 1$ is trivially a product of primes. ✓

Case 2: $k + 1$ is composite. Then $k + 1 = ab$ where $2 \leq a, b < k + 1$. (Both factors are smaller than $k + 1$ but at least 2.)

Since $a \leq k$ and $b \leq k$, the inductive hypothesis applies to both. So a is a product of primes and b is a product of primes. Therefore $k + 1 = ab$ is also a product of primes. ✓

By strong induction, every $n \geq 2$ is a product of primes. □

(This is half of the Fundamental Theorem of Arithmetic. The other half—that the factorization is unique—requires more work.)

Proof Strategy

[Structural induction] To prove a property P holds for all elements of a recursively-defined set S:

1. **Base case(s):** Prove P for each base element of S.
2. **Inductive step(s):** For each recursive rule, assume P holds for the inputs and prove P for the output.

This is induction generalized beyond numbers. If you've defined a data structure recursively (trees, lists, formulas, etc.), structural induction is the natural way to prove things about it.

Example 1.3. Define binary trees recursively:

- **Base:** A single node • is a binary tree.
- **Recursive:** If T_1 and T_2 are binary trees, then the tree with a root node connected to T_1 (left) and T_2 (right) is a binary tree.

Prove: Every binary tree has one more leaf than internal node.

Proof. By structural induction. Let $P(T)$ be the statement "T has one more leaf than internal node."

Base case: A single node • has 1 leaf (itself) and 0 internal nodes. Since $1 = 0 + 1$, we have one more leaf than internal node. ✓

Inductive step: Suppose T_1 has ℓ_1 leaves and i_1 internal nodes with $\ell_1 = i_1 + 1$. Similarly, T_2 has $\ell_2 = i_2 + 1$.

Form tree T by connecting a new root to T_1 and T_2. In T:

- The leaves are exactly the leaves of T_1 and T_2 (the new root isn't a leaf).
- The internal nodes are the internal nodes of T_1 and T_2, plus the new root.

So:

$$\text{leaves}(T) = \ell_1 + \ell_2$$
$$\text{internal}(T) = i_1 + i_2 + 1$$

Check: $\ell_1 + \ell_2 = (i_1 + 1) + (i_2 + 1) = (i_1 + i_2 + 1) + 1 = \text{internal}(T) + 1$. ✓
By structural induction, every binary tree has one more leaf than internal node. □

Common Inference Rules

These are the "legal moves" in formal reasoning. You'll use them implicitly in most proofs.

Name	Rule	In English
Modus ponens	$P,\ P \to Q \vdash Q$	"If P and 'P implies Q', then Q"
Modus tollens	$\neg Q,\ P \to Q \vdash \neg P$	"If not Q and 'P implies Q', then not P"
Hypothetical syllogism	$P \to Q,\ Q \to R \vdash P \to R$	Chain implications together
Disjunctive syllogism	$P \lor Q,\ \neg P \vdash Q$	"It's P or Q, and not P, so Q"
Conjunction	$P,\ Q \vdash P \land Q$	Combine two facts
Simplification	$P \land Q \vdash P$	Extract one part of an AND
Addition	$P \vdash P \lor Q$	Weaken to a disjunction
Resolution	$P \lor Q,\ \neg P \lor R \vdash Q \lor R$	The workhorse of automated theorem provers

Modus ponens and modus tollens are the two you'll use most. Modus ponens is "forward reasoning" (from premise to conclusion). Modus tollens is "backward reasoning" (from failure of conclusion to failure of premise).

Existential and Universal Proofs

Proof Strategy

[Proving $\exists x.\, P(x)$] Find a specific witness c and show $P(c)$ holds.
This is often the easiest type of proof: just produce an example.
Example. Prove: There exists an integer n such that $n^2 = n$.
Proof. Take $n = 1$. Then $1^2 = 1$. □
(Or take $n = 0$. There are often multiple witnesses.)

Proof Strategy

[Proving $\forall x.\, P(x)$] Let x be an *arbitrary* element of the domain and prove $P(x)$ without assuming anything special about x.
The word "arbitrary" is key. You can't say "let $x = 5$" because then you've only proved $P(5)$, not $P(x)$ for all x.
Example. Prove: For all integers n, if n is odd, then n^2 is odd.
Proof. Let n be an arbitrary odd integer. Then $n = 2k + 1$ for some integer k.
So $n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$.
This has the form $2m + 1$ where $m = 2k^2 + 2k$ is an integer, so n^2 is odd. □

Proof Strategy

[Disproving $\forall x.\, P(x)$] Find a counterexample: a specific c where $P(c)$ is false.
Remember: the negation of "for all x, $P(x)$" is "there exists an x where $P(x)$ fails." One counterexample is enough.
Example. Disprove: For all primes p, $2^p - 1$ is prime.
Counterexample. Let $p = 11$, which is prime. Then $2^{11} - 1 = 2047 = 23 \times 89$. Since 2047 is composite, the claim is false. □
(Numbers of the form $2^p - 1$ are called Mersenne numbers. When they're prime, they're Mersenne primes. Finding large Mersenne primes is an active area of number theory.)

Common Pitfalls

Here's where students most often go wrong. Read these carefully—you'll probably make at least one of these mistakes this quarter, and recognizing it quickly will save you grief.

Common Mistake

Assuming what you're trying to prove. In a direct proof of $P \to Q$, you assume P, not Q. If you find yourself writing "Assume Q..." in a direct proof, you've gone circular. The whole point is to *derive* Q from other things.

Common Mistake

Confusing implication with its converse. $P \to Q$ is NOT the same as $Q \to P$. Proving "if n is even, then n^2 is even" does *not* prove "if n^2 is even, then n is even." (The latter happens to be true, but it requires a separate proof.)

Common Mistake

Induction: not using the hypothesis. In the inductive step, you *must* use the assumption that $P(k)$ holds. If your proof of $P(k+1)$ doesn't mention $P(k)$ anywhere, either:

- Your proof has a gap (most likely), or
- You didn't actually need induction (rare, but possible)

Common Mistake

Induction: wrong base case. If your claim is "for all $n \geq 5$, $P(n)$," your base case must be $n = 5$, not $n = 0$ or $n = 1$. The base case is where your chain of dominoes starts.

Common Mistake

Proof by example. Checking that $P(1), P(2), P(3)$ are true does *not* prove $\forall n.\, P(n)$. You need either a general argument or proper induction. "It worked for the first few cases" is not a proof.

Common Mistake

Existential overgeneralization. From "there exists an x with property P," you cannot conclude that every x has property P. Just because some integer is even doesn't mean all integers are even.

Practice

1. Prove by induction: $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$.
2. Prove by induction: $n! > 2^n$ for all $n \geq 4$. (What's the base case?)
3. Prove by strong induction: Every amount of postage ≥ 12 cents can be made using 4-cent and 5-cent stamps. (Hint: you'll need multiple base cases.)
4. Prove by contrapositive: If n^2 is divisible by 3, then n is divisible by 3.
5. Prove by contradiction: There are infinitely many primes. (This is Euclid's proof, one of the most beautiful in mathematics.)
6. Prove or disprove: For all integers a, b, c, if $a \mid bc$, then $a \mid b$ or $a \mid c$. (The notation $a \mid b$ means "a divides b.")
7. Prove by structural induction: For any arithmetic expression built from integers using $+$ and $\times$, the result is an integer. (First, define the set of arithmetic expressions recursively.)
8. Negate the following statement, then determine which version (original or negation) is true:

   "For every $\epsilon > 0$, there exists $\delta > 0$ such that for all x, if $|x| < \delta$ then $|f(x)| < \epsilon$."

   (This is related to the definition of a limit at 0.)
9. Find the error in this "proof":

   Claim: All horses are the same color.

   "Proof" by induction on the number of horses:

   Base case: One horse is trivially the same color as itself. ✓

   Inductive step: Assume any group of k horses are the same color. Given $k+1$ horses, remove one; the remaining k are the same color (by the inductive hypothesis). Put that horse back and remove a *different* one; those k horses are also the same color. Since the two groups overlap, all $k+1$ horses are the same color. □

   (Where exactly does this argument fail?)
10. Prove: For all sets A and B, if $A \subseteq B$, then $\mathcal{P}(A) \subseteq \mathcal{P}(B)$.

2 Week 1: Set Theory

Or: "What even is a collection of things?"

Reading

Epp §6.1–6.4.
Category theory companion: Week 1–2 (`category_theory_companion.pdf`).

Why Sets?

Here's a question that seems like it should be easy: what do we mean when we say "the even numbers" or "all the students in this class" or "every program that halts"?

We're gesturing at *collections*. Bags of stuff. And you might think—reasonably!—that this is so obvious it doesn't need formalization. Surely we can just talk about collections without making a federal case out of it?

Well. Mathematicians tried that in the late 1800s, and it went *spectacularly* badly. Bertrand Russell came along and said "okay, consider the collection of all collections that don't contain themselves" and the whole edifice caught fire. We'll get to that disaster later. For now, just know: sets are the careful, paradox-avoiding way we talk about collections in mathematics.

Learning objectives

- Translate between roster and set-builder notation.
- Prove set identities with the element method.
- Apply standard set laws (commutative, associative, distributive).
- Use power sets and partitions correctly.
- Understand Cartesian products and their properties.

Key definitions and facts

What Is a Set?

A **set** is a collection of distinct objects where duplicates don't count (writing $\{1, 1, 2\}$ is the same as $\{1, 2\}$) and order doesn't matter ($\{1, 2\} = \{2, 1\}$).

That's basically it. A bag of things. The things in the bag are called **elements** or **members**, and we write $x \in A$ to mean "x is in the set A."

Now, you might reasonably ask: what counts as a "thing"? Can sets contain other sets? Can a set contain itself? What about the set of all sets?

These are excellent questions, and the answers are: yes, technically yes but we try to avoid it, and *absolutely not* (that way lies Russell's paradox). Modern set theory (ZFC, if you want to look it up) has careful rules about what you're allowed to construct. For this course, we'll stay in safe territory.

Two Ways to Write Sets

Roster notation is just listing the elements:

$$\{1, 2, 3\}, \quad \{a, b, c\}, \quad \{\text{Alice}, \text{Bob}\}$$

This works great for small finite sets. Less great for "all even integers."

Set-builder notation describes elements by a property:

$$\{x \in \mathbb{Z} : x \text{ is even}\} = \{x \in \mathbb{Z} : \exists k \in \mathbb{Z},\, x = 2k\}$$

Read the colon as "such that." The thing on the left of the colon is what you're collecting; the thing on the right is the condition it must satisfy.

Warning (Specify your universe). Students often write $\{x : x^2 = 4\}$ when they mean $\{x \in \mathbb{R} : x^2 = 4\}$. Where are you drawing x from? The integers? The reals? The complex numbers? This matters! $\{x \in \mathbb{N} : x^2 = 4\}$ is just $\{2\}$, but $\{x \in \mathbb{Z} : x^2 = 4\} = \{-2, 2\}$.

Subsets and Equality

Definition 2.1 (Subset and equality)**.** For sets A, B, $A \subseteq B$ means every element of A is in B. Think of it as "A fits inside B."

We define set **equality** in terms of subsets: $A = B$ means $A \subseteq B$ and $B \subseteq A$.

Why define equality this way? Because sets don't have any structure *except* their elements. Two sets are the same if they have exactly the same members. The $\subseteq$-both-ways definition captures this: if everything in A is in B, and everything in B is in A, then they must have exactly the same elements.

This gives us our main proof technique for showing sets are equal: **prove inclusion in both directions**. You'll do this a lot.

Set Operations (The Boolean Algebra Connection)

Okay, here's where it gets fun. We can combine sets in ways that mirror logical operations:

Definition 2.2 (Set operations)**.** Let A and B be sets:

- **Union:** $A \cup B = \{x : x \in A \text{ or } x \in B\}$ (like logical OR)
- **Intersection:** $A \cap B = \{x : x \in A \text{ and } x \in B\}$ (like logical AND)
- **Difference:** $A \setminus B = \{x : x \in A \text{ and } x \notin B\}$ (like AND NOT)
- **Complement:** $A^c = \{x \in U : x \notin A\}$ (relative to universal set U) (like NOT)

This correspondence isn't a coincidence! Sets form a **Boolean algebra**, and if you squint, everything we do with sets is secretly logic in disguise. De Morgan's laws? Same for sets as for propositions. If you've written code with boolean conditions, you already know this: `!(a || b)` is the same as `!a && !b`. Same idea, different notation.

Theorem 2.1 (Set laws)**.** *For all sets A, B, C:*

- ***Commutative:*** $A \cup B = B \cup A$, $A \cap B = B \cap A$

- ***Associative:*** $(A \cup B) \cup C = A \cup (B \cup C)$
- ***Distributive:*** $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- ***Absorption:*** $A \cup (A \cap B) = A$, $A \cap (A \cup B) = A$
- ***Identity:*** $A \cup \emptyset = A$, $A \cap U = A$

Theorem 2.2 (De Morgan's laws)**.** $(A \cup B)^c = A^c \cap B^c$ *and* $(A \cap B)^c = A^c \cup B^c$.

If these look exactly like the logical equivalences from Week 0, that's because they are. The Boolean algebra structure is the same; we're just using $\cup$ instead of $\vee$ and $\cap$ instead of $\wedge$.

The Power Set (Where Things Get Meta)

Definition 2.3 (Power set and partition)**.** The **power set** $\mathcal{P}(A)$ is the set of all subsets of A. If $|A| = n$, then $|\mathcal{P}(A)| = 2^n$.

A **partition** of A is a collection of nonempty, pairwise disjoint subsets whose union is A.

Yes, we're making a set whose elements are themselves sets. This is allowed and useful.

Try this before reading on: list all subsets of $\{1, 2\}$.

(Take a moment to write them down before continuing.)

Got them? There are four:

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

Notice: $\emptyset$ (the empty set) is a subset of everything, including $\{1, 2\}$. And $\{1, 2\}$ is a subset of itself. Both of these trip people up.

Why is $|\mathcal{P}(A)| = 2^{|A|}$*?* Think about it this way: for each element of A, you have a binary choice—include it or don't. That's 2 choices per element, so 2^n total subsets. (We'll formalize this counting argument in Week 4.)

Cartesian Products

Definition 2.4 (Cartesian product)**.** The Cartesian product $A \times B = \{(a, b) : a \in A, b \in B\}$. We have $|A \times B| = |A| \cdot |B|$ for finite sets.

Unlike sets, pairs *do* care about order: $(1, 2) \neq (2, 1)$.

If you've used coordinates in a plane, you've used Cartesian products: $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$. The name comes from Descartes, who figured out you could turn geometry into algebra by slapping coordinates on everything.

For finite sets: $|A \times B| = |A| \cdot |B|$. Two elements in A, three in B? Six pairs. This will be foundational for counting in Week 4.

Russell's Paradox (The Promised Disaster)

Warning (Russell's paradox). Remember when I said "the set of all sets" is forbidden? Here's why.

Naive set theory said: any property defines a set. Want the set of all red things? Done. The set of all numbers? Sure. The set of all sets? Why not!

Russell asked: what about $R = \{x : x \notin x\}$—the set of all sets that *don't contain themselves*?

Is $R \in R$?

- If **yes**: then by definition of R, we need $R \notin R$. Contradiction.
- If **no**: then R satisfies the condition $R \notin R$, so $R \in R$. Contradiction.

We're cooked either way. This isn't a trick or a puzzle you can wriggle out of—it's a genuine inconsistency in unrestricted set formation.

Modern set theory (ZFC) fixes this by being very careful about which collections count as sets. You can't just conjure sets into existence with any old property. The rules are somewhat technical, but the upshot is: we stay away from self-referential monstrosities, and everything works out.

Philosophical aside: Some mathematicians find ZFC's restrictions unsatisfying—they're somewhat ad-hoc patches to avoid known paradoxes. Type theory offers a different foundation where Russell's paradox can't even be stated. If you're curious, the Agda materials touch on this.

Worked examples

Example 2.1 (Boolean-style simplification)**.** Simplify the set expression $(A \cap B) \cup (A \cap B^c)$.

Solution. If you stare at this for a moment, you might notice we're taking things in A that are also in B, then unioning with things in A that *aren't* in B. That sounds like just everything in A.

Let's verify with algebra. Factor out A using distributivity:

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

The key insight is that $B \cup B^c = U$ (law of excluded middle for sets—everything is either in B or not in B).

Example 2.2 (Proving the distributive law)**.** Prove $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Proof. We show mutual inclusion. This is the workhorse technique for set equalities.

($\subseteq$) Let $x \in A \cap (B \cup C)$. Then $x \in A$ and $x \in B \cup C$. Since $x \in B \cup C$, either $x \in B$ or $x \in C$.

- If $x \in B$: then $x \in A \cap B$, so $x \in (A \cap B) \cup (A \cap C)$.
- If $x \in C$: then $x \in A \cap C$, so $x \in (A \cap B) \cup (A \cap C)$.

($\supseteq$) Let $x \in (A \cap B) \cup (A \cap C)$. Then $x \in A \cap B$ or $x \in A \cap C$. In either case, $x \in A$, and $x \in B$ or $x \in C$, so $x \in B \cup C$. Thus $x \in A \cap (B \cup C)$. □

Example 2.3 (De Morgan's law)**.** Prove $(A \cup B)^c = A^c \cap B^c$.

Proof. This one we can do with a chain of if-and-only-ifs:

$x \in (A \cup B)^c$ iff $x \notin A \cup B$ iff not($x \in A$ or $x \in B$) iff ($x \notin A$ and $x \notin B$) (De Morgan for logic!) iff $x \in A^c$ and $x \in B^c$ iff $x \in A^c \cap B^c$. □

Notice how the set proof *is* the logic proof—we just translated notation.

Example 2.4 (Power set enumeration)**.** List the power set $\mathcal{P}(\{1, 2\})$ and verify that $|\mathcal{P}(\{1, 2\})| = 2^2$.

Solution. The subsets of $\{1, 2\}$ are:

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

We have $|\mathcal{P}(\{1, 2\})| = 4 = 2^2$. ✓

A systematic way to list these: for each element, decide in/out. That's a binary string of length 2:

1 in?	2 in?	Subset
no	no	$\emptyset$
yes	no	$\{1\}$
no	yes	$\{2\}$
yes	yes	$\{1, 2\}$

This is why $|\mathcal{P}(A)| = 2^{|A|}$—each element gives a binary choice.

Example 2.5 (Cartesian product)**.** Let $A = \{0, 1\}$ and $B = \{a, b, c\}$. Find $A \times B$ and $|A \times B|$.

Solution.

$$A \times B = \{(0, a), (0, b), (0, c), (1, a), (1, b), (1, c)\}$$

We have $|A \times B| = 6 = 2 \times 3 = |A| \cdot |B|$. ✓

Think of it as a grid: rows indexed by A, columns by B, and each cell is a pair.

Example 2.6 (Disproving a false identity)**.** Disprove: $(A \cup B) \setminus C = A \cup (B \setminus C)$ for all sets A, B, C.

Solution. When asked to disprove a "for all" statement, we need one counterexample. The question is: where does this identity go wrong?

Think about what each side does:

- LHS: Take everything in A or B, then remove anything in C.
- RHS: Keep all of A, then add things from B that aren't in C.

The difference: the LHS removes C from A too, but the RHS keeps A intact!

So we need: something in $A \cap C$ that's not in B. Try $A = \{1\}$, $B = \{2\}$, $C = \{1\}$:

- LHS: $A \cup B = \{1, 2\}$, so $(A \cup B) \setminus C = \{1, 2\} \setminus \{1\} = \{2\}$.
- RHS: $B \setminus C = \{2\} \setminus \{1\} = \{2\}$, so $A \cup (B \setminus C) = \{1\} \cup \{2\} = \{1, 2\}$.

Since $\{2\} \neq \{1, 2\}$, the identity fails. □

Example 2.7 (Absorption law)**.** Prove the absorption law: $A \cup (A \cap B) = A$.

Proof. Before diving into the formal proof, let's think about why this should be true. $A \cup (A \cap B)$ says: take everything in A, and also add things that are in both A and B. But wait—things in both A and B are already in A! So we're adding nothing new.

Formally:

($\supseteq$) If $x \in A$, then $x \in A \cup (A \cap B)$ since x is in the first part of the union.

($\subseteq$) If $x \in A \cup (A \cap B)$, then $x \in A$ or $x \in A \cap B$. In either case, $x \in A$ (since $A \cap B \subseteq A$).

Therefore $A \cup (A \cap B) = A$. □

Going Deeper: Arrows and Diagrams

This begins a running thread through the course: learning to think in terms of *arrows* and *diagrams*. If you find yourself thinking "these concepts feel weirdly similar across weeks," you're not wrong—and this section will eventually show you why.

For more detail, see the Category Theory Companion, Week 1–2.

Functions as Arrows

We've been writing $f : A \to B$ for functions. The arrow notation isn't accidental—it suggests *direction* and *connection*. Let's take this seriously.

Key observations:

- **Arrows compose:** If $f : A \to B$ and $g : B \to C$, we get $g \circ f : A \to C$.
- **Composition is associative:** $(h \circ g) \circ f = h \circ (g \circ f)$, so we can write $h \circ g \circ f$ without ambiguity.
- **Identity arrows exist:** Every set A has an identity function $\mathrm{id}_A : A \to A$ with $\mathrm{id}_A(x) = x$.
- **Identities are neutral:** $f \circ \mathrm{id}_A = f$ and $\mathrm{id}_B \circ f = f$.

If this looks like the axioms for something more general, you're onto something. We'll return to this.

Elements as Maps from 1

In the category of sets, a singleton set $1 = \{*\}$ is special. Every element $a \in A$ corresponds to a unique map $1 \to A$ that sends $*$ to a. In categorical language, *elements of A are arrows from 1 to A.*
This might seem like pointless abstraction, but it pays off: it lets us talk about "elements" in contexts where there aren't any literal elements, like in categories of spaces or diagrams.

Terminal and Initial Objects

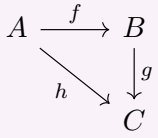
The singleton set 1 is a **terminal object**: for every set A, there is exactly one map $A \to 1$. (What else could it do? Send everything to the only point available.)
The empty set $\emptyset$ is an **initial object**: for every set A, there is exactly one map $\emptyset \to A$. (The empty function—it has no elements to send anywhere, so there's exactly one way to do nothing.)

Products via Universal Properties

The Cartesian product $A \times B$ is the categorical product: for any set X with maps $f : X \to A$ and $g : X \to B$, there is a unique map $\langle f, g \rangle : X \to A \times B$ making the projection triangles commute.
This "universal property" will reappear in many disguises throughout the course. It's a way of defining things by what they *do* rather than what they *are*.

Diagrams as Visual Equations

A *commutative diagram* is a picture representing equations between composites of functions. Consider:



This diagram **commutes** if $g \circ f = h$. In words: "going from A to C via B gives the same result as going directly."

A more complex example—a commutative square:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle g} \\ C & \xrightarrow[k]{} & D \end{array}$$

This commutes if $g \circ f = k \circ h$. Both paths from A to D give the same composite.
Why diagrams? Complex equations become pictures you can *see*. Proofs become *path-finding*: to show two composites are equal, find paths in a commuting diagram connecting them.

Exercises: Arrows and Diagrams

1. Draw the diagram representing $h \circ g \circ f = k$ for functions $f : A \to B$, $g : B \to C$, $h : C \to D$, and $k : A \to D$.
2. Consider the commutative square above. Write down the equation this diagram represents.
3. If the square above commutes, and we also have $m : D \to E$, draw the extended diagram. What new equation(s) can we derive involving m?
4. True or False: If $g \circ f = g \circ f'$, then $f = f'$. Either prove this or give a counterexample with small sets. (Hint: think about what properties g would need.)
5. The identity law says $f \circ \mathrm{id}_A = f$. Draw this as a commutative triangle.
6. Associativity says $(h \circ g) \circ f = h \circ (g \circ f)$. Explain why this means we can unambiguously write $h \circ g \circ f$ without parentheses.
7. Let $f : \{1, 2\} \to \{a, b, c\}$ and $g : \{a, b, c\} \to \{x, y\}$ be specific (not arbitrary) functions. How many functions $h : \{1, 2\} \to \{x, y\}$ make the triangle commute (i.e., $g \circ f = h$)? Is it always exactly one?
8. **Diagram chase:** Suppose triangles $g \circ f = h$ and $k \circ g = \ell$ both commute. Prove that $k \circ h = \ell \circ f$ by manipulating composites.

Practice

1. Convert $\{x \in \mathbb{Z} : x^2 < 10\}$ into roster notation. (Careful: what integers have squares less than 10?)
2. Prove $A \setminus B = A \cap B^c$ using the element method.
3. List $\mathcal{P}(\{a, b, c\})$ and verify its size. (There should be 8 subsets. If you got 7, you forgot one—which?)
4. Give a counterexample showing $A \cap (B \setminus C) = (A \cap B) \setminus C$ can fail. (Hint: think about where C "hits" differently on each side.)
5. Prove the other absorption law: $A \cap (A \cup B) = A$.

6. If A has 4 elements and B has 3 elements, what is the maximum size of $A \cap B$? The minimum?

7. Prove that for any sets A, B, C: $(A \cap B) \cup (A \cap C) \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C) \cap (B \cup C)$. (This is a subset, not equality—can you find sets where it's strict?)

3 Week 2: Functions and Cardinality

Or: "How to match things up, and what happens when you can't"

Reading

Epp §7.1–7.4.
Category theory companion: Week 1–2 (`category_theory_companion.pdf`).

Why Functions?

You've been using functions since algebra class, but we haven't really said what they *are*. In calculus, a function is usually "a rule that assigns outputs to inputs"—but what counts as a "rule"? Can we have a function defined differently on every input with no pattern at all?

The answer in discrete mathematics is: yes, absolutely. A function is just a complete assignment of outputs to inputs. No formula required. If I hand you a lookup table that assigns a color to each person in the class, that's a function—even if there's no pattern, even if I just made it up arbitrarily.

This abstraction lets us reason about functions in full generality, which becomes important when we start asking questions like: "Are there more real numbers than integers?" The answer requires thinking carefully about what it means to "pair up" two infinite sets.

Learning objectives

- Identify domain, codomain, and image of a function.
- Distinguish injective, surjective, and bijective functions.
- Use inverses and composition to solve functional equations.
- Compare sizes of sets using countability arguments.
- Apply the pigeonhole principle to function problems.

Key definitions and facts

What Is a Function, Really?

Definition 3.1 (Function)**.** A **function** $f : A \to B$ assigns to each element $a \in A$ exactly one element $f(a) \in B$.

- **Domain:** the set A (where inputs come from)
- **Codomain:** the set B (where outputs are allowed to go)
- **Image/Range:** $\{f(a) : a \in A\} \subseteq B$ (the outputs that actually get hit)

The key phrase is "exactly one." Every input must have an output (the function is defined everywhere on A), and each input has only one output (no branching). This rules out things like $\pm\sqrt{x}$, which gives two values for each positive input.

Notice the distinction between codomain and image. If $f : \mathbb{R} \to \mathbb{R}$ is defined by $f(x) = x^2$, the codomain is all of $\mathbb{R}$, but the image is only $[0, \infty)$—the squares. The codomain is where outputs *could* land; the image is where they *actually* land.

Definition 3.2 (Image and preimage of sets)**.** Let $f : A \to B$ be a function. We can apply f to entire sets, not just elements:

- For $S \subseteq A$, the **image** of S under f is $f(S) = \{f(x) : x \in S\}$.
- For $T \subseteq B$, the **preimage** of T under f is $f^{-1}(T) = \{x \in A : f(x) \in T\}$.

Warning (Preimage vs. inverse). The notation $f^{-1}(T)$ for preimage does *not* require f to have an inverse function. The preimage is always defined—it's just asking "which inputs land in T?" Even if f isn't invertible, this question makes sense.

This notation is unfortunate because it looks like we're applying an inverse function, but we're not. Blame historical accident.

The Three Types of Functions

There are three properties a function can have, and they tell you about how completely and efficiently the function "covers" its codomain.

Definition 3.3 (Injective, surjective, bijective)**.** Let $f : A \to B$ be a function.

- f is **injective** (one-to-one) if different inputs always give different outputs.

  Formally: $f(x) = f(y)$ implies $x = y$.

  Intuitively: no two inputs collide at the same output. You could "reverse" the function on its image.

- f is **surjective** (onto) if every element of B gets hit by something.

  Formally: for every $b \in B$, there exists $a \in A$ with $f(a) = b$.

  Intuitively: the image equals the codomain. Nothing in B is left out.

- f is **bijective** if it is both injective and surjective.

  Intuitively: a perfect pairing. Every input goes to a unique output, and every possible output gets hit exactly once.

Why these names? "Injective" suggests that A is "injected" into B without overlap. "Surjective" suggests the function goes "sur" (French for "onto") all of B. "Bijective" is both: a two-way correspondence.

The old-fashioned terms "one-to-one" and "onto" are still common, but they're less systematic. I'll use both interchangeably.

Key Result

[Category-theoretic terminology] These three properties have categorical names that you'll see in the companion material:

- An injective function is a **monomorphism** ("mono" = one, because each output comes from at most one input)
- A surjective function is an **epimorphism** ("epi" = upon, because every output is "reached")
- A bijective function is an **isomorphism** ("iso" = equal, because it establishes a perfect

correspondence)

In categories beyond sets, these concepts generalize in subtle ways—monomorphisms and epimorphisms are defined by cancellation properties rather than element-wise conditions. But in **Set**, they coincide with injection and surjection.

Definition 3.4 (Inverse function)**.** If $f : A \to B$ is bijective, then there exists a unique function $f^{-1} : B \to A$ satisfying:

- $f^{-1}(f(a)) = a$ for all $a \in A$ (left inverse: undoes f)
- $f(f^{-1}(b)) = b$ for all $b \in B$ (right inverse: f undoes it)

Why does f need to be bijective? If f isn't injective, two inputs $a_1 \neq a_2$ might have $f(a_1) = f(a_2) = b$—then what should $f^{-1}(b)$ be? If f isn't surjective, some $b \in B$ has no preimage—then $f^{-1}(b)$ doesn't exist.

Composition and Its Properties

Proposition 3.1 (Composition preserves properties)**.** *Let $f : A \to B$ and $g : B \to C$.*

- *If f and g are injective, then $g \circ f$ is injective.*
- *If f and g are surjective, then $g \circ f$ is surjective.*
- *If f and g are bijective, then $g \circ f$ is bijective, with $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$.*

The inverse formula is worth staring at: $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$. To undo "first f, then g," you do "first g^{-1}, then f^{-1}." Like putting on and taking off socks and shoes.

Proposition 3.2 (Composition tests)**.** *Here's a useful diagnostic:*

- *If $g \circ f$ is injective, then f must be injective (but g might not be).*
- *If $g \circ f$ is surjective, then g must be surjective (but f might not be).*

The logic: if the whole pipeline doesn't have collisions, the first step can't have collisions either. If the whole pipeline hits everything, the last step must hit everything.

Comparing Infinite Sets

Here's where things get interesting. How do you decide if two infinite sets are "the same size"? You can't count to infinity and compare. Instead, we use functions.

Definition 3.5 (Countable and uncountable)**.** A set A is **countably infinite** if there is a bijection $A \leftrightarrow \mathbb{N}$. In other words, you can list its elements: $a_0, a_1, a_2, \ldots$ with nothing missing and no repeats.

A set is **countable** if it is finite or countably infinite.

A set is **uncountable** if it is not countable—too big to list.

This seems like it should make all infinite sets the same size, but it doesn't. Cantor's great discovery was that some infinities are bigger than others.

Theorem 3.1 (Countability results)**.**

- *$\mathbb{Z}$ is countable. (List: $0, 1, -1, 2, -2, 3, -3, \ldots$)*

- $\mathbb{Q}$ *is countable. (This is surprising! There are "so many" rationals.)*
- *The union of countably many countable sets is countable.*
- $\mathbb{R}$ *is* ***uncountable****. (Cantor's diagonal argument—see below.)*
- $|\mathcal{P}(\mathbb{N})| = |\mathbb{R}| > |\mathbb{N}|$*. The power set of the naturals has more elements than* $\mathbb{N}$ *itself.*

That last point is remarkable: even though $\mathbb{N}$ is infinite, its power set is "more infinite." And you can keep going: $\mathcal{P}(\mathcal{P}(\mathbb{N}))$ is bigger still. There's an infinite hierarchy of infinities.

Worked examples

Example 3.1 (Image and preimage computation)**.** Let $f : \mathbb{Z} \to \mathbb{Z}$ be defined by $f(n) = n^2 - 1$. Find $f(\{-2, 0, 3\})$ and $f^{-1}(\{0, 3, 8\})$.

Solution.

Image: Just compute f on each element:

- $f(-2) = (-2)^2 - 1 = 3$
- $f(0) = 0^2 - 1 = -1$
- $f(3) = 3^2 - 1 = 8$

So $f(\{-2, 0, 3\}) = \{-1, 3, 8\}$.

Preimage: Find all integers n with $f(n) \in \{0, 3, 8\}$. We solve $n^2 - 1 = k$ for each target:

- $n^2 - 1 = 0 \Rightarrow n^2 = 1 \Rightarrow n = \pm 1$
- $n^2 - 1 = 3 \Rightarrow n^2 = 4 \Rightarrow n = \pm 2$
- $n^2 - 1 = 8 \Rightarrow n^2 = 9 \Rightarrow n = \pm 3$

So $f^{-1}(\{0, 3, 8\}) = \{-3, -2, -1, 1, 2, 3\}$.

Notice the preimage has more elements than the target set—that's fine! Multiple inputs can map to the same output.

Example 3.2 (Injective but not surjective)**.** Let $f : \mathbb{N} \to \mathbb{N}$ be $f(n) = 2n$. Show f is injective but not surjective.

Proof.

Injective: Suppose $f(n) = f(m)$, i.e., $2n = 2m$. Dividing by 2, we get $n = m$. So different inputs give different outputs.

Not surjective: We need to find an element of $\mathbb{N}$ that nothing maps to. Consider $1 \in \mathbb{N}$. If $f(n) = 1$, then $2n = 1$, so $n = 1/2$. But $1/2 \notin \mathbb{N}$. So 1 is not in the image of f. □

This is an example of an infinite set being in bijection with a proper subset of itself—$\mathbb{N}$ bijects with the even naturals. This is actually a *characteristic* property of infinite sets: a set is infinite if and only if it bijects with a proper subset.

Example 3.3 (Composition test)**.** Prove: If $g \circ f$ is injective, then f is injective.

Proof. Suppose $f(x) = f(y)$. We want to show $x = y$.

Applying g to both sides: $g(f(x)) = g(f(y))$, i.e., $(g \circ f)(x) = (g \circ f)(y)$.

Since $g \circ f$ is injective, this implies $x = y$.

Therefore f is injective. □

The intuition: if the pipeline f then g doesn't have collisions, then f alone can't have collisions either—any collision in f would survive through g and show up in the composition.

Example 3.4 (Bijection between $\mathbb{Z}$ and $\mathbb{N}$)**.** Give a bijection between $\mathbb{Z}$ and $\mathbb{N}$.

Solution. We need to list all integers in a sequence: each integer appears exactly once. The obvious idea—$0, 1, 2, 3, \ldots$—misses all the negatives. We need to interleave:
Convention: Throughout these notes, $\mathbb{N} = \{0, 1, 2, \ldots\}$.

$$0, 1, -1, 2, -2, 3, -3, \ldots$$

The formula that does this:

$$f(n) = \begin{cases} 2n & \text{if } n \geq 0 \\ -2n - 1 & \text{if } n < 0 \end{cases}$$

Let's check: $f(0) = 0$, $f(-1) = 1$, $f(1) = 2$, $f(-2) = 3$, $f(2) = 4$, and so on. Nonnegative integers go to even naturals (including 0); negative integers go to odd naturals. Since every natural is either even or odd (but not both), this is a bijection.

Example 3.5 (Checking bijectivity)**.** Determine whether $f : \mathbb{R} \to \mathbb{R}$ defined by $f(x) = x^3$ is bijective.

Solution.

Injective: Suppose $f(a) = f(b)$, i.e., $a^3 = b^3$.

Unlike squares, cubes determine their base uniquely (even for negative numbers). Taking cube roots: $a = b$. So f is injective.

Surjective: For any $y \in \mathbb{R}$, we need x with $x^3 = y$.

Take $x = \sqrt[3]{y}$. Cube roots exist for all real numbers (including negatives—this is different from square roots!). Then $f(x) = (\sqrt[3]{y})^3 = y$. So f is surjective.

Since f is both injective and surjective, f is bijective. The inverse is $f^{-1}(y) = \sqrt[3]{y}$.

Example 3.6 (Cantor's diagonal argument)**.** Prove that the interval $(0, 1)$ is uncountable.

Proof. Suppose for contradiction that $(0, 1)$ is countable. Then we can list all its elements:

$$r_1 = 0.d_{11}d_{12}d_{13}\ldots, \quad r_2 = 0.d_{21}d_{22}d_{23}\ldots, \quad r_3 = 0.d_{31}d_{32}d_{33}\ldots, \quad \ldots$$

where d_{ij} is the jth decimal digit of r_i.

Now construct a new number $x = 0.e_1e_2e_3\ldots$ by going down the diagonal and changing each digit:

$$e_n = \begin{cases} 5 & \text{if } d_{nn} \neq 5 \\ 6 & \text{if } d_{nn} = 5 \end{cases}$$

Then $x \in (0, 1)$ (it's a decimal between 0 and 1), but $x \neq r_n$ for *any* n—because x and r_n differ in the nth decimal place.

This contradicts our assumption that we listed *all* elements of $(0, 1)$.

Therefore $(0, 1)$ is uncountable. □

This is Cantor's diagonal argument, and it's one of the most important proofs in mathematics. It shows there are "more" real numbers than natural numbers, even though both sets are infinite. The proof generalizes: for any set A, the power set $\mathcal{P}(A)$ is strictly larger than A.

Key Result

[Most functions are incomputable] Here's a striking application of cardinality to computer science: **almost all** functions from $\mathbb{N}$ to $\mathbb{N}$ cannot be computed by any program.

The argument is beautifully simple, requiring only two facts about cardinality.

Fact 1: There are uncountably many functions $\mathbb{N} \to \mathbb{N}$.

Consider the set of all functions $f : \mathbb{N} \to \mathbb{N}$. By the general formula for function spaces, $|A \to B| = |B|^{|A|}$, so $|\mathbb{N} \to \mathbb{N}| = \aleph_0^{\aleph_0}$. We'll show this equals $|\mathcal{P}(\mathbb{N})| = 2^{\aleph_0} = |\mathbb{R}|$.

For any subset $S \subseteq \mathbb{N}$, define its *characteristic function* $\chi_S : \mathbb{N} \to \mathbb{N}$ by:

$$\chi_S(n) = \begin{cases} 1 & \text{if } n \in S \\ 0 & \text{if } n \notin S \end{cases}$$

This gives an injection from $\mathcal{P}(\mathbb{N})$ into $(\mathbb{N} \to \mathbb{N})$: different subsets have different characteristic functions. So $|\mathcal{P}(\mathbb{N})| \leq |\mathbb{N} \to \mathbb{N}|$.

Conversely, we can encode any function $f : \mathbb{N} \to \mathbb{N}$ as a subset of $\mathbb{N} \times \mathbb{N}$ (its graph), and $|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$, so $|\mathbb{N} \to \mathbb{N}| \leq |\mathcal{P}(\mathbb{N} \times \mathbb{N})| = |\mathcal{P}(\mathbb{N})|$.

Therefore $|\mathbb{N} \to \mathbb{N}| = \aleph_0^{\aleph_0} = |\mathcal{P}(\mathbb{N})| = 2^{\aleph_0} = |\mathbb{R}|$, which is uncountable.

Fact 2: There are only countably many programs.

A program is a finite string of symbols from some finite alphabet (ASCII characters, say, or the syntax of your favorite programming language). How many such strings are there?

- There are finitely many strings of length 0 (just the empty string).
- There are finitely many strings of length 1 (one for each alphabet symbol).
- There are finitely many strings of length 2, length 3, and so on.

The set of all finite strings is the union:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$$

This is a countable union of finite sets, which is countable. (In fact, you can list strings in "shortlex" order: all length-0 strings, then all length-1 strings alphabetically, then length-2, and so on.)

The Conclusion.

Every computable function is computed by some program. So the set of computable functions is at most as large as the set of programs:

$$|\{\text{computable functions } \mathbb{N} \to \mathbb{N}\}| \leq |\{\text{programs}\}| = \aleph_0$$

But we just showed $|\{\text{all functions } \mathbb{N} \to \mathbb{N}\}| = 2^{\aleph_0}$, which is strictly larger than $\aleph_0$.

Therefore: there are uncountably many functions, but only countably many programs. The computable functions are a *measure-zero* subset of all functions. In a very precise sense, if you picked a random function $\mathbb{N} \to \mathbb{N}$, the probability that it's computable is zero.

This argument doesn't tell us *which* functions are incomputable—for that, you need specific examples like the halting problem. But it tells us that incomputability is the norm, not the exception. The functions we can actually compute are infinitesimally rare among all the functions that mathematically exist.

Example 3.7 (Surjective composition test)**.** Let $f : A \to B$ and $g : B \to C$. Prove that if $g \circ f$ is surjective, then g is surjective.

Proof. Let $c \in C$. We need to find some $b \in B$ with $g(b) = c$.

Since $g \circ f$ is surjective, there exists $a \in A$ with $(g \circ f)(a) = c$, i.e., $g(f(a)) = c$.
Let $b = f(a)$. Then $b \in B$ and $g(b) = c$.
So every element of C is hit by g, meaning g is surjective. □

Note: this doesn't mean f is surjective! The element $b = f(a)$ we found might not be every element of B—just enough to cover what we need.

Going Deeper: The Art of Diagram Chasing

Building on Week 1's introduction to diagrams, we now develop *diagram chasing* as a proof technique and encounter our first *universal property*.
For more detail, see the Category Theory Companion, Week 1–2.

Diagram Chasing as Proof

When a diagram commutes, we can prove equations between composites by finding different paths. The technique is almost mechanical: follow arrows, use commutativity, conclude equality.

Example. Suppose this square commutes (meaning $g \circ f = k \circ h$):

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle g} \\ C & \xrightarrow[k]{} & D \end{array}$$

If we extend with another arrow $m : D \to E$, we automatically get:

$$m \circ g \circ f = m \circ k \circ h$$

Because $g \circ f = k \circ h$, so composing with m on the left preserves the equality.

Cancellation Properties

Injective functions are "left-cancellable":

$$f \circ g = f \circ h \implies g = h \quad \text{(when } f \text{ is injective)}$$

Surjective functions are "right-cancellable":

$$g \circ f = h \circ f \implies g = h \quad \text{(when } f \text{ is surjective)}$$

These can be drawn as diagrams:

$$X \underset{h}{\overset{g}{\rightrightarrows}} A \xrightarrow{f} B \quad \text{(left cancellation)} \qquad A \xrightarrow{f} B \underset{h}{\overset{g}{\rightrightarrows}} X \quad \text{(right cancellation)}$$

Monomorphisms, Epimorphisms, and Isomorphisms

We introduced the categorical names earlier. Now let's see why category theorists use different definitions than the set-theoretic ones.
A **monomorphism** is a map $f : A \to B$ that is *left-cancellable*: if $f \circ g = f \circ h$, then $g = h$.
In **Set**, this is equivalent to being injective.

An **epimorphism** is a map $f : A \to B$ that is *right-cancellable*: if $g \circ f = h \circ f$, then $g = h$. In **Set**, this is equivalent to being surjective.
An **isomorphism** is a map $f : A \to B$ for which there exists $g : B \to A$ with $g \circ f = \mathrm{id}_A$ and $f \circ g = \mathrm{id}_B$. In **Set**, isomorphisms are exactly the bijections.
Why the cancellation definitions? Because they make sense in any category, even when there are no "elements" to talk about. In **Set**, the definitions coincide with injection/surjection. But in other categories (groups, rings, topological spaces), they can differ in surprising ways.

Sections and Retractions

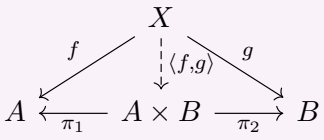
We can have one-sided inverses:

- A **section** (right inverse) of $f : A \to B$ is a map $s : B \to A$ with $f \circ s = \mathrm{id}_B$. If f has a section, f must be an epimorphism (surjective in **Set**).
- A **retraction** (left inverse) of $f : A \to B$ is a map $r : B \to A$ with $r \circ f = \mathrm{id}_A$. If f has a retraction, f must be a monomorphism (injective in **Set**).

In **Set**, a function has a section iff it's surjective (you can "pick" a preimage for each output), and has a retraction iff it's injective and A is nonempty (you can "invert" on the image and send everything else somewhere).

Universal Property of Products

The Cartesian product $A \times B$ isn't just "pairs of elements"—it's the *unique* (up to isomorphism) set that satisfies a certain property.
Universal property: For any set X with functions $f : X \to A$ and $g : X \to B$, there exists a **unique** function $\langle f, g \rangle : X \to A \times B$ making this diagram commute:



The function is $\langle f, g \rangle(x) = (f(x), g(x))$. The universal property says: if you want to map into a product, you just need to say where each component goes.
Why uniqueness matters: If someone claims to have another function $h : X \to A \times B$ with $\pi_1 \circ h = f$ and $\pi_2 \circ h = g$, you can immediately conclude $h = \langle f, g \rangle$. Uniqueness lets you prove equality by verifying a property, not by comparing definitions.

Exercises: Diagram Chasing

1. Let $f : A \to B$ be injective. If $f \circ g = f \circ h$ for $g, h : X \to A$, prove $g = h$ element-by-element.
2. Draw the diagram expressing "f is left-cancellable."
3. If $g \circ f$ is injective, prove f is injective.
4. If $g \circ f$ is injective, must g be injective? Prove or give a counterexample.

5. If $g \circ f$ is surjective, prove g is surjective.
6. If $g \circ f$ is surjective, must f be surjective? Prove or give a counterexample.
7. Let $A = \{1, 2\}$, $B = \{a, b, c\}$, $X = \{*\}$. If $f(*) = 1$ and $g(*) = b$, what is $\langle f, g \rangle(*)$?
8. For $A = \{1, 2\}$, $B = \{a, b\}$, $X = \{x, y\}$ with $f(x) = 1$, $f(y) = 2$, $g(x) = a$, $g(y) = b$, write out $\langle f, g \rangle$ explicitly.
9. Suppose $h, h' : X \to A \times B$ both satisfy $\pi_1 \circ h = \pi_1 \circ h' = f$ and $\pi_2 \circ h = \pi_2 \circ h' = g$. Prove $h = h'$.
10. If $f : A \to B$ has both a left inverse g and a right inverse h, prove $g = h$. (Hint: compute $g \circ f \circ h$ two ways.)

Practice

1. Give an explicit bijection between $\mathbb{Z}$ and $\mathbb{N}$. (If you use the one from the notes, verify it's actually a bijection.)
2. Decide whether $f(x) = x^3$ from $\mathbb{R}$ to $\mathbb{R}$ is bijective and justify. What about $f(x) = x^3$ from $\mathbb{R}$ to $\mathbb{R}_{\geq 0}$?
3. Prove that if $g \circ f$ is injective, then f is injective. (Don't just cite the theorem—write out the proof.)
4. Prove that a finite set cannot be in bijection with a proper subset of itself. (This fails for infinite sets!)
5. Prove: $f : A \to B$ is injective if and only if there exists $g : B \to A$ with $g \circ f = \mathrm{id}_A$. (Assume A is nonempty.)
6. Prove: $f : A \to B$ is surjective if and only if there exists $g : B \to A$ with $f \circ g = \mathrm{id}_B$.
7. Show that the set of all *finite* subsets of $\mathbb{N}$ is countable. (Hint: can you encode a finite subset as a single natural number?)

4 Week 3: Relations and Modular Arithmetic

Or: "When 7 and 2 are secretly the same number"

Reading

Epp §8.1–8.5.
Category theory companion: Week 3 (`category_theory_companion.pdf`).

Why Relations?

So far we've talked about sets and functions. Functions are special—every input goes to exactly one output. But lots of interesting relationships aren't like that. "Is a sibling of" isn't a function (you can have multiple siblings). "Divides" isn't a function (3 divides many numbers). "Is the same color as" isn't a function (many things share a color).

Relations are the general framework for talking about how elements of sets can be connected. They turn out to be everywhere: equivalence relations give us a way to identify things that are "the same in some sense," and partial orders give us hierarchies and dependencies.

And modular arithmetic? That's what happens when you decide that some numbers should be treated as equivalent—and it's the mathematics that makes computer science work, from hash tables to cryptography.

Learning objectives

- Describe a relation using sets, matrices, or digraphs.
- Test whether a relation is reflexive, symmetric, or transitive.
- Form equivalence classes and connect them to partitions.
- Compute congruences and modular inverses.
- Apply the extended Euclidean algorithm.

Key definitions and facts

What Is a Relation?

Definition 4.1 (Relation)**.** A **relation** from set A to set B is a subset $R \subseteq A \times B$. We write aRb or $(a, b) \in R$ to indicate that a is related to b.

A relation *on* A is a relation from A to itself (a subset of $A \times A$).

That's it. A relation is just a set of pairs. No requirements about every element having something related to it, or having at most one thing related. Just: which pairs are "in" the relation?

Properties of Relations

Relations can have various nice properties. Here are the important ones:

Definition 4.2 (Properties of relations)**.** Let R be a relation on set A.

- R is **reflexive** if aRa for all $a \in A$. (Everything is related to itself.)

- R is **symmetric** if aRb implies bRa for all $a, b \in A$. (If a relates to b, then b relates to a.)
- R is **antisymmetric** if aRb and bRa together imply $a = b$. (The only way to have mutual relation is to be the same element.)
- R is **transitive** if aRb and bRc imply aRc. (Chains collapse.)
- R is **irreflexive** if $\neg(aRa)$ for all $a \in A$. (Nothing relates to itself.)
- R is **total** (or **connex**) if for all $a, b \in A$, either aRb or bRa. (Everything is comparable.)

Common Mistake

Symmetric and antisymmetric are not opposites! A relation can be both (like equality: $a = b$ and $b = a$ implies $a = b$, trivially). A relation can be neither. The identity relation $\{(a, a) : a \in A\}$ is both symmetric and antisymmetric.

Equivalence Relations

The combination reflexive + symmetric + transitive is special enough to get its own name.

Definition 4.3 (Equivalence relation)**.** A relation R on a set A is an **equivalence relation** if it is reflexive, symmetric, and transitive.

For an equivalence relation $\sim$, the **equivalence class** of a is:

$$[a] = \{x \in A : x \sim a\}$$

This is the set of all elements equivalent to a.

Equivalence relations are the mathematical way of saying "these things should be considered the same." Fractions: $1/2$ and $2/4$ are equivalent. Angles: $0°$ and $360°$ are equivalent. Integers mod 5: 7 and 2 are equivalent.

Theorem 4.1 (Equivalence classes partition)**.** *If $\sim$ is an equivalence relation on A, then:*

1. *Every element belongs to exactly one equivalence class.*
2. *Two equivalence classes are either identical or disjoint (no partial overlap).*
3. *The equivalence classes partition A: $A = \bigsqcup_{a \in A} [a]$.*

This is a beautiful result: equivalence relations and partitions are two views of the same thing. Give me an equivalence relation, I'll show you a partition. Give me a partition, I'll define an equivalence relation (two things are equivalent iff they're in the same piece).

Quotienting: The Art of Controlled Forgetting

Here's the deeper idea behind equivalence relations: they let us *forget* certain distinctions while remembering others. The resulting structure—the **quotient set**—is often exactly what we need.

Definition 4.4 (Quotient set)**.** If $\sim$ is an equivalence relation on A, the **quotient set** (or **quotient of A by $\sim$**) is:

$$A/\sim = \{[a] : a \in A\}$$

This is the set of equivalence classes. Its elements are not elements of A, but *sets* of elements of A that we've decided to treat as identical.

The notation $A/\sim$ is read "A mod $\sim$" or "A quotiented by $\sim$." Think of it as "A, but with equivalent elements collapsed together."

Example 4.1 (Integers mod n)**.** The prototypical example: $\mathbb{Z}/\equiv_n$ (usually written $\mathbb{Z}_n$ or $\mathbb{Z}/n\mathbb{Z}$) is the integers where we've identified numbers that differ by multiples of n. The elements are $[0], [1], \ldots, [n-1]$, and arithmetic on these classes is well-defined.

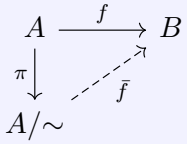
Example 4.2 (Rational numbers)**.** Here's one you've used since grade school without thinking about it. A fraction a/b is really an equivalence class of pairs: $\frac{1}{2}$ and $\frac{2}{4}$ are the same rational number because $(1,2) \sim (2,4)$ under the relation $(a,b) \sim (c,d) \iff ad = bc$. The rationals $\mathbb{Q}$ are a quotient of $\mathbb{Z} \times (\mathbb{Z} \setminus \{0\})$.

Definition 4.5 (Quotient map)**.** The **quotient map** (or **canonical projection**) is the function $\pi : A \to A/\sim$ defined by $\pi(a) = [a]$. It sends each element to its equivalence class.

This function is always surjective (every class has a representative). It's injective only when $\sim$ is equality (otherwise distinct elements can map to the same class).

Key Result

[The universal property of quotients] The quotient map $\pi : A \to A/\sim$ has a crucial property: if $f : A \to B$ is any function that "respects" the equivalence relation (meaning $a \sim a'$ implies $f(a) = f(a')$), then f factors uniquely through π:



There exists a unique $\bar{f} : A/\sim \to B$ with $f = \bar{f} \circ \pi$. We define $\bar{f}([a]) = f(a)$, and this is well-defined precisely because f respects $\sim$.

In plain English: if a function gives the same output for equivalent inputs, it's really a function on equivalence classes, not on elements.

This universal property explains why quotients are so useful: they're the "right" way to define functions that don't distinguish between equivalent elements.

Example 4.3 (Modular arithmetic, revisited)**.** Why is it okay to do arithmetic in $\mathbb{Z}_n$? Addition and multiplication on $\mathbb{Z}$ respect congruence: if $a \equiv a'$ and $b \equiv b'$, then $a + b \equiv a' + b'$ and $ab \equiv a'b'$. The operations "descend" to well-defined operations on $\mathbb{Z}/n\mathbb{Z}$ by the universal property.

Common Mistake

Confusing elements with equivalence classes. The elements of $A/\sim$ are sets, not elements of A. When we write $[a] = [b]$, we mean $a \sim b$, which is a statement about a and b in A. But $[a]$ itself is a single element of the quotient. This takes some getting used to.

Partial Orders

Different combination: reflexive + antisymmetric + transitive. This gives hierarchies instead of equivalences.

Definition 4.6 (Partial order)**.** A relation R on A is a **partial order** if it is reflexive, antisymmetric, and transitive. The pair (A, R) is called a **partially ordered set** (poset).

Definition 4.7 (Total order)**.** A partial order $\leq$ on A is a **total order** (or **linear order**) if for all $a, b \in A$, either $a \leq b$ or $b \leq a$.

The usual $\leq$ on numbers is a total order. Divisibility on positive integers is a partial order but not total ($2 \nmid 3$ and $3 \nmid 2$). Subset inclusion on $\mathcal{P}(X)$ is a partial order.

Posets and Hasse diagrams

Definition 4.8 (Minimal/maximal vs least/greatest)**.** Let $(P, \leq)$ be a poset. These concepts sound similar but differ:

- $m \in P$ is **minimal** if there is no $x \in P$ with $x < m$. (Nothing is strictly below it.)
- $m \in P$ is **maximal** if there is no $x \in P$ with $m < x$. (Nothing is strictly above it.)
- $\ell \in P$ is the **least element** if $\ell \leq x$ for all $x \in P$. (It's below everything.)
- $g \in P$ is the **greatest element** if $x \leq g$ for all $x \in P$. (It's above everything.)

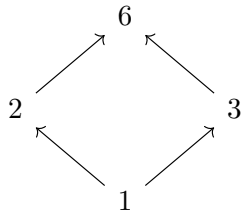
Least/greatest elements, if they exist, are unique. But there can be multiple minimal/maximal elements.

The difference: minimal means "nothing below," least means "below everything." In a total order these coincide, but in a partial order, you can have multiple minimal elements that are incomparable to each other.

Definition 4.9 (Hasse diagram)**.** For a finite poset, the **Hasse diagram** is a visual representation:

- Draw one vertex per element.
- Draw an edge upward from a to b if b *covers* a (meaning $a < b$ and there's nothing between them).
- Omit self-loops and edges implied by transitivity.

Example 4.4. The divisibility poset on $\{1, 2, 3, 6\}$ has Hasse diagram:



Here 1 is least (it divides everything); 6 is greatest (everything divides it); 2 and 3 are incomparable (neither divides the other).

Definition 4.10 (Bounds, meet, join, lattice)**.** Let $S \subseteq P$ be a subset of a poset.

- u is an **upper bound** of S if $x \leq u$ for all $x \in S$.

- ℓ is a **lower bound** of S if $\ell \leq x$ for all $x \in S$.
- The **least upper bound** (lub, join, supremum) of $\{a, b\}$ is denoted $a \vee b$.
- The **greatest lower bound** (glb, meet, infimum) of $\{a, b\}$ is denoted $a \wedge b$.

A poset is a **lattice** if every pair has both a meet and a join.

In divisibility, $a \wedge b = \gcd(a, b)$ and $a \vee b = \text{lcm}(a, b)$. In subset inclusion, $A \wedge B = A \cap B$ and $A \vee B = A \cup B$.

Modular Arithmetic

Now for the payoff: treating numbers as equivalent when they differ by a multiple of n.

Definition 4.11 (Congruence modulo n)**.** For integers a, b and positive integer n, we say a is **congruent** to b modulo n, written $a \equiv b \pmod{n}$, if n divides $a - b$. Equivalently:

$$a \equiv b \pmod{n} \iff a \bmod n = b \bmod n \iff \exists k \in \mathbb{Z} : a = b + kn$$

So $17 \equiv 3 \pmod{7}$ because $17 - 3 = 14 = 7 \times 2$. And $-2 \equiv 5 \pmod{7}$ because $-2 - 5 = -7 = 7 \times (-1)$.

Theorem 4.2 (Congruence is an equivalence relation)**.** *For any positive integer n, congruence modulo n is an equivalence relation on $\mathbb{Z}$. The equivalence classes are the **residue classes**:*

$$[0], [1], [2], \ldots, [n-1]$$

The set of residue classes is denoted $\mathbb{Z}_n$ or $\mathbb{Z}/n\mathbb{Z}$.

This is why clock arithmetic works: on a 12-hour clock, 14:00 is the same as 2:00 because $14 \equiv 2 \pmod{12}$.

Theorem 4.3 (Modular arithmetic preserves operations)**.** *For all $a, b, c, d \in \mathbb{Z}$ and positive integer n:*

1. *If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $a + c \equiv b + d \pmod{n}$.*
2. *If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $ac \equiv bd \pmod{n}$.*
3. *If $a \equiv b \pmod{n}$, then $a^k \equiv b^k \pmod{n}$ for all $k \geq 0$.*

This is crucial: we can do arithmetic within equivalence classes. We don't need to keep track of the actual numbers, just their remainders. This makes modular arithmetic computationally efficient—compute with small numbers, get correct answers about big numbers.

Definition 4.12 (Modular inverse)**.** For $a \in \mathbb{Z}_n$, the **modular inverse** of a modulo n is an integer b such that $ab \equiv 1 \pmod{n}$. We write $b = a^{-1} \pmod{n}$.

Theorem 4.4 (Existence of modular inverse)**.** *The integer a has a modular inverse modulo n if and only if $\gcd(a, n) = 1$.*

So 3 has an inverse mod 7 (since $\gcd(3, 7) = 1$), but 4 has no inverse mod 6 (since $\gcd(4, 6) = 2 \neq 1$).

The Euclidean Algorithm

Theorem 4.5 (Division algorithm)**.** *For any integers a and b with $b > 0$, there exist unique integers q (quotient) and r (remainder) such that:*

$$a = bq + r \quad \textit{and} \quad 0 \leq r < b$$

Theorem 4.6 (Euclidean algorithm)**.** *For positive integers a and b, $\gcd(a, b)$ can be computed by repeatedly applying: $\gcd(a, b) = \gcd(b, a \bmod b)$, until one argument becomes 0. The last nonzero remainder is the gcd.*

This is fast—the number of steps is at most $O(\log(\min(a, b)))$.

Theorem 4.7 (Extended Euclidean algorithm (Bézout's identity))**.** *For any positive integers a and b, there exist integers x and y such that:*

$$ax + by = \gcd(a, b)$$

The extended Euclidean algorithm computes x and y by working backwards through the division steps.

Proof Strategy

[Finding modular inverses] To find $a^{-1} \pmod{n}$ when $\gcd(a, n) = 1$:

1. Use the extended Euclidean algorithm to find x, y with $ax + ny = 1$.
2. Then $a^{-1} \equiv x \pmod{n}$.

Why? Because $ax + ny = 1$ means $ax \equiv 1 \pmod{n}$.

Representations of relations

Definition 4.13 (Matrix representation)**.** A relation R on a finite set $A = \{a_1, \ldots, a_n\}$ can be represented by an $n \times n$ **relation matrix** M where:

$$M_{ij} = \begin{cases} 1 & \text{if } a_i R a_j \\ 0 & \text{otherwise} \end{cases}$$

Proposition 4.1 (Reading properties from the matrix)**.** *For a relation matrix M:*

- *R is reflexive iff all diagonal entries are 1.*
- *R is symmetric iff $M = M^T$ (the matrix is symmetric).*
- *R is antisymmetric iff $M_{ij} = 1$ and $M_{ji} = 1$ together imply $i = j$.*

Definition 4.14 (Digraph representation)**.** A relation R on A can be represented as a directed graph (digraph) with vertices A and a directed edge from a to b whenever aRb.

Closures

Sometimes a relation is *almost* reflexive or transitive. Closures let us "fix it up."

Definition 4.15 (Closure)**.** The **reflexive closure** of R is the smallest reflexive relation containing R: $R \cup \{(a, a) : a \in A\}$.

The **symmetric closure** of R is the smallest symmetric relation containing R: $R \cup R^{-1}$ where $R^{-1} = \{(b, a) : (a, b) \in R\}$.

The **transitive closure** of R, denoted R^+, is the smallest transitive relation containing R.

Theorem 4.8 (Computing transitive closure)**.** *$R^+ = R \cup R^2 \cup R^3 \cup \cdots$ where R^n is the n-fold composition of R. For finite sets with $|A| = n$, we have $R^+ = R \cup R^2 \cup \cdots \cup R^n$.*

The transitive closure captures reachability: aR^+b means you can get from a to b by following R-edges.

Worked examples

Example 4.5 (Verifying congruence is an equivalence relation)**.** Show that congruence modulo n is an equivalence relation on $\mathbb{Z}$.

Proof.

- **Reflexive:** For any $a \in \mathbb{Z}$, we have $a - a = 0 = n \cdot 0$, so $n \mid (a - a)$. Thus $a \equiv a \pmod{n}$.
- **Symmetric:** Suppose $a \equiv b \pmod{n}$. Then $n \mid (a - b)$, so $a - b = nk$ for some integer k. Then $b - a = -nk = n(-k)$, so $n \mid (b - a)$. Thus $b \equiv a \pmod{n}$.
- **Transitive:** Suppose $a \equiv b \pmod{n}$ and $b \equiv c \pmod{n}$. Then $a - b = nj$ and $b - c = nk$ for integers j, k. Adding: $a - c = nj + nk = n(j + k)$. So $n \mid (a - c)$, and $a \equiv c \pmod{n}$. □

Example 4.6 (Finding a modular inverse)**.** Find the inverse of 3 modulo 7.

Solution. We need x such that $3x \equiv 1 \pmod{7}$.

Method 1 (Trial): Just try small values: $3 \cdot 1 = 3$, $3 \cdot 2 = 6$, $3 \cdot 3 = 9 \equiv 2$, $3 \cdot 4 = 12 \equiv 5$, $3 \cdot 5 = 15 \equiv 1 \pmod{7}$. So $3^{-1} \equiv 5 \pmod{7}$.

Method 2 (Extended Euclidean):

$$7 = 2 \cdot 3 + 1$$
$$1 = 7 - 2 \cdot 3 = 1 \cdot 7 + (-2) \cdot 3$$

So $(-2) \cdot 3 + 1 \cdot 7 = 1$, meaning $3^{-1} \equiv -2 \equiv 5 \pmod{7}$.

Example 4.7 (Equivalence classes from a relation)**.** Let $A = \{1, 2, 3, 4\}$ and $R = \{(1, 1), (1, 2), (2, 1), (2, 2), (3, 3), (3$
Show this is an equivalence relation and find the equivalence classes.

Solution. Check the three properties:

- **Reflexive:** $(1, 1), (2, 2), (3, 3), (4, 4) \in R$. ✓
- **Symmetric:** For each $(a, b) \in R$ with $a \neq b$, we need $(b, a) \in R$. We have $(1, 2)$ and $(2, 1)$; $(3, 4)$ and $(4, 3)$. ✓
- **Transitive:** Check chains: $(1, 2), (2, 1) \Rightarrow (1, 1) \in R$; $(3, 4), (4, 3) \Rightarrow (3, 3) \in R$; etc. All implied pairs are present. ✓

Equivalence classes: $[1] = [2] = \{1, 2\}$ and $[3] = [4] = \{3, 4\}$.

The partition is $\{\{1, 2\}, \{3, 4\}\}$—two blocks, each containing elements that are equivalent to each other.

Example 4.8 (Euclidean algorithm)**.** Find $\gcd(252, 105)$.

Solution.

$$\begin{aligned} 252 &= 2 \cdot 105 + 42 \\ 105 &= 2 \cdot 42 + 21 \\ 42 &= 2 \cdot 21 + 0 \end{aligned}$$

So $\gcd(252, 105) = 21$ (the last nonzero remainder).

Example 4.9 (Extended Euclidean algorithm)**.** Express $\gcd(252, 105)$ as a linear combination of 252 and 105.

Solution. Work backwards through the Euclidean algorithm steps:

$$\begin{aligned} 21 &= 105 - 2 \cdot 42 && \text{(from the second step)} \\ &= 105 - 2 \cdot (252 - 2 \cdot 105) && \text{(substitute for 42)} \\ &= 105 - 2 \cdot 252 + 4 \cdot 105 \\ &= 5 \cdot 105 + (-2) \cdot 252 \end{aligned}$$

So $21 = 5 \cdot 105 + (-2) \cdot 252$.

Check: $5 \times 105 - 2 \times 252 = 525 - 504 = 21$. ✓

Example 4.10 (Toy RSA)**.** This example illustrates RSA arithmetic (not secure for real use!). Let $p = 5$ and $q = 11$, so $n = pq = 55$ and $\phi(n) = (p - 1)(q - 1) = 4 \cdot 10 = 40$.

Choose public exponent $e = 3$ (coprime to 40). Find the private exponent d such that $ed \equiv 1 \pmod{40}$.

Solution. We need $3d \equiv 1 \pmod{40}$. Using extended Euclidean:

$$40 = 13 \cdot 3 + 1 \implies 1 = 40 - 13 \cdot 3$$

So $d \equiv -13 \equiv 27 \pmod{40}$. Check: $3 \cdot 27 = 81 = 2 \cdot 40 + 1 \equiv 1 \pmod{40}$. ✓

Encryption: To encrypt message $m = 12$, compute $c \equiv m^e \pmod{n}$:

$$c \equiv 12^3 = 1728 \equiv 1728 - 31 \cdot 55 = 23 \pmod{55}$$

Decryption: Compute $c^d \pmod{n}$. Using repeated squaring: $23^2 = 529 \equiv 34 \pmod{55}$, $23^4 \equiv 34^2 = 1156 \equiv 1 \pmod{55}$.

So $23^{27} = 23^{24} \cdot 23^3 = (23^4)^6 \cdot 23^3 \equiv 1^6 \cdot 12167 \equiv 12 \pmod{55}$.

We recover the original message! This is how RSA encryption works (with much larger primes).

Example 4.11 (A subtlety about relation properties)**.** Prove: If R is symmetric and transitive, and every element is related to *something*, then R is reflexive.

Proof. Let $a \in A$. By assumption, there exists some b such that aRb. By symmetry, bRa. By transitivity (using aRb and bRa), we get aRa. Since a was arbitrary, R is reflexive. □

This is a cute trick: you might think symmetric + transitive would be enough, but you also need "everything relates to something" to bootstrap reflexivity.

Advanced number theory

The following theorems are powerful tools for modular arithmetic, especially in cryptography.

Definition 4.16 (Euler's totient function)**.** For a positive integer n, **Euler's totient function** $\phi(n)$ counts the integers from 1 to n that are coprime to n:

$$\phi(n) = |\{k : 1 \leq k \leq n \text{ and } \gcd(k, n) = 1\}|$$

Theorem 4.9 (Computing $\phi(n)$)**.**

- *If p is prime:* $\phi(p) = p - 1$ *(everything except p is coprime to p)*
- *If p is prime:* $\phi(p^k) = p^{k-1}(p-1)$
- *If* $\gcd(m, n) = 1$*:* $\phi(mn) = \phi(m)\phi(n)$ *(multiplicative)*

In general, if $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$*, then:*

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right)$$

Example 4.12. Compute $\phi(12)$.

Solution. $12 = 2^2 \cdot 3$. So:

$$\phi(12) = 12 \cdot \left(1 - \frac{1}{2}\right) \cdot \left(1 - \frac{1}{3}\right) = 12 \cdot \frac{1}{2} \cdot \frac{2}{3} = 4$$

Verify: the integers from 1 to 12 coprime to 12 are $\{1, 5, 7, 11\}$. Indeed, there are 4.

Theorem 4.10 (Fermat's Little Theorem)**.** *If p is prime and* $\gcd(a, p) = 1$*, then:*

$$a^{p-1} \equiv 1 \pmod{p}$$

Equivalently, for any integer a: $a^p \equiv a \pmod{p}$.

Example 4.13. Compute $2^{100} \bmod 7$.

Solution. By Fermat's Little Theorem, $2^6 \equiv 1 \pmod{7}$ (since 7 is prime and $\gcd(2, 7) = 1$). Write $100 = 6 \cdot 16 + 4$. Then:

$$2^{100} = (2^6)^{16} \cdot 2^4 \equiv 1^{16} \cdot 16 \equiv 16 \equiv 2 \pmod{7}$$

Theorem 4.11 (Euler's Theorem)**.** *If* $\gcd(a, n) = 1$*, then:*

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

This generalizes Fermat's Little Theorem (when $n = p$ is prime, $\phi(p) = p - 1$).

Key Result

Euler's Theorem explains *why RSA works.* If $n = pq$ with primes p, q, and $ed \equiv 1 \pmod{\phi(n)}$, then for any message m coprime to n:

$$(m^e)^d = m^{ed} = m^{1+k\phi(n)} = m \cdot (m^{\phi(n)})^k \equiv m \cdot 1^k = m \pmod{n}$$

Decryption recovers the original message!

Theorem 4.12 (Chinese Remainder Theorem)**.** *Let $n_1, \ldots, n_k$ be pairwise coprime positive integers. Then for any $a_1, \ldots, a_k$, the system:*

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

has a unique solution modulo $N = n_1 n_2 \cdots n_k$.

Example 4.14. Solve: $x \equiv 2 \pmod 3$, $x \equiv 3 \pmod 5$.

Solution. From the first equation: $x = 2 + 3t$ for some integer t.
Substitute into the second: $2 + 3t \equiv 3 \pmod 5$, so $3t \equiv 1 \pmod 5$.
Find $3^{-1} \pmod 5$: $3 \cdot 2 = 6 \equiv 1$, so $t \equiv 2 \pmod 5$.
Thus $x = 2 + 3 \cdot 2 = 8$. Check: $8 = 2 + 2 \cdot 3 \equiv 2 \pmod 3$ and $8 = 3 + 1 \cdot 5 \equiv 3 \pmod 5$. ✓
So $x \equiv 8 \pmod{15}$.

Example 4.15 (Divisibility matrix)**.** Write the relation matrix for "divides" on $\{1, 2, 3, 4\}$.

Solution. $M_{ij} = 1$ iff $i \mid j$.

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reading properties: diagonal is all 1s (reflexive), matrix is not symmetric (not symmetric), and it's antisymmetric and transitive. So divisibility is a partial order.

Going Deeper: Preorders as Categories

This week we discover that the arrow-and-diagram language from Weeks 1–2 applies beautifully to familiar structures: preorders. This gives us concrete, easy-to-visualize categories to practice with.
For more detail, see the Category Theory Companion, Week 3.

Preorders You Already Know

A *preorder* is a set with a reflexive and transitive relation. You've seen many:

- $(\mathbb{N}, \leq)$: natural numbers with "less than or equal to"
- $(\mathcal{P}(X), \subseteq)$: subsets of X ordered by inclusion
- $(Div_n, |)$: divisors of n ordered by divisibility

Preorders ARE Categories

Here's the key insight: a preorder *is* a category. Given $(P, \leq)$:

- **Objects:** Elements of P
- **Arrows:** There is exactly one arrow $a \to b$ if $a \leq b$, and no arrow otherwise

- **Identity:** $a \leq a$ (reflexivity) gives the identity arrow $a \to a$
- **Composition:** $a \leq b$ and $b \leq c$ imply $a \leq c$ (transitivity)

This is called a *thin category*: between any two objects, there's at most one arrow. The existence of an arrow just records that $a \leq b$.

Monotone Functions = Functors

A function $f : P \to Q$ between preorders is **monotone** if $a \leq b$ implies $f(a) \leq f(b)$. In categorical language: if there's an arrow $a \to b$, then there's an arrow $f(a) \to f(b)$. The function preserves arrows—it's a **functor**.

Products and Coproducts in Preorders

The universal properties specialize nicely:

- **Product** of a and b = greatest lower bound (meet)
- **Coproduct** of a and b = least upper bound (join)

In divisibility: product = gcd, coproduct = lcm. In subset inclusion: product = $\cap$, coproduct = $\cup$.

Exercises: Preorders as Categories

1. Draw the divisibility preorder on $\{1, 2, 4, 8\}$ as a diagram with arrows. Is this a total order?
2. List all arrows in $(\mathcal{P}(\{1, 2\}), \subseteq)$. (Don't forget identities!)
3. Is $f(n) = n^2$ monotone from $(\mathbb{N}, \leq)$ to $(\mathbb{N}, \leq)$? What about from $(\mathbb{Z}, \leq)$ to $(\mathbb{Z}, \leq)$?
4. In $(\{1, 2, 3, 6\}, |)$, what is the product (glb) of 2 and 3? The coproduct (lub)?
5. Verify that $\gcd(6, 10)$ satisfies the universal property of products in divisibility.
6. Prove: if $f : P \to Q$ and $g : Q \to R$ are monotone, then $g \circ f$ is monotone.

Practice

1. For $n = 5$, list the equivalence classes of $\mathbb{Z}$ modulo n.
2. Find the inverse of 3 modulo 7 using the extended Euclidean algorithm.
3. Decide whether the relation xRy iff $x - y$ is even is an equivalence relation on $\mathbb{Z}$.
4. Prove that every equivalence relation on A defines a partition of A.
5. Let $R = \{(a, b) \in \mathbb{Z} \times \mathbb{Z} : a \leq b\}$. Which properties does R have: reflexive, symmetric, antisymmetric, transitive?
6. Find $\gcd(1071, 462)$ and express it as a linear combination.

7. Solve $17x \equiv 1 \pmod{43}$.

8. Let R be a relation on $\{1, 2, 3\}$ with matrix:

$$M = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

   Find the transitive closure R^+ and give its matrix.

9. Prove: If $a \equiv b \pmod{n}$ and $c \mid n$, then $a \equiv b \pmod{c}$.

10. Show that the intersection of two equivalence relations on A is an equivalence relation.

11. Compute $\phi(60)$.

12. Use Fermat's Little Theorem to find $5^{302} \bmod 7$.

13. Solve: $x \equiv 1 \pmod{4}$, $x \equiv 2 \pmod{5}$, $x \equiv 3 \pmod{7}$.

14. What are the last two digits of 7^{2024}?

15. Draw the Hasse diagram for divisibility on $\{1, 2, 3, 6, 12\}$. Identify minimal, maximal, least, and greatest elements.

16. In $(\mathcal{P}(\{1, 2, 3\}), \subseteq)$, compute $\{1, 2\} \wedge \{2, 3\}$ and $\{1, 2\} \vee \{2, 3\}$.

5 Week 4: Counting and Probability I

Or: "The surprisingly hard question of 'how many?"'

Reading

Epp §9.1–9.4, 9.9.
Category theory companion: Weeks 4–5 (`category_theory_companion.pdf`).

Why Counting?

Counting seems like it should be trivial—after all, you learned it before kindergarten. But counting *correctly* when things get complicated is surprisingly tricky. How many 8-character passwords are possible? How many ways can you seat 10 people at a round table? How likely is it that two people in a class of 30 share a birthday?

These questions require systematic techniques, and getting them wrong has consequences: underestimate password strength and you get hacked; miscount arrangements and your combinatorial proof falls apart; miscalculate probabilities and you make bad decisions.

This week introduces the fundamental counting principles. They're deceptively simple—multiplication, addition, inclusion-exclusion—but applying them correctly takes practice.

Learning objectives

- Build sample spaces and events for basic probability models.
- Apply the multiplication rule to count outcomes.
- Apply the addition rule and inclusion–exclusion for two or more sets.
- Use the pigeonhole principle to force collisions.
- Count permutations and arrangements with restrictions.

Key definitions and facts

Probability Basics

Definition 5.1 (Sample space and event)**.** A **sample space** S is the set of all possible outcomes of an experiment. An **event** is a subset $E \subseteq S$. The event E **occurs** if the actual outcome is in E.

Definition 5.2 (Probability (equally likely outcomes))**.** If all outcomes in a finite sample space S are equally likely, then for any event E:

$$P(E) = \frac{|E|}{|S|} = \frac{\text{number of favorable outcomes}}{\text{number of possible outcomes}}$$

This is why counting matters for probability: if you can count the favorable outcomes and the total outcomes, you can compute probabilities. The challenge is counting correctly.

Theorem 5.1 (Basic probability properties)**.** *For any events A, B in sample space S:*

1. $0 \leq P(A) \leq 1$ *(probabilities are between 0 and 1)*
2. $P(S) = 1$ *and* $P(\emptyset) = 0$ *(something happens; nothing is impossible)*

3. $P(A^c) = 1 - P(A)$ *where* $A^c = S \setminus A$ *(complement rule)*

4. *If* $A \cap B = \emptyset$*, then* $P(A \cup B) = P(A) + P(B)$ *(disjoint addition)*

The Fundamental Counting Principles

Theorem 5.2 (Multiplication rule (product rule))**.** *If a procedure can be broken into* k *successive steps, where:*

- *Step 1 can be done in* n_1 *ways*
- *Step 2 can be done in* n_2 *ways (regardless of step 1's outcome)*
- $\ldots$
- *Step* k *can be done in* n_k *ways*

then the total number of ways to complete the procedure is $n_1 \times n_2 \times \cdots \times n_k$*.*

The key requirement is independence: the number of ways to do each step must not depend on the choices made in previous steps. If step 2 has different numbers of options depending on what you chose in step 1, you need to be more careful.

Theorem 5.3 (Addition rule (sum rule))**.** *If a task can be done either by method* A *(in* n_1 *ways) or by method* B *(in* n_2 *ways), and the two methods are mutually exclusive (no overlap), then the total number of ways is* $n_1 + n_2$*.*

Theorem 5.4 (Inclusion-exclusion (two sets))**.** *For any finite sets* A *and* B*:*

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

For probabilities: $P(A \cup B) = P(A) + P(B) - P(A \cap B)$*.*

Why subtract the intersection? Because if you just add $|A|$ and $|B|$, you count the overlap twice. Subtracting $|A \cap B|$ corrects for the double-counting.

Theorem 5.5 (Inclusion-exclusion (three sets))**.**

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

Theorem 5.6 (Inclusion-exclusion (general))**.** *For finite sets* $A_1, \ldots, A_n$*:*

$$\left| \bigcup_{i=1}^{n} A_i \right| = \sum_i |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \cdots$$

The pattern alternates: add singles, subtract pairs, add triples, subtract quadruples, etc.

Conditional Probability and Independence

Definition 5.3 (Conditional probability)**.** If $P(B) > 0$, the probability of A given that B has occurred is:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Think of it as restricting the sample space to B, then asking how much of B is also in A.

Theorem 5.7 (Multiplication rule for probabilities)**.**

$$P(A \cap B) = P(A \mid B) \cdot P(B) = P(B \mid A) \cdot P(A)$$

Definition 5.4 (Independence)**.** Events A and B are **independent** if:

$$P(A \cap B) = P(A) \cdot P(B)$$

Equivalently (when $P(B) > 0$): $P(A \mid B) = P(A)$—knowing B happened doesn't change the probability of A.

Warning. Independent and disjoint are very different! If A and B are disjoint (no overlap), they are usually *not* independent—knowing A happened tells you B definitely didn't happen.

Bayes' Rule and Total Probability

Theorem 5.8 (Law of total probability)**.** *If $\{B_1, \ldots, B_k\}$ is a partition of S (disjoint, covering everything), then:*

$$P(A) = \sum_{i=1}^{k} P(A \mid B_i) \cdot P(B_i)$$

This is useful when computing $P(A)$ directly is hard, but computing it conditionally on different cases is easier.

Theorem 5.9 (Bayes' rule)**.**

$$P(B_j \mid A) = \frac{P(A \mid B_j) \cdot P(B_j)}{\sum_{i=1}^{k} P(A \mid B_i) \cdot P(B_i)}$$

Bayes' rule "flips" conditional probabilities. You know $P(A \mid B)$, but you want $P(B \mid A)$. This comes up constantly: you observe evidence (tested positive, saw smoke, etc.) and want to infer the cause.

Permutations

Definition 5.5 (Permutation)**.** A **permutation** of a set is an arrangement of its elements in a sequence. The number of permutations of n distinct objects is:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$$

By convention, $0! = 1$.

Definition 5.6 (r-permutation)**.** An **r-permutation** of n objects is an ordered arrangement of r objects chosen from n distinct objects. The count is:

$$P(n, r) = \frac{n!}{(n-r)!} = n \times (n-1) \times \cdots \times (n-r+1)$$

The Pigeonhole Principle

Theorem 5.10 (Pigeonhole principle (basic))**.** *If $n+1$ objects are placed into n boxes, then at least one box contains at least 2 objects.*

This is almost embarrassingly obvious, but it's surprisingly powerful for proving existence results.

Theorem 5.11 (Pigeonhole principle (generalized))**.** *If n objects are placed into k boxes, then at least one box contains at least $\lceil n/k \rceil$ objects.*

Key Result

The pigeonhole principle guarantees existence without telling you *which* box has multiple objects. It's a pure existence proof—powerful, but non-constructive.

Derangements

Definition 5.7 (Derangement)**.** A **derangement** of $\{1, 2, \ldots, n\}$ is a permutation with no fixed points: $\sigma(i) \neq i$ for all i. The number of derangements is denoted D_n.

Theorem 5.12 (Counting derangements)**.**

$$D_n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!} = n! \left(1 - 1 + \frac{1}{2!} - \frac{1}{3!} + \cdots + \frac{(-1)^n}{n!}\right)$$

Key Result

For large n: $D_n \approx n!/e$. The probability that a random permutation is a derangement approaches $1/e \approx 0.368$ as $n \to \infty$. About a third of all permutations have no fixed points.

Counting Strategies

Definition 5.8 (With vs. without replacement)**.**

- **With replacement:** After selecting an object, it goes back. Selections are independent.
- **Without replacement:** Once selected, an object is removed. Later selections have fewer choices.

Proposition 5.1 (Counting sequences)**.** *From n distinct elements:*

- *Sequences of length k* ***with replacement****: n^k*
- *Sequences of length k* ***without replacement****: $P(n,k) = \frac{n!}{(n-k)!}$*

Proof Strategy

[Complement counting] Sometimes it's easier to count what you *don't* want:

$$|\text{desired}| = |\text{total}| - |\text{undesired}|$$

Use this when the undesired outcomes have a simpler structure.

Worked examples

Example 5.1 (License plates)**.** A license plate consists of 3 letters followed by 3 digits. How many are possible?

Solution. Using the multiplication rule:

- 3 letters: $26 \times 26 \times 26 = 26^3$ choices (with replacement)
- 3 digits: $10 \times 10 \times 10 = 10^3$ choices

Total: $26^3 \times 10^3 = 17,576,000$ plates.

Example 5.2 (No repeated letters)**.** How many 3-letter strings over $\{A, B, C, D\}$ have no repeated letters?

Solution. This is without replacement:

- First letter: 4 choices
- Second letter: 3 choices (can't repeat first)
- Third letter: 2 choices

Total: $P(4, 3) = 4 \times 3 \times 2 = 24$.

Example 5.3 (Sum of dice)**.** A fair die is rolled twice. What is the probability the sum is 7?

Solution.

- Sample space: All pairs (a, b) with $a, b \in \{1, 2, 3, 4, 5, 6\}$. Size: 36.
- Event: pairs summing to 7. These are $(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)$. Size: 6.

$P(\text{sum} = 7) = 6/36 = 1/6$.

Example 5.4 (Complement counting)**.** How many 5-bit binary strings contain at least one 1?

Solution. Direct counting is messy (exactly one 1, or two 1s, or...). Use the complement.

- Total 5-bit strings: $2^5 = 32$
- Strings with no 1s: just 00000, so 1

Strings with at least one 1: $32 - 1 = 31$.

Example 5.5 (Inclusion-exclusion)**.** How many integers in $\{1, 2, \ldots, 100\}$ are divisible by 2 or 3?

Solution. Let $A = \{n : 2 \mid n\}$ and $B = \{n : 3 \mid n\}$.

- $|A| = \lfloor 100/2 \rfloor = 50$
- $|B| = \lfloor 100/3 \rfloor = 33$
- $|A \cap B| = |\{n : 6 \mid n\}| = \lfloor 100/6 \rfloor = 16$

$|A \cup B| = 50 + 33 - 16 = 67$.

Example 5.6 (Pigeonhole: birth months)**.** Prove: Among any 13 people, at least two share a birth month.

Proof. There are 12 months (boxes) and 13 people (objects). By the pigeonhole principle, at least one month contains at least 2 people. □

Example 5.7 (Pigeonhole: remainders)**.** Prove: In any set of 6 integers, two have the same remainder when divided by 5.

Proof. Remainders mod 5 are in $\{0, 1, 2, 3, 4\}$ (5 boxes). With 6 integers (objects), at least two land in the same box. □

Example 5.8 (Adjacent seating)**.** How many ways can 8 people sit in a row if Alice and Bob must sit together?

Solution. Treat Alice-Bob as a single "super-person." Then:

- Arrange 7 objects: 7! ways
- Alice and Bob can swap within their block: 2 ways

Total: $7! \times 2 = 5040 \times 2 = 10080$.

Example 5.9 (Non-adjacent seating)**.** How many ways can 8 people sit in a row if Alice and Bob must NOT sit together?

Solution. Use complement counting.

- Total arrangements: $8! = 40320$
- Arrangements where they sit together: 10080 (from previous)

Answer: $40320 - 10080 = 30240$.

Example 5.10 (Pigeonhole: geometry)**.** Prove: Among any 5 points in a unit square, at least two are within distance $\sqrt{2}/2$ of each other.

Proof. Divide the unit square into 4 smaller squares of side 1/2. By pigeonhole, at least two of the 5 points lie in the same small square. The maximum distance between two points in a square of side 1/2 is the diagonal: $\sqrt{2}/2$. □

Example 5.11 (Conditional probability)**.** Two cards are drawn without replacement from a standard deck. Given that the first is an ace, what's the probability the second is also an ace?

Solution. After drawing an ace, the deck has 51 cards remaining, 3 of which are aces.

$$P(\text{2nd ace} \mid \text{1st ace}) = \frac{3}{51} = \frac{1}{17}$$

Example 5.12 (Bayes' rule: factory machines)**.** A factory has two machines. M_1 produces 70% of items with 2% defect rate; M_2 produces 30% with 5% defect rate. If an item is defective, what's the probability it came from M_2?

Solution. First, find $P(\text{defective})$:

$$P(D) = P(D \mid M_1)P(M_1) + P(D \mid M_2)P(M_2) = 0.02 \cdot 0.7 + 0.05 \cdot 0.3 = 0.029$$

Then apply Bayes:

$$P(M_2 \mid D) = \frac{P(D \mid M_2)P(M_2)}{P(D)} = \frac{0.05 \cdot 0.3}{0.029} = \frac{0.015}{0.029} \approx 0.517$$

A defective item is slightly more likely to come from M_2, even though M_2 produces fewer items overall. The higher defect rate more than compensates.

Example 5.13 (Computing derangements)**.** Compute D_4.

Solution. Using the formula:

$$D_4 = 4!\left(1 - 1 + \frac{1}{2} - \frac{1}{6} + \frac{1}{24}\right) = 24 \cdot \frac{9}{24} = 9$$

We can verify: the derangements of $\{1, 2, 3, 4\}$ are permutations where no element is in its original position. They are: 2143, 2341, 2413, 3142, 3412, 3421, 4123, 4312, 4321. Indeed, 9.

Common Mistake

Overcounting. Make sure you're not counting the same configuration multiple times. Ask:

- Does order matter? (permutation vs. combination)
- Are objects distinguishable?
- Are positions/boxes distinguishable?

Common Mistake

Misapplying the multiplication rule. The rule requires that choices at each step are independent of previous choices. If earlier choices constrain later options, you must account for this carefully.

Going Deeper: The Algebra of Types

The counting rules—multiplication and addition—have a surprising connection to types in programming. This reveals why these rules are fundamental.
For more detail, see the Category Theory Companion, Weeks 4–5.

Types Have Sizes

In programming, a type is a set of values:

Type	Description	Size
`Void`	empty type	0
`Unit` or `()`	single value	1
`Bool`	`True` or `False`	2

Products and Sums

When we combine types:

- **Product type** (A, B): $|A \times B| = |A| \times |B|$
- **Sum type** `Either A B`: $|A + B| = |A| + |B|$

This is exactly the multiplication and addition rules! The type `(Bool, Bool)` has $2 \times 2 = 4$ values. The type `Either Bool ()` has $2 + 1 = 3$ values.

Function Types as Exponentials

Function types $A \to B$ have $|B|^{|A|}$ values. Why? For each input in A, you choose one output in B. That's $|B|$ choices for each of $|A|$ inputs, so $|B|^{|A|}$ total functions.

Exercises

1. How many values does `(Bool, Bool, Bool)` have?
2. How many values does `Either Bool Bool` have? Different from `(Bool, Bool)`?
3. If $|A| = 3$ and $|B| = 4$, what are $|A \times B|$ and $|A + B|$?
4. Verify: there are $3^2 = 9$ functions from `Bool` to $\{1, 2, 3\}$.

Practice

1. A fair die is rolled twice. What is the probability the sum is 7?
2. How many 5-bit binary strings contain at least one 1?
3. Use inclusion–exclusion to count integers in $\{1, \ldots, 100\}$ divisible by 2 or 3.
4. Use the pigeonhole principle to show that among 13 people, two share a birth month.
5. How many 4-digit PINs (digits 0–9) have no repeated digits?
6. How many ways can 6 books be arranged on a shelf if two specific books must be at the ends?
7. How many bit strings of length 8 start with 1 or end with 00?
8. Prove: Among any 10 integers, two have a difference divisible by 9.
9. A restaurant offers 3 appetizers, 5 mains, and 2 desserts. How many 3-course meals are possible?
10. How many permutations of ABCDEF contain ABC as a consecutive substring?
11. Use inclusion-exclusion to count integers in $\{1, \ldots, 1000\}$ divisible by 2, 3, or 5.
12. Compute D_5 (derangements of 5 elements).
13. Five friends exchange gifts so no one receives their own. How many ways?
14. Two dice are rolled. Let $A =$ "sum is even" and $B =$ "sum ≥ 10". Compute $P(A \mid B)$ and determine if A, B are independent.
15. A box has 3 fair coins and 1 double-headed coin. A coin is chosen randomly and flipped. If heads, what's the probability it was double-headed?
16. A disease test has sensitivity 0.95 and false positive rate 0.02. If 1% of the population has the disease, what's the probability someone who tests positive actually has it?

6 Week 5: Counting and Probability II

Or: "The art of choosing without caring about order"

Reading

Epp §9.5–9.7; 5.6–5.7. Supplemental: generating functions.
Category theory companion: Weeks 4–5 (`category_theory_companion.pdf`).

Why Combinations?

Last week we counted permutations—arrangements where order matters. But often we don't care about order: a committee is just a set of people, not a sequence. A poker hand is the same hand regardless of which card you drew first. A subset is a subset.

This leads to binomial coefficients, one of the most important tools in discrete mathematics. They show up everywhere: in probability, in algebra (the binomial theorem), in Pascal's triangle, in combinatorial proofs. Master these and you've mastered a significant chunk of counting.

Learning objectives

- Compute combinations and binomial coefficients.
- Count with repetition using stars and bars.
- Use Pascal's identity and the binomial theorem.
- Set up and solve basic recurrence relations.

Key definitions and facts

Binomial Coefficients

Definition 6.1 (Binomial coefficient)**.** The **binomial coefficient** $\binom{n}{k}$ (read "n choose k") is:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

It counts the number of ways to choose k items from n items without regard to order. Equivalently, it counts k-element subsets of an n-element set.

Why the formula? Start with the k-permutations: $P(n,k) = \frac{n!}{(n-k)!}$ ways to choose k items in order. But each subset gets counted $k!$ times (once for each ordering of its elements). Divide out: $\binom{n}{k} = \frac{P(n,k)}{k!} = \frac{n!}{k!(n-k)!}$.

Theorem 6.1 (Pascal's identity)**.** *For $1 \leq k \leq n$:*

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

Combinatorial proof: Focus on element n. Every k-subset either contains n or doesn't.

- Subsets containing n: choose $k-1$ more from the other $n-1$ elements $\Rightarrow \binom{n-1}{k-1}$

- Subsets not containing n: choose all k from the other $n-1$ elements $\Rightarrow \binom{n-1}{k}$

Add them up. □

This identity gives Pascal's triangle, where each entry is the sum of the two above it.

Theorem 6.2 (Binomial theorem)**.** *For any x, y and non-negative integer n:*

$$(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

Why? When you expand $(x+y)^n = (x+y)(x+y)\cdots(x+y)$, each term in the expansion comes from choosing x or y from each factor. The coefficient of $x^{n-k}y^k$ counts how many ways to choose y from exactly k of the n factors, which is $\binom{n}{k}$.

Theorem 6.3 (Useful identities)**.**

- $\binom{n}{k} = \binom{n}{n-k}$ *(symmetry: choosing what to include = choosing what to exclude)*
- $\sum_{k=0}^{n} \binom{n}{k} = 2^n$ *(total number of subsets)*
- $\sum_{k=0}^{n} (-1)^k \binom{n}{k} = 0$ *(alternating sum; set $x = 1, y = -1$ in binomial theorem)*
- $\binom{n}{0} + \binom{n}{2} + \cdots = \binom{n}{1} + \binom{n}{3} + \cdots = 2^{n-1}$ *(even and odd sized subsets)*

Stars and Bars

Theorem 6.4 (Stars and bars)**.** *The number of ways to place n identical objects into k distinct bins is:*

$$\binom{n+k-1}{k-1}$$

Equivalently, this counts non-negative integer solutions to $x_1 + x_2 + \cdots + x_k = n$.

Why it works: Represent the objects as stars (*) and use bars (—) to separate bins. Example with $n = 5$ objects in $k = 3$ bins: $**|*|**$ represents $(2, 1, 2)$.

Any arrangement is a string of n stars and $k-1$ bars, totaling $n+k-1$ symbols. We choose which $k-1$ positions are bars: $\binom{n+k-1}{k-1}$.

Definition 6.2 (Multinomial coefficient)**.** The number of ways to partition n objects into groups of sizes $k_1, k_2, \ldots, k_r$ (where $k_1 + \cdots + k_r = n$) is:

$$\binom{n}{k_1, k_2, \ldots, k_r} = \frac{n!}{k_1! k_2! \cdots k_r!}$$

This generalizes the binomial coefficient: $\binom{n}{k} = \binom{n}{k, n-k}$.

Recurrence relations

Definition 6.3 (Recurrence relation)**.** A **recurrence relation** defines a sequence $\{a_n\}$ by expressing a_n in terms of earlier terms, together with initial conditions.

For example, the Fibonacci sequence: $F_0 = 0, F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$.

Theorem 6.5 (Second-order linear homogeneous recurrences)**.** *If $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ for $n \geq 2$, then the* ***characteristic equation*** *is:*

$$r^2 = c_1 r + c_2 \quad \textit{or equivalently} \quad r^2 - c_1 r - c_2 = 0$$

- *If roots are distinct ($r_1 \neq r_2$): $a_n = \alpha r_1^n + \beta r_2^n$*
- *If roots are repeated ($r_1 = r_2 = r$): $a_n = (\alpha + \beta n) r^n$*

Constants α, β are determined from initial conditions.

Example 6.1. Solve $a_n = 2a_{n-1} + 1$ with $a_0 = 0$.

Solution. This is inhomogeneous (the $+1$ term). One approach: unroll.

$$\begin{aligned} a_n &= 2a_{n-1} + 1 = 2(2a_{n-2} + 1) + 1 = 4a_{n-2} + 2 + 1 \\ &= 8a_{n-3} + 4 + 2 + 1 = \cdots = 2^n a_0 + (2^{n-1} + \cdots + 2 + 1) \\ &= 0 + (2^n - 1) = 2^n - 1 \end{aligned}$$

Check: $a_0 = 2^0 - 1 = 0$ ✓, $a_1 = 2 \cdot 0 + 1 = 1 = 2^1 - 1$ ✓.

Going Deeper: Generating Functions

The **ordinary generating function** (OGF) of a sequence $\{a_n\}$ is the formal power series:

$$A(x) = \sum_{n=0}^{\infty} a_n x^n$$

Example 1 (constant sequence). If $a_n = 1$ for all n:

$$A(x) = 1 + x + x^2 + \cdots = \frac{1}{1-x}$$

Example 2 (Fibonacci). Let $F(x) = \sum_{n \geq 0} F_n x^n$. The recurrence $F_n = F_{n-1} + F_{n-2}$ translates to:

$$F(x) - x = xF(x) + x^2 F(x) \quad \Rightarrow \quad F(x) = \frac{x}{1 - x - x^2}$$

Partial fractions give the closed form for F_n.

Generating functions are a powerful technique for solving recurrences and counting. The algebraic manipulation of series encodes combinatorial reasoning.

Category Lens: Subsets as Maps to 2

In **Set**, every subset $S \subseteq A$ corresponds to a **characteristic function**:

$$\chi_S : A \to 2, \quad \chi_S(a) = \begin{cases} 1 & \text{if } a \in S \\ 0 & \text{otherwise} \end{cases}$$

where $2 = \{0, 1\}$.

So the set of all subsets of A is the exponential object 2^A (the set of all functions from A to 2).

- $|2^A| = 2^{|A|}$ explains why there are 2^n subsets of an n-element set.

- $\binom{n}{k}$ counts functions $A \to 2$ where exactly k elements map to 1.

This reframes binomial coefficients as counting *maps* rather than subsets—a perspective that generalizes to other categories.

Worked examples

Example 6.2 (Stars and bars)**.** How many non-negative integer solutions are there to $x_1+x_2+x_3 = 7$?

Solution. We have 7 identical "units" to distribute among 3 distinct variables. By stars and bars:

$$\binom{7+3-1}{3-1} = \binom{9}{2} = \frac{9 \cdot 8}{2} = 36$$

Example 6.3 (Proving a sum identity)**.** Prove $\sum_{k=0}^{n} \binom{n}{k} = 2^n$.

Proof 1 (Binomial theorem): Set $x = y = 1$ in $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$. Get $2^n = \sum_{k=0}^{n} \binom{n}{k}$.

Proof 2 (Combinatorial): The LHS counts subsets by size (0-subsets + 1-subsets + ... + n-subsets). The RHS counts subsets directly: each element is in or out, 2^n choices. Both count all subsets. □

Example 6.4 (Arrangements with repetition)**.** How many ways can the letters of MISSISSIPPI be arranged?

Solution. Total 11 letters with repetitions: M(1), I(4), S(4), P(2).

Using the multinomial coefficient:

$$\binom{11}{1,4,4,2} = \frac{11!}{1! \cdot 4! \cdot 4! \cdot 2!} = \frac{39916800}{1 \cdot 24 \cdot 24 \cdot 2} = 34650$$

Example 6.5 (Pascal's identity algebraically)**.** Prove $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ algebraically.

Proof.

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

Common denominator is $k!(n-k)!$:

$$= \frac{(n-1)! \cdot k}{k!(n-k)!} + \frac{(n-1)! \cdot (n-k)}{k!(n-k)!}$$
$$= \frac{(n-1)![k+(n-k)]}{k!(n-k)!} = \frac{(n-1)! \cdot n}{k!(n-k)!} = \frac{n!}{k!(n-k)!} = \binom{n}{k}$$

Example 6.6 (Binomial expansion)**.** Expand $(2x-1)^4$.

Solution. By the binomial theorem with $a = 2x$ and $b = -1$:

$$\begin{aligned}(2x-1)^4 &= \sum_{k=0}^{4} \binom{4}{k} (2x)^{4-k} (-1)^k \\ &= (2x)^4 - 4(2x)^3 + 6(2x)^2 - 4(2x) + 1 \\ &= 16x^4 - 32x^3 + 24x^2 - 8x + 1\end{aligned}$$

Example 6.7 (Positive integer solutions)**.** How many positive integer solutions are there to $x_1 + x_2 + x_3 = 10$?

Solution. For positive integers ($x_i \geq 1$), substitute $y_i = x_i - 1$ so $y_i \geq 0$:

$$(y_1 + 1) + (y_2 + 1) + (y_3 + 1) = 10 \implies y_1 + y_2 + y_3 = 7$$

Now count non-negative solutions by stars and bars:

$$\binom{7+3-1}{3-1} = \binom{9}{2} = 36$$

Example 6.8 (Committee with constraints)**.** A committee of 5 is chosen from 6 men and 4 women. How many committees have at least 2 women?

Solution. "At least 2 women" means 2, 3, or 4 women:

- 2 women, 3 men: $\binom{4}{2}\binom{6}{3} = 6 \cdot 20 = 120$
- 3 women, 2 men: $\binom{4}{3}\binom{6}{2} = 4 \cdot 15 = 60$
- 4 women, 1 man: $\binom{4}{4}\binom{6}{1} = 1 \cdot 6 = 6$

Total: $120 + 60 + 6 = 186$.

Alternative (complement): Total committees: $\binom{10}{5} = 252$. Fewer than 2 women: $\binom{4}{0}\binom{6}{5} + \binom{4}{1}\binom{6}{4} = 6 + 60 = 66$. Answer: $252 - 66 = 186$. ✓

Example 6.9 (Distributing identical objects)**.** In how many ways can 10 identical apples be distributed among 4 children?

Solution. This is stars and bars with $n = 10$ objects, $k = 4$ bins:

$$\binom{10+4-1}{4-1} = \binom{13}{3} = \frac{13 \cdot 12 \cdot 11}{6} = 286$$

Example 6.10 (Vandermonde's identity)**.** Prove $\binom{m+n}{r} = \sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k}$.

Combinatorial proof: The LHS counts r-subsets of a set with m red and n blue elements. The RHS counts the same by cases: choose k red elements and $r - k$ blue elements, for each possible k. □

Practice

1. Compute $\binom{12}{5}$.
2. How many 8-card poker hands contain exactly 3 hearts?
3. Prove Pascal's identity combinatorially.
4. Use the binomial theorem to expand $(2x - 1)^5$.
5. How many positive integer solutions are there to $x_1 + x_2 + x_3 + x_4 = 15$?
6. Prove: $\binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2 = \binom{2n}{n}$. (Hint: think about choosing n items from $2n$.)
7. A committee of 5 is to be chosen from 6 men and 4 women. How many committees have at least 2 women?

8. Solve the recurrence $a_n = 3a_{n-1} - 2$ with $a_0 = 1$.

9. Solve the recurrence $a_n = 4a_{n-1} - 4a_{n-2}$ with $a_0 = 1$, $a_1 = 2$.

10. Let t_n be the number of ways to tile a $2 \times n$ board with 1×2 dominoes. Find a recurrence for t_n with initial conditions, and compute t_5.

11. Find the ordinary generating function for the sequence $a_n = n$ for $n \geq 0$.

7 Week 6: Expected Value and Introduction to Graphs

Or: "What happens on average, and how things connect"

Reading

Epp §9.8; 10.1; 10.2.
Category theory companion: Weeks 6–7 (`category_theory_companion.pdf`).

Why Expected Value?

When outcomes are random, you can't predict what will happen, but you can often say something useful about what happens *on average*. Expected value captures this: it's the long-run average you'd see if you repeated an experiment many times.

Expected value is central to decision-making under uncertainty, algorithm analysis (average-case complexity), and basically anywhere probability meets the real world. The key insight this week is *linearity of expectation*—a shockingly powerful tool that makes many calculations trivial.

Why Graphs?

Graphs are everywhere: social networks, road maps, the internet, molecular structures, dependency relationships in software. Anything with "things" and "connections between things" is a graph.

This week introduces the basic vocabulary: vertices, edges, degrees, adjacency. The handshake theorem—the sum of degrees equals twice the number of edges—is our first fundamental result about graph structure.

Learning objectives

- Define and compute expected value for discrete random variables.
- Apply linearity of expectation to simplify calculations.
- Use indicator random variables for counting.
- Define graphs, vertices, edges, and basic terminology.
- Apply the handshake theorem to relate degrees and edges.
- Distinguish simple graphs, multigraphs, and digraphs.

Part I: Expected Value

Probability Foundations

Definition 7.1 (Probability axioms)**.** A **probability measure** on a sample space S assigns to each event $A \subseteq S$ a number $P(A)$ satisfying:

1. $P(A) \geq 0$ for all events A (non-negativity)
2. $P(S) = 1$ (something happens)
3. If $A_1, A_2, \ldots$ are pairwise disjoint, then $P(\bigcup_i A_i) = \sum_i P(A_i)$ (additivity)

From these three axioms, everything else follows: $P(\emptyset) = 0$, $P(A^c) = 1 - P(A)$, $P(A \cup B) = P(A) + P(B) - P(A \cap B)$, etc.

Definition 7.2 (Random variable)**.** A **random variable** X on sample space S is a function $X : S \to \mathbb{R}$ assigning a real number to each outcome. For discrete random variables, the possible values form a finite or countable set.

Definition 7.3 (Expected value)**.** The **expected value** (or **expectation** or **mean**) of a discrete random variable X is:

$$E[X] = \sum_x x \cdot P(X = x)$$

where the sum is over all possible values x of X.

Think of expected value as a weighted average: each value is weighted by its probability.

Theorem 7.1 (Linearity of expectation)**.** *For any random variables X and Y and constants a, b:*

$$E[aX + bY] = aE[X] + bE[Y]$$

More generally:

$$E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i]$$

Key Result

Linearity of expectation works *regardless of whether the random variables are independent.* This is its superpower. You can break a complicated random variable into simple pieces, compute expectations of the pieces separately, and add them up—even when the pieces are tangled together.

Definition 7.4 (Indicator random variable)**.** The **indicator** I_A for event A is:

$$I_A = \begin{cases} 1 & \text{if } A \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

Key fact: $E[I_A] = P(A)$.

Indicators are the workhorse of expected value calculations. If you're counting "how many of events $A_1, \ldots, A_n$ occur," write $X = I_{A_1} + \cdots + I_{A_n}$ and apply linearity:

$$E[X] = P(A_1) + \cdots + P(A_n)$$

Definition 7.5 (Common distributions)**.**

- **Bernoulli(p):** $X = 1$ with probability p, $X = 0$ with probability $1 - p$. $E[X] = p$.
- **Binomial(n, p):** Number of successes in n independent trials. $E[X] = np$.
- **Geometric(p):** Number of trials until first success. $E[X] = 1/p$.
- **Uniform on $\{1, \ldots, n\}$:** Each value equally likely. $E[X] = (n + 1)/2$.

Definition 7.6 (Variance)**.** The **variance** of X measures spread around the mean:

$$\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2$$

Standard deviation is $\sigma = \sqrt{\text{Var}(X)}$.

Part II: Introduction to Graphs

Definition 7.7 (Graph)**.** A **graph** $G = (V, E)$ consists of:

- V: a finite nonempty set of **vertices** (or nodes)
- E: a set of **edges**, each connecting two vertices

Definition 7.8 (Types of graphs)**.**

- **Simple graph:** No loops (edges from a vertex to itself) and no multiple edges.
- **Multigraph:** Allows multiple edges between the same pair.
- **Pseudograph:** Allows loops and multiple edges.
- **Directed graph (digraph):** Edges have direction.

Unless otherwise stated, "graph" means simple graph.

Definition 7.9 (Basic terminology)**.**

- Two vertices are **adjacent** if an edge connects them.
- An edge is **incident** to its endpoints.
- The **degree** $\deg(v)$ is the number of edges incident to v (loops count twice).
- Degree 0 = **isolated**; degree 1 = **leaf** (or pendant).
- The **neighborhood** $N(v)$ is the set of vertices adjacent to v.

Theorem 7.2 (Handshake theorem)**.** *In any graph $G = (V, E)$:*

$$\sum_{v \in V} \deg(v) = 2|E|$$

Why it works: Each edge has two endpoints. When we sum degrees, each edge gets counted exactly twice (once at each end). □

Corollary 7.1. *Every graph has an even number of vertices with odd degree.*

Proof: The sum of degrees is even (it's $2|E|$). If you add up numbers and get an even total, you must have an even number of odd summands. □

Definition 7.10 (Special graphs)**.**

- **Complete graph K_n:** All $\binom{n}{2}$ possible edges present.
- **Cycle C_n:** Vertices $v_1, \ldots, v_n$ with edges $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1$.
- **Path P_n:** Like a cycle but no edge from v_n to v_1; has $n - 1$ edges.
- **Complete bipartite $K_{m,n}$:** Two parts of sizes m and n; every cross-pair connected; mn edges.
- **n-cube Q_n:** Vertices are n-bit strings; edges connect strings differing in one bit.

Definition 7.11 (Degree sequence)**.** The **degree sequence** is the list of degrees in non-increasing order. Example: K_4 has degree sequence $(3, 3, 3, 3)$.

A natural question: given a sequence of numbers, is it the degree sequence of some graph? The handshake theorem gives a necessary condition (sum must be even), but that's not sufficient. The Erdős–Gallai conditions or Havel–Hakimi algorithm give complete answers.

Definition 7.12 (Subgraph and complement)**.** H is a **subgraph** of G if H's vertices and edges are subsets of G's.

The **complement** $\overline{G}$ has the same vertices as G; two vertices are adjacent in $\overline{G}$ iff they're not adjacent in G.

Category Lens: Graphs as a Category

Graphs form a category **Graph** where morphisms are **graph homomorphisms**—vertex maps that preserve adjacency. If $\phi : G \to H$ is a homomorphism and uv is an edge in G, then $\phi(u)\phi(v)$ is an edge in H.

There's a forgetful functor $U : \mathbf{Graph} \to \mathbf{Set}$ sending a graph to its vertex set. This lets us use set-theoretic reasoning while respecting graph structure.

Events (subsets of a sample space) correspond to characteristic functions $A \to 2$. If $f : A \to B$, the preimage map $f^{-1} : \mathcal{P}(B) \to \mathcal{P}(A)$ is contravariant and preserves unions/intersections—explaining why probability behaves nicely under functions.

Worked examples

Example 7.1 (Die roll)**.** A fair die is rolled. Let X be the outcome. Compute $E[X]$.

Solution. Each outcome 1–6 has probability 1/6:

$$E[X] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + \cdots + 6 \cdot \frac{1}{6} = \frac{1+2+3+4+5+6}{6} = \frac{21}{6} = 3.5$$

Example 7.2 (Coin flips with linearity)**.** A fair coin is flipped 10 times. What is the expected number of heads?

Solution. Let $X_i = 1$ if flip i is heads, 0 otherwise. Then $X = X_1 + \cdots + X_{10}$ counts heads.

By linearity: $E[X] = E[X_1] + \cdots + E[X_{10}] = 10 \cdot \frac{1}{2} = 5$.

Note: we didn't need to know the distribution of X explicitly!

Example 7.3 (Fixed points in permutations)**.** In a random permutation of $\{1, \ldots, n\}$, what is the expected number of fixed points?

Solution. Let $X_i = 1$ if element i is in position i. Then $X = \sum_{i=1}^{n} X_i$.

$P(\text{element } i \text{ fixed}) = 1/n$ (out of $n!$ permutations, $(n-1)!$ fix position i).

By linearity: $E[X] = n \cdot \frac{1}{n} = 1$.

The expected number of fixed points is exactly 1, regardless of n. This is a beautiful result!

Example 7.4 (Geometric distribution)**.** What is the expected number of die rolls to get a 6?

Solution. This is geometric with success probability $p = 1/6$.

$$E[X] = \frac{1}{p} = 6$$

Example 7.5 (Handshake theorem for K_4)**.** Verify the handshake theorem for the complete graph K_4.

Solution. K_4 has 4 vertices, each with degree 3.

- Sum of degrees: $3 + 3 + 3 + 3 = 12$

- Number of edges: $\binom{4}{2} = 6$
- Check: $2 \times 6 = 12$ ✓

Example 7.6 (Degree sequence realizability)**.** Is there a simple graph with degree sequence $(3, 3, 2, 2, 2)$?

Solution. Sum $= 12$, which is even. ✓

Use Havel–Hakimi: take the largest degree (3), remove it, subtract 1 from the next 3 degrees:

$$(3, 3, 2, 2, 2) \to (2, 1, 1, 2) \to (2, 2, 1, 1)$$

Repeat: $(2, 2, 1, 1) \to (1, 0, 1) \to (1, 1, 0) \to (0, 0)$ ✓

Yes, such a graph exists.

Example 7.7 (Edges in Q_n)**.** How many edges does the n-cube Q_n have?

Solution. Q_n has 2^n vertices (all n-bit strings). Each vertex has degree n (flip any of n bits).

Sum of degrees: $n \cdot 2^n$.

By handshake: $|E| = \frac{n \cdot 2^n}{2} = n \cdot 2^{n-1}$.

Example 7.8 (Pigeonhole for degrees)**.** Prove: Every simple graph on $n \geq 2$ vertices has at least two vertices of the same degree.

Solution. In a simple graph, degrees range from 0 to $n - 1$ (can't have degree n—no loops). That's n possible values.

But wait: if some vertex has degree 0 (isolated), no vertex can have degree $n - 1$ (connected to everyone). So at most $n - 1$ distinct degrees are achievable.

By pigeonhole: n vertices, at most $n - 1$ distinct degrees $\Rightarrow$ two vertices share a degree.

Common Mistake

Assuming independence for linearity. The formula $E[X + Y] = E[X] + E[Y]$ works even when X and Y are dependent! Many students add "assuming independence" when it's not needed.

Common Mistake

Confusing $E[XY]$ with $E[X] \cdot E[Y]$. These are equal only when X and Y are independent. In general, $E[XY] \neq E[X]E[Y]$.

Practice

1. A coin is flipped 10 times. What is the expected number of heads?
2. Show that the sum of degrees in a tree on n vertices is $2(n - 1)$.
3. Find $E[X]$ for a geometric random variable with success probability p.
4. Decide whether a graph with degree sequence $(3, 3, 2, 2, 2)$ is possible.
5. In a random permutation of $\{1, \ldots, n\}$, what is the expected number of elements greater than all previous elements?
6. How many edges does $K_{4,5}$ have? What are the vertex degrees?
7. Prove that the complement of K_n is an empty graph.

8. A bag has 5 red and 3 blue marbles. Two are drawn without replacement. What is the expected number of red marbles drawn?
9. Show that a simple graph on n vertices has at most $\binom{n}{2}$ edges.
10. Using handshake: If every vertex has degree $\geq k$, show $|E| \geq k|V|/2$.
11. Prove every graph has an even number of odd-degree vertices.
12. In a room of 100 people, everyone shakes hands with exactly 3 others. Is this possible?

8 Week 7: Graph Theory I — Paths and Connectivity

Or: "How do you get from here to there, and is there more than one way?"

Reading

Epp §10.1–10.3.
Category theory companion: Weeks 6–7 (`category_theory_companion.pdf`).

Why Paths and Connectivity?

Last week we introduced graphs as a way to model "things with connections." But having a graph is just the beginning. The interesting questions are about *navigation*: Can you get from vertex A to vertex B? How many ways? What's the shortest route?

These questions show up everywhere. Can a message travel through a network from sender to receiver? Can you drive from Portland to Boston? Can a chess knight visit every square? Can you trace a figure without lifting your pen?

This week introduces the vocabulary for talking about movement through graphs: walks, paths, circuits, and the beautiful theorem of Euler that tells us exactly when we can traverse every edge exactly once.

Learning objectives

- Distinguish walks, trails, paths, and circuits.
- Apply Euler's criteria for trails and circuits.
- Determine graph connectivity and connected components.
- Represent graphs with adjacency matrices and adjacency lists.
- Determine whether two graphs are isomorphic.

Key definitions and facts

Ways of Walking Through a Graph

Definition 8.1 (Walk)**.** A **walk** from vertex v_0 to vertex v_n is a sequence:

$$v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n$$

where each e_i connects v_{i-1} to v_i. The **length** is n (the number of edges traversed).

A walk is the most general notion: you're just wandering through the graph, following edges. You can revisit vertices, cross the same edge multiple times, go in circles—anything goes as long as each step follows an actual edge.

But often we want more structure:

Definition 8.2 (Types of walks)**.**

- A **trail** is a walk with no repeated edges. You can revisit vertices, but you can't cross the same bridge twice.
- A **path** is a walk with no repeated vertices. This is the "clean" version—no backtracking, no loops, just a direct route.

- A **closed walk** returns to its starting point ($v_0 = v_n$).
- A **circuit** is a closed trail—returns to start without repeating any edge.
- A **cycle** is a closed path—returns to start without repeating any vertex (except the start/end).

The hierarchy goes: walk $\supseteq$ trail $\supseteq$ path, and closed walk $\supseteq$ circuit $\supseteq$ cycle. Each additional restriction makes the object "nicer" but harder to find.

Proposition 8.1 (Path existence)**.** *If there's a walk from u to v, there's a path from u to v.*

Why? If your walk repeats a vertex, you went in a circle—just cut out the circle and you have a shorter walk. Keep cutting until no repeats remain. This "shortcutting" argument shows that walks and paths detect the same reachability information.

Connectivity

Definition 8.3 (Connectivity)**.**

- A graph is **connected** if there's a path between every pair of vertices. You can get anywhere from anywhere.
- A **connected component** is a maximal connected piece—add any more vertices and you'd break connectivity.
- A **cut vertex** (or articulation point) is a vertex whose removal disconnects the graph. It's a bottleneck.
- A **bridge** is an edge whose removal disconnects the graph.

Every graph is a disjoint union of its connected components. A connected graph has exactly one component. The extreme case: a graph with no edges has n components (each vertex is its own island).

Euler Trails and Circuits

Here's a famous problem from 1736: the city of Königsberg had seven bridges connecting four landmasses. Can you walk through the city crossing each bridge exactly once?

Euler showed the answer is no, and in doing so invented graph theory.

Definition 8.4 (Euler trail and circuit)**.** An **Euler trail** is a trail that uses every edge exactly once. An **Euler circuit** is an Euler trail that starts and ends at the same vertex.

The question is: when do these exist? The answer is surprisingly clean.

Theorem 8.1 (Euler's theorem)**.** *Let G be a connected graph.*

1. *G has an Euler circuit if and only if every vertex has even degree.*
2. *G has an Euler trail (but no circuit) if and only if exactly two vertices have odd degree. The trail must start at one odd-degree vertex and end at the other.*

Why does this work? Think about what happens when you walk through a vertex: you enter on one edge and leave on another. That uses up edges in pairs. If a vertex has odd degree, you can't pair up all its edges—one will be left over. That means an odd-degree vertex must be where you start or end your journey (using that unpaired edge as the first or last step).

An Euler circuit visits every edge and returns home, so every vertex gets entered and exited the same number of times—all degrees must be even. An Euler trail can have exactly two odd-degree vertices (start and end).

Key Result

The Euler circuit/trail test:

1. Count vertices of odd degree.
2. 0 odd-degree vertices $\Rightarrow$ Euler circuit exists.
3. 2 odd-degree vertices $\Rightarrow$ Euler trail exists (no circuit).
4. > 2 odd-degree vertices $\Rightarrow$ no Euler trail at all.

For Königsberg: each landmass had odd degree (3, 3, 3, 5). Four odd-degree vertices, so no Euler trail. The citizens couldn't do it.

Definition 8.5 (Hamiltonian path and cycle)**.** A **Hamiltonian path** visits every *vertex* exactly once. A **Hamiltonian cycle** visits every vertex exactly once, returning to start.

Here's an interesting contrast: Euler is about edges, Hamilton is about vertices. You might think they'd have similar characterizations, but they don't.

Remark. Unlike Euler paths/circuits, there's no simple test for Hamiltonian paths/cycles. Determining whether one exists is NP-complete—one of the classic hard problems in computer science.

Graph representations

How do we actually store a graph in a computer?

Definition 8.6 (Adjacency matrix)**.** The **adjacency matrix** A of a graph with n vertices is an $n \times n$ matrix where:

$$A_{ij} = \text{number of edges between vertex } i \text{ and vertex } j$$

For simple graphs, $A_{ij} \in \{0, 1\}$. The matrix is symmetric for undirected graphs.

Proposition 8.2 (Properties of adjacency matrices)**.**

- *The sum of row i equals* $\deg(v_i)$.
- *The sum of all entries equals* $2|E|$ *(handshake theorem in matrix form).*
- *The diagonal is all zeros for simple graphs (no loops).*
- $(A^k)_{ij}$ *counts the number of walks of length k from v_i to v_j.*

That last property is remarkable. Matrix multiplication corresponds to "extending walks by one step." The (i, j) entry of A^2 counts length-2 walks from i to j because it sums over all intermediate vertices m: how many ways to go $i \to m \to j$?

Definition 8.7 (Adjacency list)**.** An **adjacency list** stores, for each vertex, a list of its neighbors. More space-efficient for sparse graphs (when $|E| \ll |V|^2$).

Adjacency matrix: $O(|V|^2)$ space, $O(1)$ edge lookup. Adjacency list: $O(|V| + |E|)$ space, $O(\deg(v))$ edge lookup. Choose based on your graph's density.

Graph isomorphism

When are two graphs "the same"? They might have different vertex names, different drawings, but the same underlying structure.

Definition 8.8 (Graph isomorphism)**.** Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic**, written $G_1 \cong G_2$, if there exists a bijection $f : V_1 \to V_2$ such that:

$$\{u, v\} \in E_1 \iff \{f(u), f(v)\} \in E_2$$

The function f is an **isomorphism**.

An isomorphism is a relabeling of vertices that preserves adjacency. If two graphs are isomorphic, they have the same structure—they're the same graph wearing different name tags.

Key Result

[Categorical terminology] In the category **Graph**, graph isomorphisms are exactly the bijective graph homomorphisms whose inverse is also a homomorphism. This matches the general categorical definition: an isomorphism is a morphism with a two-sided inverse. Monomorphisms and epimorphisms depend on the chosen notion of graph morphism; we will not characterize them here.

Theorem 8.2 (Isomorphism invariants)**.** *If $G_1 \cong G_2$, then they share:*

1. *Vertex count: $|V_1| = |V_2|$*
2. *Edge count: $|E_1| = |E_2|$*
3. *Degree sequence (sorted list of degrees)*
4. *Number of cycles of each length*
5. *Number of connected components*
6. *Corresponding subgraph structure*

These are necessary *but not* sufficient*. Two graphs can match on all these counts and still not be isomorphic.*

Proof Strategy

To show two graphs are NOT isomorphic: find an invariant they differ on. To show they ARE isomorphic: construct an explicit bijection and verify edge preservation.

Definition 8.9 (Automorphism)**.** An **automorphism** is an isomorphism from a graph to itself—a "symmetry" of the graph. The set of automorphisms forms a group under composition.

A highly symmetric graph (like the complete graph or the hypercube) has many automorphisms. A "lopsided" graph with no symmetry has only the trivial automorphism (the identity).

Distance and diameter

Definition 8.10 (Distance)**.** The **distance** $d(u, v)$ between vertices u and v is the length of a shortest path between them. If no path exists, $d(u, v) = \infty$.

Distance is a metric: $d(u, u) = 0$, $d(u, v) = d(v, u)$, and $d(u, w) \leq d(u, v) + d(v, w)$.

Definition 8.11 (Eccentricity, radius, diameter)**.**

- The **eccentricity** of vertex v is the maximum distance from v to any other vertex: how far away is the farthest point?
- The **diameter** of a connected graph is the maximum eccentricity—the farthest two points can be.
- The **radius** is the minimum eccentricity—there's a "center" vertex that's as close to everything as possible.
- A **center** is a vertex with eccentricity equal to the radius.

Graph coloring

Here's a classic problem: given a map, color the countries so no two adjacent countries share a color. How many colors do you need?

Definition 8.12 (Vertex coloring)**.** A **(proper) vertex coloring** assigns colors to vertices such that no two adjacent vertices share a color. A **k-coloring** uses at most k colors.

Definition 8.13 (Chromatic number)**.** The **chromatic number** $\chi(G)$ is the minimum number of colors needed to properly color G.

Theorem 8.3 (Chromatic number bounds)**.**

1. *$\chi(G) \geq \omega(G)$, where $\omega(G)$ is the clique number (largest complete subgraph). If there's a K_5 hiding in your graph, you need at least 5 colors.*
2. *$\chi(G) \leq \Delta(G) + 1$, where $\Delta(G)$ is the maximum degree. Greedy coloring never needs more than this.*
3. ***Brooks' theorem:** If G is connected and isn't a complete graph or odd cycle, then $\chi(G) \leq \Delta(G)$.*

Theorem 8.4 (Chromatic numbers of special graphs)**.**

- $\chi(K_n) = n$ — *complete graph needs n colors (everyone's adjacent)*
- $\chi(C_n) = 2$ *if n is even,* $\chi(C_n) = 3$ *if n is odd*
- $\chi(K_{m,n}) = 2$ — *bipartite graphs are 2-colorable (color the parts!)*
- *Trees (with at least one edge) have* $\chi(T) = 2$

Definition 8.14 (Bipartite graph)**.** A graph is **bipartite** if its vertices can be split into two sets where every edge crosses between sets. Equivalently, $\chi(G) \leq 2$.

Theorem 8.5 (Bipartite characterization)**.** *A graph is bipartite if and only if it contains no odd-length cycle.*

This is a beautiful characterization: odd cycles are the only obstruction to bipartiteness.

Planar graphs

Definition 8.15 (Planar graph)**.** A graph is **planar** if it can be drawn in the plane with no edge crossings (except at vertices).

Definition 8.16 (Faces)**.** In a planar drawing, the plane is divided into **faces** (regions), including one unbounded outer face.

Theorem 8.6 (Euler's formula for planar graphs)**.** *For a connected planar graph with V vertices, E edges, and F faces:*

$$V - E + F = 2$$

This is a topological invariant—it doesn't depend on how you draw the graph, only on its structure.

Theorem 8.7 (Edge bound for planar graphs)**.** *For a connected planar graph with $V \geq 3$:*

$$E \leq 3V - 6$$

If triangle-free (no 3-cycles), then $E \leq 2V - 4$.

Corollary 8.1. *K_5 and $K_{3,3}$ are not planar.*

Proof. K_5: $V = 5$, $E = 10$. But $3V - 6 = 9 < 10$. Too many edges.

$K_{3,3}$: $V = 6$, $E = 9$. It's bipartite, so triangle-free. Need $E \leq 2V - 4 = 8 < 9$. Too many edges. □

Theorem 8.8 (Kuratowski's theorem)**.** *A graph is planar if and only if it contains no subdivision of K_5 or $K_{3,3}$.*

A **subdivision** inserts degree-2 vertices into edges—"stretching" them without changing the essential structure.

Theorem 8.9 (Four Color Theorem)**.** *Every planar graph can be colored with at most 4 colors.*

This was conjectured in 1852 and finally proved in 1976—with substantial computer assistance. It remains one of the most famous theorems to require computational verification.

Worked examples

Example 8.1 (Euler circuit check)**.** Does a connected graph with degrees $(2, 2, 2, 4, 4)$ have an Euler circuit?

Solution. All degrees are even, so yes. By Euler's theorem, an Euler circuit exists.

Example 8.2 (No Euler trail)**.** Does K_4 have an Euler circuit?

Solution. In K_4, every vertex has degree 3 (odd). All four vertices have odd degree. Since we need 0 or 2 odd-degree vertices for an Euler trail, K_4 has neither an Euler circuit nor an Euler trail.

Example 8.3 (Euler circuit exists)**.** Does K_5 have an Euler circuit?

Solution. In K_5, every vertex has degree 4 (even). All vertices have even degree, so an Euler circuit exists.

Example 8.4 (Adjacency matrix)**.** Find the adjacency matrix for the cycle C_4 on vertices $\{1, 2, 3, 4\}$.

Solution. Edges are $\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}$.

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Notice: symmetric, zero diagonal, each row sums to 2 (the degree).

Example 8.5 (Matrix entries sum)**.** Show that the sum of entries in an adjacency matrix of a simple graph equals $2|E|$.

Solution. Each edge $\{u, v\}$ contributes 1 to entry (u, v) and 1 to entry (v, u). Total contribution per edge: 2. Sum over all edges: $2|E|$.

Example 8.6 (Testing isomorphism)**.** Are these graphs isomorphic?

G_1: vertices $\{a, b, c, d\}$, edges $\{a, b\}, \{b, c\}, \{c, d\}, \{d, a\}$

G_2: vertices $\{1, 2, 3, 4\}$, edges $\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}$

Solution. Both are 4-cycles.

- Same vertex count: 4 ✓
- Same edge count: 4 ✓
- Same degree sequence: $(2, 2, 2, 2)$ ✓

The bijection $f : a \mapsto 1, b \mapsto 2, c \mapsto 3, d \mapsto 4$ preserves edges. So $G_1 \cong G_2$.

Example 8.7 (Non-isomorphic by edge count)**.** Prove C_5 and K_5 are not isomorphic.

Solution. C_5 has 5 edges. K_5 has $\binom{5}{2} = 10$ edges. Different edge counts, so not isomorphic.

Example 8.8 (Same degree sequence, not isomorphic)**.** Are two graphs with the same degree sequence necessarily isomorphic?

Solution. No! Consider C_6 (a 6-cycle) and $K_3 \sqcup K_3$ (two disjoint triangles). Both have degree sequence $(2, 2, 2, 2, 2, 2)$. But C_6 is connected and $K_3 \sqcup K_3$ is not. Not isomorphic.

Example 8.9 (Diameter of complete graph)**.** Find the diameter of K_n.

Solution. Every pair of vertices is adjacent, so $d(u, v) = 1$ for all $u \neq v$. Diameter $= 1$.

Example 8.10 (Finding an Euler trail)**.** Find an Euler trail in the graph with vertices $\{A, B, C, D\}$ and edges $\{A, B\}, \{A, C\}, \{B, C\}, \{B, D\}, \{C, D\}$.

Solution. First check degrees: $\deg(A) = 2$, $\deg(B) = 3$, $\deg(C) = 3$, $\deg(D) = 2$.

Odd-degree vertices: B and C (exactly 2). An Euler trail exists, starting at B and ending at C (or vice versa).

One Euler trail from B: $B \to A \to C \to B \to D \to C$.

Check: uses edges $\{B, A\}, \{A, C\}, \{C, B\}, \{B, D\}, \{D, C\}$—all 5 edges, each exactly once. ✓

Example 8.11 (Powers of adjacency matrix)**.** Compute A^2 for the path P_3 on vertices $\{1, 2, 3\}$ with edges $\{1, 2\}$ and $\{2, 3\}$. Interpret the result.

Solution.

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$A^2 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

$(A^2)_{ij}$ counts length-2 walks from i to j:

- $(A^2)_{11} = 1$: one walk $1 \to 2 \to 1$
- $(A^2)_{13} = 1$: one walk $1 \to 2 \to 3$
- $(A^2)_{22} = 2$: two walks $2 \to 1 \to 2$ and $2 \to 3 \to 2$

Common Mistake

Confusing Euler and Hamiltonian.

- Euler: visits every *edge* exactly once.
- Hamiltonian: visits every *vertex* exactly once.

Euler has a simple characterization (degree parity). Hamiltonian is NP-complete to decide.

Common Mistake

Thinking matching invariants proves isomorphism. Equal vertex count, edge count, and degree sequence are necessary but not sufficient. You must either construct an explicit bijection or find a property they differ on.

Going Deeper: Graphs Generate Categories

Graphs give rise to categories in a natural way. This perspective explains why adjacency matrices count paths. For more detail, see the Category Theory Companion, Weeks 6–7.

The Free Category on a Graph

Given a directed graph G, we can build a category $\mathbf{Path}(G)$:

- **Objects:** Vertices of G
- **Morphisms from u to v:** Directed paths from u to v
- **Composition:** Concatenation of paths
- **Identity at v:** The empty path (length 0) staying at v

This is the *free category* on G—the category with "just enough structure" to capture the graph.

Example. For the graph $1 \to 2 \to 3$:

- Morphisms $1 \to 3$: just the path $1 \to 2 \to 3$ (one morphism)
- Morphisms $2 \to 2$: just the empty path id_2
- Morphisms $3 \to 1$: none (no directed path backward)

Example. For a directed cycle $1 \to 2 \to 3 \to 1$:

- Morphisms $1 \to 1$: the empty path, once around, twice around, thrice around, and so on
- Infinitely many morphisms! The cycle generates unboundedly many paths.

Adjacency Matrices Count Morphisms

Here's the categorical insight: $(A^k)_{ij}$ counts the morphisms of length k from i to j in $\mathbf{Path}(G)$.
For $k = 2$:

$$(A^2)_{ij} = \sum_m A_{im} \cdot A_{mj}$$

Each term counts paths that go $i \to m \to j$ for intermediate vertex m. This is exactly composition of morphisms in $\mathbf{Path}(G)$.

Composition as Matrix Multiplication

The correspondence:

Category	Matrix
Composition of paths	Matrix multiplication
Length-k paths	A^k
Identity (length-0 path)	I (identity matrix)

Connected Components as a Functor

There's a functor $\pi_0 : \mathbf{Graph} \to \mathbf{Set}$ that sends a graph to its set of connected components. A graph homomorphism $f : G \to H$ induces a function $\pi_0(f)$ by sending each component of G to the component of H containing its image.
This functor captures the fact that connectivity is preserved by structure-preserving maps.

Graph Homomorphisms and Colorings

A graph k-coloring is secretly a graph homomorphism! If K_k is the complete graph on k vertices ("colors"), then a k-coloring of G is a homomorphism $G \to K_k$. Adjacent vertices in G must map to adjacent vertices in K_k—which are all pairs, so they must map to *different* colors.

Exercises: Graphs and Paths

1. For the graph $1 \to 2 \to 3$, list all morphisms in $\mathbf{Path}(G)$ from each vertex to each vertex.
2. For the directed 3-cycle $1 \to 2 \to 3 \to 1$, how many morphisms of length 3 are there from 1 to 1? Verify using A^3.
3. For a graph with edges $a \to b$, $b \to c$, $a \to c$: how many distinct paths are there from a to c?

4. Write the adjacency matrix for the 3-cycle. Compute A^2 and verify that $(A^2)_{11}$ equals the number of length-2 paths from 1 to 1.
5. A graph has edges $1 \to 2$, $2 \to 1$ (a 2-cycle). How many morphisms of length 4 are there from 1 to 1? Compute using A^4.
6. **Challenge:** If G is a directed acyclic graph (DAG), prove that $\mathbf{Path}(G)$ has finitely many morphisms between any two vertices.
7. Explain why a k-coloring of graph G corresponds to a graph homomorphism $G \to K_k$.
8. In the path category, explain why composition is associative and why the empty path is the identity.

Practice

1. Give the adjacency matrix for the 4-cycle C_4.
2. Construct two non-isomorphic graphs with the same degree sequence and prove they're not isomorphic.
3. Find an Euler trail in a graph with exactly two odd-degree vertices.
4. Show that the sum of entries in an adjacency matrix equals $2|E|$.
5. Prove: If G is a simple graph and $\overline{G}$ is its complement, then $G \cong \overline{G}$ implies $|V| \equiv 0$ or $1 \pmod 4$.
6. Compute A^2 for K_3 and interpret the entries.
7. Prove that every connected graph on n vertices has at least $n - 1$ edges.
8. Find the diameter of the n-cube Q_n.
9. Does $K_{3,3}$ have an Euler circuit? An Euler trail? Justify.
10. Prove that a graph is bipartite if and only if it contains no odd-length cycles.
11. How many automorphisms does the cycle C_n have?
12. Prove: If G is connected with exactly 2 odd-degree vertices, any Euler trail must start and end at those vertices.
13. Find $\chi(C_7)$ and $\chi(C_8)$.
14. Find the chromatic number of the wheel graph W_5 (a 5-cycle with a central vertex connected to all).
15. Use Euler's formula to find the number of faces in a connected planar graph with 10 vertices and 15 edges.
16. Prove that every planar graph has a vertex of degree at most 5.
17. Is the Petersen graph planar? Prove your answer.

18. A planar graph has 12 faces, and each face is bounded by exactly 3 edges. How many edges and vertices does it have?

19. Give a 3-coloring of K_4 minus one edge.

20. Prove: If G is planar with no cycles of length ≤ 4, then $E \leq \frac{5}{3}(V-2)$.

9 Week 8: Trees and Graph Algorithms

Or: "The simplest connected graphs, and how to navigate them efficiently"

Reading

Epp §10.4–10.6. Supplemental: matchings and flows.
Category theory companion: Week 8 (`category_theory_companion.pdf`).

Why Trees?

Trees are the "Goldilocks" graphs: they have just enough edges to be connected, but no more. Remove any edge and the graph falls apart. Add any edge and you create a cycle. This makes them the simplest connected structures, and remarkably useful.

File systems are trees. Organizational hierarchies are trees. Parse trees show how programs are structured. Decision trees guide classification. Family trees (well, directed acyclic graphs, really) trace ancestry. Binary search trees enable fast lookup. Spanning trees let us navigate networks efficiently.

The recursive structure of trees—a tree is either a single node or a node with subtrees—makes them perfect for recursive algorithms. But there's something deeper here: this recursive structure isn't just convenient, it's *definitional*. We can define trees by their construction rules (they're "algebraic datatypes"), and this gives us principled ways to define functions (structural recursion) and prove properties (structural induction). This week we'll see both the practical algorithms and the deeper pattern they exemplify.

Learning objectives

- Identify trees, forests, and rooted trees.
- Use characterizations of trees (connected + acyclic, $|E| = |V| - 1$, etc.).
- Understand m-ary trees and binary trees.
- **View trees as algebraic datatypes and define functions by structural recursion.**
- **Prove properties of trees using structural induction.**
- Find spanning trees using BFS and DFS.
- Apply shortest-path algorithms (Dijkstra's, Bellman-Ford).
- Find minimum spanning trees (Prim's, Kruskal's).

Key definitions and facts

What Makes a Tree a Tree?

Definition 9.1 (Tree)**.** A **tree** is a connected graph with no cycles. A **forest** is a graph with no cycles—each connected component is a tree.

Theorem 9.1 (Characterizations of trees)**.** *For a graph G on n vertices, the following are equivalent:*

1. *G is a tree (connected and acyclic).*
2. *G is connected and has exactly $n - 1$ edges.*
3. *G is acyclic and has exactly $n - 1$ edges.*
4. *There is exactly one path between any two vertices.*
5. *G is connected, but removing any edge disconnects it.*
6. *G is acyclic, but adding any edge creates exactly one cycle.*

These characterizations reveal the "barely connected" nature of trees. With $n - 1$ edges, you have the minimum needed for connectivity. One fewer and you're disconnected; one more and you have redundancy (a cycle).

Definition 9.2 (Rooted tree)**.** A **rooted tree** designates one vertex as the **root**, inducing a parent-child hierarchy: every non-root vertex has a unique parent (the neighbor closer to the root) and zero or more children.

- **Depth** of a vertex: distance from the root.
- **Height** of the tree: maximum depth.
- **Leaf**: a vertex with no children.
- **Internal vertex**: a vertex with at least one child.

The same underlying tree can be rooted at different vertices, producing different hierarchies. The choice of root is a choice of perspective.

Definition 9.3 (m-ary tree)**.** An **m-ary tree** is a rooted tree where every internal vertex has at most m children.

- **Binary tree**: $m = 2$ (at most two children each).
- **Full m-ary tree**: every internal vertex has exactly m children.
- **Complete m-ary tree**: full, with all leaves at the same depth.

Theorem 9.2 (Properties of full m-ary trees)**.** *For a full m-ary tree with i internal vertices:*

1. *Total vertices: $n = mi + 1$*
2. *Leaves: $\ell = (m - 1)i + 1$*
3. *Internal vertices: $i = \frac{n-1}{m} = \frac{\ell-1}{m-1}$*

Theorem 9.3 (Height bounds for binary trees)**.** *A binary tree with ℓ leaves has height h satisfying:*

$$\lceil \log_2 \ell \rceil \leq h \leq \ell - 1$$

Minimum height: complete binary tree (perfectly balanced). Maximum height: a degenerate "linear" tree (essentially a linked list).

This is why balanced trees matter: the difference between $O(\log n)$ and $O(n)$ operations.

Definition 9.4 (Spanning tree)**.** A **spanning tree** of a connected graph G is a subgraph that's a tree containing all vertices of G.

Theorem 9.4 (Existence of spanning trees)**.** *Every connected graph has a spanning tree.*

The proof is constructive: if there's a cycle, remove an edge from it. The graph stays connected (the cycle provided redundancy). Repeat until no cycles remain. You have a spanning tree.

Tree traversals

Definition 9.5 (Binary tree traversals)**.** Three classic ways to visit every node in a binary tree:

- **Preorder**: Visit root, traverse left subtree, traverse right subtree.
- **Inorder**: Traverse left subtree, visit root, traverse right subtree.
- **Postorder**: Traverse left subtree, traverse right subtree, visit root.

The names describe when you visit the root relative to its children. Preorder visits the root first ("pre"), inorder visits it between children ("in"), postorder visits it last ("post").

For a binary search tree, inorder traversal yields elements in sorted order. This isn't a coincidence—it's why we call it "in order."

Definition 9.6 (BFS and DFS)**.**

- **Breadth-First Search (BFS)**: Explore vertices layer by layer—all vertices at distance 1 from start, then distance 2, etc. Uses a queue.
- **Depth-First Search (DFS)**: Plunge as deep as possible before backtracking. Uses a stack (or recursion, which implicitly uses the call stack).

Both produce spanning trees of connected graphs.

BFS finds shortest paths (in unweighted graphs). DFS finds paths, but not necessarily shortest ones—it just finds *a* path.

Trees as Algebraic Datatypes

Here's a shift in perspective that will pay dividends throughout your mathematical career. Instead of thinking of binary trees as a particular kind of graph (connected, acyclic, each node has at most two children), think of them as defined by their *construction rules*:

Definition 9.7 (Binary tree as algebraic datatype)**.** A **binary tree with data of type** A is either:

- A **leaf** containing a value $a \in A$, or
- A **node** with a left subtree and a right subtree (both binary trees with data of type A)

That's it. Those are the only ways to build a binary tree.

This is called an *algebraic* or *inductive* definition. We specify the type by saying exactly how to construct its elements. You might write this in symbols as:

$$\text{Tree}(A) = A + \text{Tree}(A) \times \text{Tree}(A)$$

where $+$ means "or" (disjoint union) and $\times$ means "and" (product). The tree type is a solution to this equation—in fact, the *smallest* solution, which turns out to be important.

Why does this perspective matter? Because it gives us two powerful tools for free.

Defining Functions by Recursion on Structure

When you define a datatype by its constructors, you get a recipe for defining functions on it: just say what to do for each constructor.

Example 9.1 (Counting leaves)**.** To define a function $\mathit{countLeaves} : \text{Tree}(A) \to \mathbb{N}$, we just need two cases:

$$\begin{aligned} \mathit{countLeaves}(\text{Leaf } a) &= 1 \\ \mathit{countLeaves}(\text{Node } \ell\, r) &= \mathit{countLeaves}(\ell) + \mathit{countLeaves}(r) \end{aligned}$$

That's a complete, well-defined function. Every tree is built from these two constructors, so we've covered all cases.

Notice the pattern: for a leaf, we give a direct answer. For a node, we assume we already have answers for the subtrees (the recursive calls) and combine them. This is *structural recursion*—the recursion follows the structure of the datatype.

Example 9.2 (Tree height)**.**

$$\begin{aligned} \mathit{height}(\text{Leaf } a) &= 0 \\ \mathit{height}(\text{Node } \ell\, r) &= 1 + \max(\mathit{height}(\ell), \mathit{height}(r)) \end{aligned}$$

Example 9.3 (Summing a tree of numbers)**.**

$$\begin{aligned} \mathit{sum}(\text{Leaf } n) &= n \\ \mathit{sum}(\text{Node } \ell\, r) &= \mathit{sum}(\ell) + \mathit{sum}(r) \end{aligned}$$

The tree traversals we defined earlier? Those are exactly this pattern:

$$\begin{aligned} \mathit{preorder}(\text{Leaf } a) &= [a] \\ \mathit{preorder}(\text{Node } \ell\, r) &= \mathit{preorder}(\ell) +\!\!+ \mathit{preorder}(r) \end{aligned}$$

(For a tree where nodes also carry data, you'd insert the node's value at the appropriate position.)

Structural Induction: Proofs that Follow Construction

Here's the remarkable thing: the same pattern that lets us *define* functions also lets us *prove* things. This is **structural induction**.

Key Result

[Structural induction on trees] To prove a property P holds for all binary trees:

1. **Base case**: Prove $P(\text{Leaf } a)$ for any leaf.
2. **Inductive case**: Assume $P(\ell)$ and $P(r)$ for subtrees (the *inductive hypothesis*). Prove $P(\text{Node } \ell\, r)$.

If both hold, then $P(t)$ is true for every tree t.

Why does this work? Because every tree is built from leaves and nodes. A leaf satisfies P by the base case. A tree built from subtrees that satisfy P also satisfies P by the inductive case. Since trees are the *smallest* things you can build this way, every tree satisfies P.

Theorem 9.5. *For any binary tree t: $\mathit{countLeaves}(t) \leq 2^{\mathit{height}(t)}$.*

Proof. By structural induction.

Base case: For Leaf a: $\mathit{countLeaves}(\text{Leaf } a) = 1$ and $2^{\mathit{height}(\text{Leaf } a)} = 2^0 = 1$. So $1 \leq 1$. ✓

Inductive case: Assume the property holds for subtrees ℓ and r. That is:

$$\mathit{countLeaves}(\ell) \leq 2^{\mathit{height}(\ell)}$$
$$\mathit{countLeaves}(r) \leq 2^{\mathit{height}(r)}$$

For Node $\ell\, r$:

$$\begin{aligned}
\mathit{countLeaves}(\text{Node } \ell\, r) &= \mathit{countLeaves}(\ell) + \mathit{countLeaves}(r) \\
&\leq 2^{\mathit{height}(\ell)} + 2^{\mathit{height}(r)} \quad \text{(by IH)} \\
&\leq 2^{\max(\mathit{height}(\ell),\mathit{height}(r))} + 2^{\max(\mathit{height}(\ell),\mathit{height}(r))} \\
&= 2 \cdot 2^{\max(\mathit{height}(\ell),\mathit{height}(r))} \\
&= 2^{1+\max(\mathit{height}(\ell),\mathit{height}(r))} \\
&= 2^{\mathit{height}(\text{Node } \ell\, r)}
\end{aligned}$$

Therefore, by structural induction, the property holds for all trees. □

The Deep Connection

Structural recursion (for definitions) and structural induction (for proofs) are two sides of the same coin. When you define a function by structural recursion, the proof that it terminates is essentially a structural induction. When you prove something by structural induction, you're defining a "function" from trees to proofs.

This isn't just a metaphor. In constructive mathematics and type theory, proofs *are* programs and propositions *are* types. The correspondence goes by various names: Curry-Howard correspondence, propositions-as-types, proofs-as-programs. The "Going Deeper" section at the end of this week explores this further through the lens of initial algebras.

The Pattern Generalizes

Trees aren't special. Any algebraic datatype gets the same treatment:

Example 9.4 (Natural numbers)**.** $\mathbb{N}$ is defined by:

- Zero is a natural number
- If n is a natural number, then $\text{Succ}(n)$ is a natural number

Functions by recursion:

$$\begin{aligned}
\mathit{double}(\text{Zero}) &= \text{Zero} \\
\mathit{double}(\text{Succ } n) &= \text{Succ}(\text{Succ}(\mathit{double}(n)))
\end{aligned}$$

Proofs by induction: This is ordinary mathematical induction! The base case is $n = 0$; the inductive case assumes $P(n)$ and proves $P(n+1)$.

Example 9.5 (Lists)**.** $\text{List}(A)$ is defined by:

- Nil (empty list)
- Cons(a, xs) where $a \in A$ and xs is a list

The recursion pattern:

$$\begin{aligned} \mathit{length}(\text{Nil}) &= 0 \\ \mathit{length}(\text{Cons}(a, xs)) &= 1 + \mathit{length}(xs) \end{aligned}$$

Structural induction: base case for empty list, inductive case assumes property holds for the tail.

So when we prove things about trees "by induction," we're using exactly the same principle as when we prove things about natural numbers—just applied to a richer datatype.

Shortest path algorithms

Definition 9.8 (Weighted graph)**.** A **weighted graph** assigns a weight $w(e)$ to each edge. The weight of a path is the sum of its edge weights.

Definition 9.9 (Shortest path problem)**.** Given a weighted graph and vertices s and t, find a path from s to t with minimum total weight.

Theorem 9.6 (Dijkstra's algorithm)**.** *For graphs with non-negative edge weights, Dijkstra's algorithm finds shortest paths from a source to all other vertices.*

***Idea**: Maintain a set of vertices with known shortest distances. Repeatedly add the closest unvisited vertex, updating its neighbors' tentative distances.*

***Complexity**: $O((|V| + |E|) \log |V|)$ with a priority queue.*

Dijkstra is greedy: it commits to the closest vertex, confident that no later discovery will provide a shorter path. This confidence is justified only when all weights are non-negative.

Theorem 9.7 (Bellman-Ford algorithm)**.** *For graphs with possibly negative edges (but no negative cycles), Bellman-Ford finds shortest paths from a source.*

***Idea**: Relax all edges $|V| - 1$ times. "Relaxing" an edge means checking if we found a shorter path through it.*

***Complexity**: $O(|V| \cdot |E|)$.*

Slower than Dijkstra, but handles negative weights. It can also detect negative cycles (if relaxation still improves after $|V| - 1$ rounds, there's a negative cycle).

Theorem 9.8 (Floyd-Warshall algorithm)**.** *Finds shortest paths between* all *pairs of vertices.*

***Idea**: Dynamic programming. For each vertex k, update all pairs (i, j) to consider paths through k.*

***Complexity**: $O(|V|^3)$.*

Minimum spanning trees

Definition 9.10 (Minimum spanning tree (MST))**.** For a connected weighted graph, a **minimum spanning tree** is a spanning tree with minimum total edge weight.

MSTs are useful when you want to connect everything at minimum cost: network design, circuit layout, clustering.

Theorem 9.9 (Cut property)**.** *For any cut (partition of vertices into two sets), the minimum-weight edge crossing the cut belongs to some MST.*

This is the key insight behind MST algorithms: it's always safe to include a minimum-weight edge that connects two components.

Theorem 9.10 (Prim's algorithm)**.** *Start from any vertex. Repeatedly add the minimum-weight edge connecting the growing tree to a new vertex.*

***Complexity**: $O((|V| + |E|) \log |V|)$ with a priority queue.*

Prim grows a single tree outward, like a drop of water spreading.

Theorem 9.11 (Kruskal's algorithm)**.** *Sort edges by weight. Add edges in order, skipping any that would create a cycle.*

***Complexity**: $O(|E| \log |E|)$, dominated by sorting.*

Kruskal grows a forest that gradually merges into a single tree.

Matchings in bipartite graphs

Definition 9.11 (Matching)**.** A **matching** is a set of edges with no shared endpoints—each vertex is in at most one edge. A **maximum matching** has the largest possible size. A **perfect matching** covers every vertex.

Think of matchings as pairings: jobs to applicants, dancers to partners, tasks to machines.

Theorem 9.12 (Hall's marriage theorem)**.** *Let $G = (X \cup Y, E)$ be a bipartite graph. There's a matching covering all of X if and only if for every subset $S \subseteq X$:*

$$|N(S)| \geq |S|$$

where $N(S)$ is the set of neighbors of S in Y.

In words: every subset of X must have at least as many neighbors as members. If 5 people can only apply to 4 jobs, you can't match everyone.

Example 9.6. $X = \{1, 2, 3\}$, $Y = \{a, b, c\}$, edges $\{1a, 1b, 2b, 3b, 3c\}$.

Check Hall's condition:

- $|N(\{1\})| = 2 \geq 1$ ✓
- $|N(\{2\})| = 1 \geq 1$ ✓
- $|N(\{3\})| = 2 \geq 1$ ✓
- $|N(\{1, 2\})| = 2 \geq 2$ ✓
- $|N(\{2, 3\})| = 2 \geq 2$ ✓
- $|N(\{1, 3\})| = 3 \geq 2$ ✓
- $|N(\{1, 2, 3\})| = 3 \geq 3$ ✓

Hall's condition holds, so a matching covering X exists: $\{1a, 2b, 3c\}$.

Remark. Maximum matchings can be found via **augmenting paths**: paths that alternate between unmatched and matched edges. Flipping along such a path increases the matching size by one.

Network flows

Definition 9.12 (Flow network)**.** A **flow network** is a directed graph with a source s, sink t, and capacities $c(u, v) \geq 0$ on edges.

A **flow** assigns values $f(u, v)$ satisfying:

- **Capacity constraint**: $0 \leq f(u, v) \leq c(u, v)$
- **Conservation**: For all $v \neq s, t$: flow in = flow out

The **value** of the flow is the total leaving s.

Definition 9.13 (Cut)**.** An s-t **cut** partitions vertices into (S, T) with $s \in S$ and $t \in T$. Its **capacity** is the sum of capacities from S to T.

Theorem 9.13 (Max-flow min-cut)**.** *The maximum flow value equals the minimum cut capacity.*

This is a beautiful duality result. Flow can never exceed cut capacity (the cut is a bottleneck). The theorem says the bottleneck is always achievable.

Remark. The Ford-Fulkerson method finds max flow by repeatedly finding augmenting paths in the residual graph until none remain.

Worked examples

Example 9.7 (Tree edge count by induction)**.** Prove that a tree on n vertices has exactly $n - 1$ edges.

Proof by induction.

Base case: $n = 1$. A single vertex has 0 edges. $0 = 1 - 1$. ✓

Inductive step: Assume true for trees with k vertices. Let T be a tree with $k + 1$ vertices.

Every tree has at least one leaf (vertex of degree 1)—if every vertex had degree ≥ 2, following edges would eventually create a cycle.

Remove a leaf v and its edge. The result T' has k vertices and is still a tree (removing a leaf doesn't disconnect or create cycles).

By induction, T' has $k - 1$ edges. Adding back v's edge gives k edges for T.

$k = (k + 1) - 1$. ✓

Example 9.8 (Leaves in a full binary tree)**.** A full binary tree has 15 vertices. How many are leaves?

Solution. For full binary trees: $\ell = i + 1$ where i is the number of internal vertices.

Also $n = i + \ell = 15$, so $i + (i + 1) = 15$, giving $i = 7$ and $\ell = 8$.

Alternatively: $\ell = \frac{(m-1)n+1}{m} = \frac{15+1}{2} = 8$.

Example 9.9 (Maximum leaves at height h)**.** How many leaves can a full m-ary tree of height h have?

Solution. A complete m-ary tree of height h has all leaves at depth h. At depth d, there are at most m^d vertices. Maximum leaves: m^h.

Example 9.10 (Spanning tree of K_4)**.** Find a spanning tree of K_4.

Solution. K_4 has 4 vertices and 6 edges. A spanning tree needs 3 edges.

Keep edges $\{1, 2\}, \{2, 3\}, \{3, 4\}$. This is the path $1 - 2 - 3 - 4$, a spanning tree.

(Many other choices work—there are 16 spanning trees of K_4.)

Example 9.11 (Why removing an edge disconnects a tree)**.** Explain why removing any edge from a tree disconnects it.

Solution. In a tree, there's exactly one path between any two vertices. Edge $\{u, v\}$ is on the unique path from u to v. Remove it and no path remains—u and v become disconnected.

Example 9.12 (Dijkstra's algorithm)**.** Run Dijkstra on vertices $\{A, B, C, D\}$ with weighted edges: A-B (1), A-C (4), B-C (2), B-D (5), C-D (1). Find shortest paths from A.

Solution.

1. Initialize: $d[A] = 0$, others $= \infty$.
2. Process A: Update $d[B] = 1$, $d[C] = 4$.
3. Process B (closest unvisited): Update $d[C] = \min(4, 1 + 2) = 3$, $d[D] = 6$.
4. Process C: Update $d[D] = \min(6, 3 + 1) = 4$.
5. Process D: No updates.

Shortest distances: $A \to A$: 0, $A \to B$: 1, $A \to C$: 3, $A \to D$: 4.

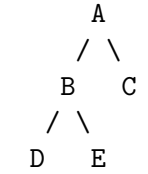
Example 9.13 (Kruskal's algorithm)**.** Find an MST of vertices $\{A, B, C, D\}$ with edges: A-B (3), A-C (1), A-D (4), B-C (2), B-D (5), C-D (6).

Solution.

1. Sort edges: A-C (1), B-C (2), A-B (3), A-D (4), B-D (5), C-D (6).
2. Add A-C (1): No cycle. MST so far: $\{A$-$C\}$.
3. Add B-C (2): No cycle. MST: $\{A$-C, B-$C\}$.
4. Add A-B (3): Would create cycle A-B-C-A. Skip.
5. Add A-D (4): No cycle. MST: $\{A$-C, B-C, A-$D\}$.

Total weight: $1 + 2 + 4 = 7$.

Example 9.14 (Tree traversals)**.** Give preorder, inorder, and postorder traversals of:



Solution.

- **Preorder** (root, left, right): A, B, D, E, C
- **Inorder** (left, root, right): D, B, E, A, C
- **Postorder** (left, right, root): D, E, B, C, A

Example 9.15 (Forest edge count)**.** Prove: A forest with n vertices and k components has $n - k$ edges.

Solution. Each component is a tree. If component i has n_i vertices, it has $n_i - 1$ edges.

Total edges: $\sum_{i=1}^{k}(n_i - 1) = \sum n_i - k = n - k$.

Common Mistake

Using Dijkstra with negative weights. Dijkstra assumes non-negative weights. With negative edges, it can give wrong answers. Use Bellman-Ford instead.

Common Mistake

Thinking $n-1$ edges implies tree. Having exactly $n-1$ edges is necessary but not sufficient. You also need the graph to be connected (or acyclic). A disconnected graph with $n-1$ edges is a forest, not a tree.

Going Deeper: Trees as Initial Algebras and the Universality of Fold

There's a deep reason why recursive functions on trees always terminate, and why the "fold" pattern is universal. Trees are *initial algebras*, and fold is the unique morphism from an initial object. For more detail, see the Category Theory Companion, Week 8.

The recursive structure of trees. A binary tree is either:

- A leaf (containing data), or
- A node with a left subtree and a right subtree

As an equation: $\text{Tree}(A) = A + \text{Tree}(A) \times \text{Tree}(A)$.

Here $+$ means "or" (disjoint union) and $\times$ means "and" (product). This equation *defines* the type.

Algebras for tree-building. An "algebra" for this structure over set B consists of:

- A function $\mathit{leaf} : A \to B$ (what to do with leaves)
- A function $\mathit{node} : B \times B \to B$ (how to combine subtree results)

The fold (catamorphism). Given any algebra $(\mathit{leaf}, \mathit{node})$ over B, there's a *unique* function $\mathit{fold} : \text{Tree}(A) \to B$ satisfying:

$$\mathit{fold}(\text{Leaf } a) = \mathit{leaf}(a) \qquad \mathit{fold}(\text{Node } \ell\, r) = \mathit{node}(\mathit{fold}(\ell), \mathit{fold}(r))$$

Examples of fold.

- **Sum leaves**: $\mathit{leaf}(a) = a$, $\mathit{node}(x, y) = x + y$
- **Count leaves**: $\mathit{leaf}(a) = 1$, $\mathit{node}(x, y) = x + y$
- **Tree height**: $\mathit{leaf}(a) = 0$, $\mathit{node}(x, y) = 1 + \max(x, y)$
- **Preorder list**: $\mathit{leaf}(a) = [a]$, $\mathit{node}(x, y) = x \mathbin{+\!\!+} y$

All tree traversals are folds with appropriate algebra choices.

Why recursion terminates. The tree type is the *initial* algebra—the "smallest" solution to the recursive equation. Uniqueness of fold means exactly one way to recursively compute any result. If recursion didn't terminate, no function would exist. If there were multiple computation paths, uniqueness would fail.

The pattern generalizes.

- Lists: $\text{List}(A) = 1 + A \times \text{List}(A)$. Fold is *foldr*.
- Natural numbers: $\mathbb{N} = 1 + \mathbb{N}$. Fold is primitive recursion.

Understanding data as initial algebras explains why structural recursion is well-founded, enables optimizations like fold fusion, and connects programming to category theory.

Exercises: Folds and Algebras

1. The "sum" function on lists adds elements. What's the base case (empty list maps to what)? What's the combining function?
2. The "length" function counts elements. Define it as a fold.
3. Define `map f` as a fold. What are the base case and combining function?
4. For natural numbers with Zero and Succ: define addition $n + m$ by folding over n with m fixed.
5. Define "tree height" as a fold on binary trees.
6. Express inorder traversal as a fold. (Hint: where does the root's data go relative to left and right results?)
7. **Challenge**: The "fold fusion" law says if certain conditions hold, $h \circ \mathit{fold}_g = \mathit{fold}_{g'}$. For lists, verify this for $h(n) = 2n$, $f(x, y) = x + y$, $z = 0$.

Practice

1. How many leaves can a full m-ary tree of height h have?
2. Find a spanning tree of K_5.
3. Explain why removing any edge from a tree disconnects it.
4. Run Dijkstra's algorithm on a 5-vertex weighted graph of your choice.
5. A full binary tree has 31 vertices. How many are leaves? Internal vertices?
6. Prove: Every tree with ≥ 2 vertices has at least 2 leaves.
7. Use Prim's algorithm to find an MST (create your own example).
8. Prove that every tree is bipartite.
9. Give BFS and DFS spanning trees of C_4 starting from vertex 1.
10. Prove: If a graph has n vertices and fewer than $n - 1$ edges, it's not connected.
11. How many spanning trees does C_n have?
12. Prove: In any tree, the sum of degrees equals $2(n - 1)$.
13. Let $X = \{1, 2, 3\}$, $Y = \{a, b, c\}$ with edges $\{1a, 1b, 2b, 2c, 3c\}$. Does a matching covering X exist? Find one or explain why not.

14. Find a maximum matching in the bipartite graph with $X = \{u, v, w\}$, $Y = \{x, y, z\}$, edges $\{ux, uy, vx, vy, wz\}$.

15. Compute the maximum flow in a network with edges $s \to a$ (3), $s \to b$ (2), $a \to b$ (1), $a \to t$ (2), $b \to t$ (3). Find a minimum cut.

16. **(Structural induction)** Define a function $\mathit{mirror} : \text{Tree}(A) \to \text{Tree}(A)$ that swaps left and right subtrees at every node. Then prove by structural induction that $\mathit{mirror}(\mathit{mirror}(t)) = t$ for all trees t.

17. **(Structural induction)** Prove: For any binary tree t, the number of nodes is $2 \cdot \mathit{countLeaves}(t) - 1$. (Hint: nodes = internal vertices + leaves.)

18. **(Recursive definition)** Define a function $\mathit{flatten} : \text{Tree}(A) \to \text{List}(A)$ that collects all values in a tree into a list. How does your definition relate to tree traversals?

19. **(Structural induction)** Prove: $\mathit{length}(\mathit{flatten}(t)) = \mathit{countLeaves}(t)$ for all trees t.

10 Week 9: Regular Expressions and Finite Automata

Or: "Pattern matching and the simplest model of computation"

Reading

Epp §12.1–12.3. Supplemental: CFGs and PDAs.
Category theory companion: Week 9 (`category_theory_companion.pdf`).

Why Automata Theory?

Every programmer uses regular expressions: searching text, validating input, parsing logs. But regex is more than a practical tool—it's a window into the theory of computation.

Finite automata are the simplest computational model: machines with finite memory that read input symbol by symbol and either accept or reject. Remarkably, these simple machines capture exactly the same patterns as regular expressions. This equivalence is beautiful and useful.

But finite automata can't do everything. Some patterns—like "balanced parentheses" or "equal numbers of 0s and 1s"—are provably impossible for them. Understanding *why* reveals fundamental limits of computation and motivates more powerful models.

Learning objectives

- Define languages, alphabets, and strings.
- Construct regular expressions to describe languages.
- Build deterministic finite automata (DFAs) and trace their execution.
- Convert between regular expressions and DFAs.
- Minimize DFAs by merging equivalent states.
- Use the pumping lemma to prove non-regularity.

Key definitions and facts

Languages and Strings

Definition 10.1 (Alphabet, string, language)**.**

- An **alphabet** Σ is a finite, nonempty set of symbols.
- A **string** (or word) over Σ is a finite sequence of symbols from Σ.
- The **empty string** ε has length 0.
- Σ^* denotes all strings over Σ (including ε).
- A **language** over Σ is a subset $L \subseteq \Sigma^*$.

Languages can be finite or infinite. The language "all binary strings of length 3" is finite (8 strings). The language "all binary strings starting with 1" is infinite.

Definition 10.2 (String operations)**.**

- **Length**: $|w|$ is the number of symbols in w.

- **Concatenation**: $w_1 w_2$ appends w_2 to w_1. Note: $w\varepsilon = \varepsilon w = w$.
- **Exponentiation**: $w^n = \underbrace{ww \cdots w}_{n \text{ times}}$; $w^0 = \varepsilon$.
- **Reversal**: w^R is w written backwards.

Definition 10.3 (Language operations)**.** For languages $L, L_1, L_2 \subseteq \Sigma^*$:

- **Union**: $L_1 \cup L_2$ — strings in either language
- **Concatenation**: $L_1 L_2 = \{w_1 w_2 : w_1 \in L_1, w_2 \in L_2\}$ — all ways to concatenate
- **Kleene star**: $L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 \cup \cdots$ — zero or more repetitions
- **Kleene plus**: $L^+ = L \cup L^2 \cup L^3 \cup \cdots = LL^*$ — one or more repetitions

Regular expressions

Definition 10.4 (Regular expression)**.** A **regular expression** (regex) over alphabet Σ is defined recursively:

1. $\emptyset$ denotes the empty language $\{\}$ (no strings).
2. ε denotes the language $\{\varepsilon\}$ (just the empty string).
3. For each $a \in \Sigma$, a denotes the language $\{a\}$.
4. If r_1 and r_2 are regexes:
   - $(r_1 \mid r_2)$ denotes $L(r_1) \cup L(r_2)$ (union/alternation)
   - $(r_1 r_2)$ denotes $L(r_1)L(r_2)$ (concatenation)
   - $(r_1)^*$ denotes $L(r_1)^*$ (Kleene star)

Definition 10.5 (Precedence)**.** Operator precedence (highest to lowest): Kleene star $*$, concatenation, union $|$.

So $ab^* \mid c$ means $(a(b^*)) \mid c$—not $(ab)^* \mid c$ or $a(b^* \mid c)$.

Example 10.1 (Common regex patterns)**.** Over $\Sigma = \{0, 1\}$:

- All strings: $(0 \mid 1)^*$
- Strings starting with 1: $1(0 \mid 1)^*$
- Strings ending with 01: $(0 \mid 1)^*01$
- Exactly one 1: 0^*10^*
- At least one 0: $(0 \mid 1)^*0(0 \mid 1)^*$
- Even-length strings: $((0 \mid 1)(0 \mid 1))^*$

Deterministic finite automata

Definition 10.6 (DFA)**.** A **deterministic finite automaton** (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- Q is a finite set of **states**
- Σ is the input **alphabet**
- $\delta : Q \times \Sigma \to Q$ is the **transition function**
- $q_0 \in Q$ is the **start state**
- $F \subseteq Q$ is the set of **accept states**

A DFA is "deterministic" because from any state, each input symbol leads to exactly one next state. No choices, no ambiguity.

Definition 10.7 (DFA execution)**.** A DFA **accepts** string $w = a_1 a_2 \cdots a_n$ if there's a sequence of states $r_0, r_1, \ldots, r_n$ such that:

1. $r_0 = q_0$ (start in the start state)
2. $r_{i+1} = \delta(r_i, a_{i+1})$ (follow transitions)
3. $r_n \in F$ (end in an accept state)

The **language** $L(M)$ is the set of strings M accepts.

Definition 10.8 (State diagram)**.** A DFA can be drawn as a directed graph:

- Vertices are states
- Edge from q to q' labeled a means $\delta(q, a) = q'$
- An arrow from nowhere points to the start state
- Accept states have double circles

Nondeterministic finite automata

Definition 10.9 (NFA)**.** A **nondeterministic finite automaton** (NFA) relaxes the DFA rules:

- Transition function: $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$
- From a state, there can be 0, 1, or many transitions on the same symbol
- ε-transitions allow state changes without consuming input

An NFA accepts if *some* computational path leads to an accept state.

NFAs seem more powerful—they can "guess" the right path. But they're not.

Theorem 10.1 (NFA-DFA equivalence)**.** *For every NFA, there's a DFA accepting the same language. The* ***subset construction*** *converts an n-state NFA to a DFA with at most 2^n states.*

The subset construction treats sets of NFA states as single DFA states. It tracks all possible states the NFA might be in.

Regular languages

Definition 10.10 (Regular language)**.** A language is **regular** if it's recognized by some DFA (equivalently, by some NFA, or described by some regex).

Theorem 10.2 (Kleene's theorem)**.** *For a language L, the following are equivalent:*

1. *L is described by a regular expression.*
2. *L is recognized by a DFA.*
3. *L is recognized by an NFA.*

This is a remarkable three-way equivalence. Three different formalisms—algebraic (regex), operational (DFA), and nondeterministic (NFA)—capture exactly the same class of languages.

Theorem 10.3 (Closure properties)**.** *Regular languages are closed under:*

- *Union, concatenation, Kleene star (by definition of regex)*
- *Complement: swap accept/non-accept states in DFA*
- *Intersection: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ (De Morgan)*
- *Reversal: reverse all transitions and swap start/accept to get an NFA (possibly with multiple start states or ε-transitions), then determinize*

DFA minimization

Different DFAs can recognize the same language. Is there a "best" one?

Definition 10.11 (Equivalent states)**.** States p and q are **equivalent** if for all strings w:

$$\hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$$

Equivalent states are indistinguishable—no input can tell them apart.

Theorem 10.4 (Minimization)**.** *Every regular language has a unique minimum-state DFA (up to isomorphism), obtained by merging equivalent states.*

Definition 10.12 (Table-filling algorithm)**.** To find equivalent states:

1. Mark pairs (p, q) where exactly one is in F as distinguishable.
2. Repeat: Mark (p, q) if for some $a \in \Sigma$, $(\delta(p, a), \delta(q, a))$ is already marked.
3. Unmarked pairs are equivalent; merge them.

Non-regular languages

Not all languages are regular. The pumping lemma helps prove this.

Theorem 10.5 (Pumping lemma for regular languages)**.** *If L is regular, there exists a "pumping length" p such that any string $w \in L$ with $|w| \geq p$ can be split as $w = xyz$ where:*

1. *$|y| > 0$ (the "pump" is non-empty)*
2. *$|xy| \leq p$ (the pump is near the start)*
3. *For all $i \geq 0$, $xy^iz \in L$ (pumping preserves membership)*

The intuition: if a DFA reads a long enough string, it must revisit some state. The substring between visits can be repeated (or removed) without changing acceptance.

Proof Strategy

To prove L is not regular:

1. Assume L is regular (for contradiction).
2. Let p be the pumping length.
3. Choose $w \in L$ with $|w| \geq p$ (often depending on p).
4. Show that for any split $w = xyz$ satisfying conditions 1 and 2, some $xy^iz \notin L$.
5. Contradiction: L is not regular.

Context-free grammars

Regular languages can't handle nested structure. Context-free grammars can.

Definition 10.13 (Context-free grammar)**.** A CFG is a 4-tuple (V, Σ, R, S) where:

- V is a set of variables (nonterminals)
- Σ is a set of terminals (the alphabet)
- R is a set of production rules $A \to \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$
- $S \in V$ is the start symbol

The language $L(G)$ is all strings derivable from S.

Definition 10.14 (Derivation)**.** We write $S \Rightarrow^* w$ if w can be derived from S by repeatedly applying rules. A **parse tree** records the derivation structure.

Example 10.2. The grammar $S \to aSb \mid \varepsilon$ generates $L = \{a^nb^n : n \geq 0\}$.
Derivation of $aabb$: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$.

Remark. A grammar is **ambiguous** if some string has multiple parse trees. Ambiguity complicates parsing and semantics.

Pushdown automata

Definition 10.15 (Pushdown automaton (PDA))**.** A **pushdown automaton** is an NFA with a stack. It can push, pop, or read the stack top when transitioning.

Theorem 10.6 (CFG-PDA equivalence)**.** *A language is context-free if and only if it's recognized by a pushdown automaton.*

The stack provides unbounded memory (unlike the finite memory of DFAs), enabling recognition of nested structures.

Worked examples

Example 10.3 (DFA for even number of 0s)**.** Design a DFA over $\{0, 1\}$ that accepts strings with an even number of 0s.

Solution. Two states: q_e (even 0s seen) and q_o (odd 0s seen).

- Start state: q_e (zero 0s is even)
- Accept states: $\{q_e\}$
- Transitions: On 0, toggle. On 1, stay.

Formally: $\delta(q_e, 0) = q_o$, $\delta(q_o, 0) = q_e$, $\delta(q_e, 1) = q_e$, $\delta(q_o, 1) = q_o$.

Example 10.4 (Regex for strings ending in 01)**.** Write a regex for binary strings ending with 01.

Solution. (0 | 1)*01

Any sequence, followed by 01.

Example 10.5 (DFA for no substring bb)**.** Construct a DFA over $\{a, b\}$ accepting strings with no bb substring.

Solution. Three states tracking recent history:

- q_0: Start, or last symbol was a
- q_1: Last symbol was b
- q_{dead}: Saw bb, reject forever

Transitions:

- From q_0: on a stay, on b go to q_1
- From q_1: on a go to q_0, on b go to q_{dead}
- From q_{dead}: stay forever

Accept states: $\{q_0, q_1\}$.

Example 10.6 (Intersection of regular languages)**.** Prove that the intersection of two regular languages is regular.

Solution. Let L_1 have DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and L_2 have $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Build the product DFA $M = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), F_1 \times F_2)$ where:

$$\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$$

M accepts w iff both M_1 and M_2 accept w, so $L(M) = L_1 \cap L_2$.

Example 10.7 (DFA minimization)**.** Minimize a DFA with states $\{A, B, C, D\}$, start A, accept $\{C\}$, transitions: $\delta(A,0) = B$, $\delta(A,1) = C$, $\delta(B,0) = B$, $\delta(B,1) = C$, $\delta(C,0) = D$, $\delta(C,1) = C$, $\delta(D,0) = D$, $\delta(D,1) = C$.

Solution. Using table-filling:

1. Mark $(A,C), (B,C), (D,C)$ (accept vs non-accept).
2. Check (A,B): $\delta(A,0) = B$, $\delta(B,0) = B$ (same); $\delta(A,1) = C$, $\delta(B,1) = C$ (same). Not distinguishable.
3. Check (A,D) and (B,D): similarly not distinguishable.
4. A, B, D are equivalent. Merge them.

Minimal DFA has 2 states: $\{A, B, D\}$ and $\{C\}$.

Example 10.8 (Pumping lemma: $\{0^n1^n\}$)**.** Prove $L = \{0^n1^n : n \geq 0\}$ is not regular.

Proof. Assume L is regular with pumping length p.

Choose $w = 0^p1^p \in L$. Since $|w| = 2p \geq p$, the pumping lemma applies.

Write $w = xyz$ with $|y| > 0$ and $|xy| \leq p$. Since $|xy| \leq p$ and w starts with p zeros, $y = 0^k$ for some $k > 0$.

Consider $xy^2z = 0^{p+k}1^p$. This has more 0s than 1s, so $xy^2z \notin L$.

Contradiction. L is not regular. □

Example 10.9 (Regex for at least two 0s)**.** Write a regex for binary strings with at least two 0s.

Solution. $(0 \mid 1)^*0(0 \mid 1)^*0(0 \mid 1)^*$

Two 0s, with anything before, between, and after.

Example 10.10 (DFA for divisibility by 3)**.** Design a DFA for binary strings representing numbers divisible by 3.

Solution. Track value mod 3 as we read bits left-to-right. If current value is v and we read bit b, new value is $2v + b \pmod 3$.

States q_0, q_1, q_2 for value mod 3.

- Start: q_0 (value 0)
- Accept: $\{q_0\}$
- From q_0: on 0 go to q_0, on 1 go to q_1
- From q_1: on 0 go to q_2, on 1 go to q_0
- From q_2: on 0 go to q_1, on 1 go to q_2

Test: $110_2 = 6$. Path: $q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_0$. Accept. ✓

Example 10.11 (Pumping lemma: $\{ww\}$)**.** Prove $L = \{ww : w \in \{0,1\}^*\}$ is not regular.

Proof. Assume L is regular with pumping length p.

Choose $s = 0^p10^p1 \in L$ (where $w = 0^p1$). We have $|s| \geq p$.

Write $s = xyz$ with $|y| > 0$, $|xy| \leq p$. Since $|xy| \leq p$ and s starts with p zeros, $y = 0^k$ for some $k \geq 1$.

Consider $xy^0z = 0^{p-k}10^p1$. The first half has $p - k$ zeros before its 1; the second has p zeros. Since $k \geq 1$, these halves differ, so $xy^0z \notin L$.

Contradiction. L is not regular. □

Common Mistake

Confusing $\emptyset$ and ε. $\emptyset$ is the empty language (no strings accepted). ε is the language containing exactly the empty string. They're very different: $|\emptyset| = 0$ but $|\{\varepsilon\}| = 1$.

Common Mistake

Using pumping lemma to prove regularity. The pumping lemma only proves non-regularity. Satisfying the pumping condition is necessary but not sufficient for regularity.

Going Deeper: Duality and Observation

Week 8 introduced *initial algebras* for building recursive data. This week, we glimpse the dual: *coalgebras* for observing behavior. For more detail, see the Category Theory Companion, Week 9.

Algebras vs Coalgebras

Recall: an algebra has a structure map $F(A) \to A$—it builds up data:

- List algebra: $([], \text{cons})$ tells how to construct lists
- Tree algebra: $(\text{leaf}, \text{node})$ tells how to construct trees

A coalgebra reverses the arrow: $A \to F(A)$—it *observes* or decomposes:

- Given a state, observe something about it
- The arrow points "outward"

DFAs as Coalgebras

A DFA state can be observed:

1. Is it accepting? (Output: yes/no)
2. Where does each input lead? (Transitions)

This gives a function: $Q \to \{0, 1\} \times Q^{\Sigma}$.
A DFA is a coalgebra for this pattern. The accept/reject output and transition structure are observations we can make of any state.

Automata as Monoid Actions

The set Σ^* of all strings forms a **monoid** under concatenation. A DFA extends transitions to whole strings:

$$\delta^* : Q \times \Sigma^* \to Q$$

This is a *monoid action*: the monoid Σ^* acts on the state set Q. Categorically, it's a functor from the one-object category Σ^* to **Set**.

Streams: Another Coalgebra

An infinite stream can be observed:

1. What's the first element? (head)
2. What's the rest? (tail)

Structure: $\text{Stream}(A) \to A \times \text{Stream}(A)$.
Contrast with lists: lists are built up (algebra), streams are observed (coalgebra).

The Duality Pattern

	Algebra	**Coalgebra**
Structure map	$F(A) \to A$	$A \to F(A)$
Intuition	Constructors	Observers
Universal object	Initial	Final
Universal map	Fold	Unfold
Data	Finite	Potentially infinite
Example	Lists, trees	Streams, automata

Bisimulation

Two DFA states are *bisimilar* if no observations distinguish them:

- Same accept/reject status
- Transitions lead to bisimilar states

This is exactly what DFA minimization computes: merge bisimilar states.

Exercises: Duality and Observation

1. For an infinite stream $[0, 1, 2, 3, \ldots]$, what is head(tail(tail($\cdot$)))?
2. For the "even 0s" DFA, write out the coalgebra structure: for each state, give $(\text{accept?}, \text{transitions})$.
3. Why can infinite streams exist but (in a total language) not "infinite lists"?
4. In DFA minimization, states A, B, D were merged. Verify A and B are bisimilar.
5. **Challenge**: The Brzozowski derivative $\partial_a(L) = \{w : aw \in L\}$ is the "observe a transition" operation. Show this matches the coalgebra view.

Practice

1. Write a regex for binary strings ending with 01.
2. Construct a DFA for strings over $\{a, b\}$ containing no bb substring.
3. Minimize a 4-state DFA of your choosing.

4. Prove that the intersection of two regular languages is regular.
5. Write a regex for binary strings with at least two 0s.
6. Design a DFA accepting strings where the number of as is divisible by 3.
7. Prove that palindromes $L = \{w : w = w^R\}$ over $\{0, 1\}$ are not regular.
8. Convert the regex $(a \mid b)^* aba$ to an NFA.
9. Show that if L is regular, then $L^R = \{w^R : w \in L\}$ is regular.
10. Design a DFA for binary strings representing numbers divisible by 3.
11. Prove that $L = \{a^{n^2} : n \geq 0\}$ is not regular.
12. Given DFAs for L_1 and L_2, construct a DFA for $L_1 \setminus L_2$.
13. Give a CFG for $\{0^n 1^n : n \geq 0\}$ and derive 000111.
14. Show the grammar $S \to SS \mid a$ is ambiguous by finding two parse trees for aa.
15. Describe a PDA for $\{a^n b^n : n \geq 0\}$ and trace its stack on input $aabb$.

11 Week 10: Analysis of Algorithm Efficiency

Or: "How long will this take, and why constants don't matter"

Reading

Epp §11.1–11.5.
Category theory companion: Week 10 (`category_theory_companion.pdf`).

Why Algorithm Analysis?

You've written a program. It works. But is it *fast*? Will it still work on inputs a thousand times larger? A million times? Understanding how running time grows with input size separates programs that scale from programs that don't.

Algorithm analysis gives us a language for discussing efficiency without getting bogged down in implementation details. When we say "merge sort is $O(n \log n)$" and "bubble sort is $O(n^2)$," we're capturing something essential about these algorithms that holds regardless of the programming language, the compiler, or the hardware.

This week introduces asymptotic notation (big-O and friends), techniques for analyzing loops and recursion, and the master theorem for solving divide-and-conquer recurrences. These tools are fundamental to thinking about algorithms.

Learning objectives

- Compare growth rates of functions using limits and dominance.
- Apply big-O, big-Ω, and big-Θ notation correctly.
- Analyze the time complexity of loops and nested loops.
- Solve recurrences using expansion, substitution, and the master theorem.
- Classify algorithms by their complexity class.

Key definitions and facts

Asymptotic Notation

Definition 11.1 (Asymptotic notation)**.** Let $f, g : \mathbb{N} \to \mathbb{R}^+$ be functions.

Big-O (upper bound): $f(n) = O(g(n))$ if there exist constants $c > 0$ and n_0 such that:

$$f(n) \leq c \cdot g(n) \quad \text{for all } n \geq n_0$$

Big-Omega (lower bound): $f(n) = \Omega(g(n))$ if there exist constants $c > 0$ and n_0 such that:

$$f(n) \geq c \cdot g(n) \quad \text{for all } n \geq n_0$$

Big-Theta (tight bound): $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

The notation $f(n) = O(g(n))$ is a slight abuse—it's not really "equals" but "is." Think of it as "f is big-O of g" or "f grows no faster than g."

Big-O gives upper bounds. Big-Ω gives lower bounds. Big-Θ says the function is bounded both above and below—it captures the exact growth rate.

Theorem 11.1 (Limit test)**.** *If* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$*, then:*

- $L = 0 \Rightarrow f(n) = O(g(n))$ *but* $f(n) \neq \Theta(g(n))$ *(f grows slower)*
- $0 < L < \infty \Rightarrow f(n) = \Theta(g(n))$ *(same growth rate)*
- $L = \infty \Rightarrow f(n) = \Omega(g(n))$ *but* $f(n) \neq O(g(n))$ *(f grows faster)*

Definition 11.2 (Little-o and little-omega)**. Little-o:** $f(n) = o(g(n))$ means $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. Strictly slower growth.

Little-omega: $f(n) = \omega(g(n))$ means $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$. Strictly faster growth.

Theorem 11.2 (Properties of asymptotic notation)**.**

1. ***Transitivity:*** *If* $f = O(g)$ *and* $g = O(h)$*, then* $f = O(h)$*.*
2. ***Reflexivity:*** $f = O(f)$*,* $f = \Omega(f)$*,* $f = \Theta(f)$*.*
3. ***Symmetry:*** $f = \Theta(g)$ *iff* $g = \Theta(f)$*.*
4. ***Transpose symmetry:*** $f = O(g)$ *iff* $g = \Omega(f)$*.*
5. ***Sum rule:*** $O(f) + O(g) = O(\max(f, g))$*.*
6. ***Product rule:*** $O(f) \cdot O(g) = O(f \cdot g)$*.*
7. ***Constant factors:*** $O(cf) = O(f)$ *for any constant* $c > 0$*.*

Common complexity classes

Definition 11.3 (Growth rate hierarchy)**.** From slowest to fastest:

$$O(1) \subset O(\log n) \subset O(\sqrt{n}) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^3) \subset O(2^n) \subset O(n!)$$

Notation	Name	Example
$O(1)$	Constant	Array access
$O(\log n)$	Logarithmic	Binary search
$O(n)$	Linear	Linear search
$O(n \log n)$	Linearithmic	Merge sort
$O(n^2)$	Quadratic	Bubble sort
$O(n^3)$	Cubic	Naive matrix multiplication
$O(2^n)$	Exponential	Subset enumeration
$O(n!)$	Factorial	Permutation enumeration

The jump from polynomial (n^k) to exponential (2^n) is critical. Polynomial-time algorithms are "feasible"; exponential-time algorithms quickly become impractical.

Analyzing code

Theorem 11.3 (Loop analysis)**.**

- *A loop running* n *times with* $O(1)$ *body:* $O(n)$
- *Two nested loops, each running* n *times:* $O(n^2)$
- *Three nested loops:* $O(n^3)$
- *A loop that halves the problem each iteration:* $O(\log n)$

Proof Strategy

To analyze a loop:

1. Count iterations.
2. Multiply by the cost of one iteration.
3. For nested loops, multiply the iteration counts.

Theorem 11.4 (Summation formulas)**.**

$$\sum_{i=1}^{n} 1 = n$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$$

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1} = \Theta(r^n) \text{ for } r > 1$$

$$\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n) \text{ (harmonic series)}$$

Recurrence relations

Definition 11.4 (Recurrence relation)**.** A **recurrence relation** expresses $T(n)$ in terms of smaller inputs. Common divide-and-conquer form:

$$T(n) = aT(n/b) + f(n)$$

where $a \geq 1$ subproblems of size n/b are solved, and $f(n)$ is the work outside recursion.

Theorem 11.5 (Master theorem)**.** *For $T(n) = aT(n/b) + f(n)$ with $a \geq 1$, $b > 1$:*
Let $c = \log_b a$. Compare $f(n)$ with n^c:
Case 1: *If $f(n) = O(n^{c-\epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^c)$. (Recursion dominates.)*
Case 2: *If $f(n) = \Theta(n^c \log^k n)$ for some $k \geq 0$, then $T(n) = \Theta(n^c \log^{k+1} n)$. (Balanced.)*
Case 3: *If $f(n) = \Omega(n^{c+\epsilon})$ for some $\epsilon > 0$ and $af(n/b) \leq kf(n)$ for some $k < 1$, then $T(n) = \Theta(f(n))$. (Work dominates.)*

Theorem 11.6 (Common recurrences)**.**

Recurrence	***Solution***	***Example***
$T(n) = T(n/2) + O(1)$	$O(\log n)$	*Binary search*
$T(n) = T(n-1) + O(1)$	$O(n)$	*Linear recursion*
$T(n) = T(n-1) + O(n)$	$O(n^2)$	*Selection sort*
$T(n) = 2T(n/2) + O(1)$	$O(n)$	*Tree traversal*
$T(n) = 2T(n/2) + O(n)$	$O(n \log n)$	*Merge sort*
$T(n) = 2T(n-1) + O(1)$	$O(2^n)$	*Naive Fibonacci*

Solving recurrences

Proof Strategy

Method 1: Expansion (iteration)

1. Expand the recurrence several times.
2. Identify the pattern.
3. Sum the terms.

Method 2: Substitution (guess and verify)

1. Guess the form of the solution.
2. Use induction to verify.
3. Adjust constants as needed.

Method 3: Master theorem

1. Identify a, b, and $f(n)$.
2. Compute $c = \log_b a$.
3. Determine which case applies.

Best, worst, and average case

Definition 11.5 (Case analysis)**.**

- **Worst case:** Maximum time over all inputs of size n.
- **Best case:** Minimum time over all inputs of size n.
- **Average case:** Expected time over a probability distribution on inputs.

Usually, we report worst-case complexity—it provides guarantees.

Complexity classes and reductions

Definition 11.6 (P and NP)**.**

- **P**: Problems solvable in polynomial time by a deterministic algorithm.
- **NP**: Problems whose solutions can be *verified* in polynomial time. Equivalently, solvable in polynomial time by a nondeterministic algorithm.

$\mathbf{P} \subseteq \mathbf{NP}$ (anything solvable quickly is certainly verifiable quickly). Whether $\mathbf{P} = \mathbf{NP}$ is the most famous open problem in computer science.

Definition 11.7 (Reductions and NP-completeness)**.** A polynomial-time reduction from A to B (written $A \leq_p B$) transforms instances of A into instances of B preserving yes/no answers.

- B is **NP-hard** if every problem in NP reduces to B.
- B is **NP-complete** if $B \in \mathbf{NP}$ and B is NP-hard.

If any NP-complete problem is in **P**, then **P** = **NP**. The prevailing belief is that this doesn't happen.

Proof Strategy

To prove a problem B is NP-complete:

1. Show $B \in \mathbf{NP}$ (solutions are verifiable in polynomial time).
2. Reduce a known NP-complete problem A to B in polynomial time.

Example 11.1 (Common NP-complete problems)**.** SAT, 3-SAT, CLIQUE, VERTEX COVER, HAMILTONIAN CYCLE, SUBSET SUM.

Worked examples

Example 11.2 (Proving big-Theta)**.** Show that $3n^2 + 5n + 7 = \Theta(n^2)$.

Solution.

Upper bound: For $n \geq 1$:

$$3n^2 + 5n + 7 \leq 3n^2 + 5n^2 + 7n^2 = 15n^2$$

So $3n^2 + 5n + 7 = O(n^2)$ with $c = 15$, $n_0 = 1$.

Lower bound: For $n \geq 1$:

$$3n^2 + 5n + 7 \geq 3n^2$$

So $3n^2 + 5n + 7 = \Omega(n^2)$ with $c = 3$, $n_0 = 1$.

Therefore $3n^2 + 5n + 7 = \Theta(n^2)$.

Example 11.3 (Ordering by growth rate)**.** Order: $n \log n$, $n^{1.5}$, 2^n, n^3.

Solution. Use limits:

- $\lim \frac{n \log n}{n^{1.5}} = \lim \frac{\log n}{\sqrt{n}} = 0$ (L'Hôpital). So $n \log n = o(n^{1.5})$.
- $\lim \frac{n^{1.5}}{n^3} = \lim \frac{1}{n^{1.5}} = 0$. So $n^{1.5} = o(n^3)$.
- $\lim \frac{n^3}{2^n} = 0$ (exponential dominates). So $n^3 = o(2^n)$.

Order: $n \log n \prec n^{1.5} \prec n^3 \prec 2^n$.

Example 11.4 (Nested loops, both to n)**.** Analyze:

```
for i = 1 to n:
    for j = 1 to n:
        // O(1) operation
```

Solution. Inner loop runs n times per outer iteration. Outer runs n times. Total: $n \times n = n^2$. Complexity: $O(n^2)$.

Example 11.5 (Nested loops, triangular)**.** Analyze:

```
for i = 1 to n:
    for j = 1 to i:
        // O(1) operation
```

Solution. Inner loop runs i times. Total iterations:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$

Example 11.6 (Master theorem)**.** Solve $T(n) = 2T(n/2) + n$ with $T(1) = 1$.

Solution. Identify: $a = 2$, $b = 2$, $f(n) = n$.
$c = \log_b a = \log_2 2 = 1$, so $n^c = n$.
Compare: $f(n) = n = \Theta(n^1) = \Theta(n^c \log^0 n)$.
This is Case 2 with $k = 0$: $T(n) = \Theta(n^c \log^{k+1} n) = \Theta(n \log n)$.

Example 11.7 (Expansion method)**.** Solve $T(n) = 2T(n/2) + n$ by expansion.

Solution.

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2[2T(n/4) + n/2] + n = 4T(n/4) + 2n \\ &= 4[2T(n/8) + n/4] + 2n = 8T(n/8) + 3n \\ &\vdots \\ &= 2^k T(n/2^k) + kn \end{aligned}$$

When $n/2^k = 1$, we have $k = \log_2 n$ and $T(1) = 1$:

$$T(n) = 2^{\log n} \cdot 1 + n \log n = n + n \log n = \Theta(n \log n)$$

Example 11.8 (Binary search)**.** Analyze binary search.

Solution. Each step halves the search space. Recurrence:

$$T(n) = T(n/2) + O(1), \quad T(1) = O(1)$$

By master theorem: $a = 1$, $b = 2$, $f(n) = O(1)$. $c = \log_2 1 = 0$, so $n^c = 1$.
$f(n) = \Theta(n^0)$. Case 2 with $k = 0$: $T(n) = \Theta(\log n)$.

Example 11.9 (Log factorial)**.** Show that $\log(n!) = \Theta(n \log n)$.

Solution.
Upper bound: $n! \le n^n$, so $\log(n!) \le n \log n$.
Lower bound: $n! \ge (n/2)^{n/2}$ (considering only the top half of terms), so:

$$\log(n!) \ge \frac{n}{2} \log \frac{n}{2} = \frac{n}{2}(\log n - 1) = \Omega(n \log n)$$

Therefore $\log(n!) = \Theta(n \log n)$.

Example 11.10 (Log grows slower than any polynomial)**.** Prove $\log n = O(n^\epsilon)$ for any $\epsilon > 0$.

Proof. Compute the limit:

$$\lim_{n \to \infty} \frac{\log n}{n^\epsilon}$$

This is ∞/∞. Apply L'Hôpital:

$$\lim_{n \to \infty} \frac{1/n}{\epsilon n^{\epsilon - 1}} = \lim_{n \to \infty} \frac{1}{\epsilon n^\epsilon} = 0$$

Since the limit is 0, $\log n = O(n^\epsilon)$. In fact, $\log n = o(n^\epsilon)$: log grows strictly slower than any positive power.

Example 11.11 (Constant factors in exponents matter)**.** Show $2^{n+1} = \Theta(2^n)$ but $2^{2n} \neq O(2^n)$.

Solution.

Part 1: $2^{n+1} = 2 \cdot 2^n$. So $2^{n+1} = \Theta(2^n)$ with constant 2.

Part 2: $2^{2n} = (2^2)^n = 4^n$.

$$\lim_{n\to\infty} \frac{4^n}{2^n} = \lim_{n\to\infty} 2^n = \infty$$

So 4^n grows faster than 2^n, meaning $2^{2n} \neq O(2^n)$.

Key insight: Adding a constant to the exponent is fine; multiplying the exponent is not.

Example 11.12 (Master theorem, Case 1)**.** Solve $T(n) = 3T(n/2) + n$.

Solution. $a = 3$, $b = 2$, $f(n) = n$.

$c = \log_2 3 \approx 1.585$.

Compare $f(n) = n = n^1$ with $n^c = n^{1.585}$. Since $1 < 1.585$, we have $f(n) = O(n^{c-\epsilon})$ for $\epsilon = 0.585$.

Case 1: $T(n) = \Theta(n^c) = \Theta(n^{\log_2 3})$.

Example 11.13 (Linear recursion)**.** Solve $T(n) = T(n-1) + n$ by expansion.

Solution.

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &= T(n-3) + (n-2) + (n-1) + n \\ &\vdots \\ &= T(1) + 2 + 3 + \cdots + n \\ &= T(1) + \frac{n(n+1)}{2} - 1 \end{aligned}$$

If $T(1) = 1$: $T(n) = \frac{n(n+1)}{2} = \Theta(n^2)$.

Common Mistake

Treating big-O as equality. $f = O(g)$ means f is bounded above by g, not equal. "f is $O(g)$" is more accurate than "f equals $O(g)$."

Common Mistake

Confusing worst-case and big-O. Big-O describes function growth. Worst-case describes inputs. They're related but distinct: worst-case running time *is* a function that can be described with big-O.

Common Mistake

Forgetting the regularity condition in Case 3. The master theorem's Case 3 requires $af(n/b) \leq kf(n)$ for some $k < 1$. Usually satisfied, but verify.

Going Deeper: Monoids—Categories with One Object

Throughout this course, we've seen how categories unify mathematics. We end with a beautiful observation: *monoids are categories with exactly one object*. For more detail, see the Category Theory Companion, Week 10.

Monoids: The Definition

A **monoid** $(M, \cdot, e)$ is a set M with:

- A binary operation $\cdot : M \times M \to M$
- An identity element $e \in M$

satisfying:

- **Associativity:** $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- **Identity:** $e \cdot a = a = a \cdot e$

Compare to category axioms:

Monoid	**Category**
Elements of M	Morphisms
Multiplication $\cdot$	Composition $\circ$
Identity e	Identity morphism id
Associativity	Associativity

If a category has only one object $\star$, all morphisms go from $\star$ to $\star$—they can always be composed. The morphisms form a monoid.

Examples of Monoids

1. $(\mathbb{N}, +, 0)$: natural numbers under addition.
2. $(\mathbb{N}, \times, 1)$: natural numbers under multiplication.
3. $(\Sigma^*, \cdot, \varepsilon)$: strings under concatenation (the *free monoid* on Σ).
4. $(A \to A, \circ, \mathrm{id}_A)$: functions from A to itself under composition.

The Free Monoid

Given alphabet Σ, the **free monoid** Σ^* is all strings with concatenation. It's "free" because it satisfies no equations beyond the monoid axioms.
Universal property: For any monoid M and function $f : \Sigma \to M$, there's a unique monoid homomorphism $\bar{f} : \Sigma^* \to M$ extending f.
This is why Σ^* appears in automata theory: a DFA computes a monoid homomorphism from Σ^* to a finite monoid.

Monoid Homomorphisms

A **monoid homomorphism** $\phi : M \to N$ satisfies:

- $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$
- $\phi(e_M) = e_N$

In categorical terms: a homomorphism between one-object categories is exactly a functor.
Example: String length $\text{len} : \Sigma^* \to \mathbb{N}$ is a homomorphism: $\text{len}(s \cdot t) = \text{len}(s) + \text{len}(t)$ and $\text{len}(\varepsilon) = 0$.

Connection to Complexity Analysis

Consider the monoid $(\mathbb{N}, +, 0)$ of running times. When we analyze:

$$T(n) = T(n/2) + O(1)$$

we're adding times (monoid operation) of recursive work plus local work.
For divide-and-conquer with a subproblems:

$$T(n) = \underbrace{T(n/b) + T(n/b) + \cdots + T(n/b)}_{a \text{ terms}} + f(n)$$

The structure of recurrences reflects monoid structure.

The Categorical Perspective: Full Circle

We began with sets and functions, introduced diagrams, and discovered that many structures—preorders, graphs, automata—are categorical. Now we see that even monoids are secretly categories.
This suggests a powerful principle: *category theory reveals common structure across diverse areas.* Types, proofs, and programs all form categories. The tools we've developed—diagrams, universal properties, functors—are the beginning of a vocabulary for understanding computation, logic, and mathematics as aspects of a unified whole.

Exercises: Monoids and Structure

1. Verify that $(\mathbb{Z}, +, 0)$ is a monoid. Is $(\mathbb{Z}, -, 0)$ a monoid? Why or why not?
2. The set $\{0, 1\}$ with OR ($\vee$) and identity 0 forms a monoid. Write its multiplication table.
3. Show that $n \times n$ matrices under multiplication with identity I form a monoid.
4. Is string length a monoid homomorphism from $(\Sigma^*, \cdot, \varepsilon)$ to $(\mathbb{N}, +, 0)$? Prove it.
5. Prove: If $\phi : M \to N$ is a monoid homomorphism, then $\phi(M)$ is a submonoid of N.
6. Consider $(\mathbb{Z}_n, +_n, 0)$ where $+_n$ is addition mod n. Prove $\phi(k) = k \mod n$ is a homomorphism from $\mathbb{Z}$.

7. **Challenge:** $\mathcal{P}(A)$ forms a monoid under union (identity $\emptyset$) and under intersection (identity A). Are these monoids isomorphic?

8. Define big-O equivalence: $f \sim g$ iff $f = \Theta(g)$. Show the equivalence classes form a monoid under multiplication.

Practice

1. Order: $n \log n$, $n^{1.5}$, 2^n, n^3 by growth rate.
2. Show that $3n^2 + 5n + 7$ is $\Theta(n^2)$.
3. Solve $T(n) = 2T(n/2) + n$ with $T(1) = 1$.
4. Analyze the runtime of binary search.
5. Prove: $\log n = O(n^\epsilon)$ for any $\epsilon > 0$.
6. Analyze:

```
for i = 1 to n:
    for j = i to n:
        for k = 1 to j:
            // O(1)
```

7. Solve $T(n) = 3T(n/2) + n$ using the master theorem.
8. Prove: $(n+1)! = O((n!)^2)$ but $(n+1)! \neq \Theta(n!)$.
9. Analyze $T(n) = T(n-1) + n$ with $T(1) = 1$.
10. Show $2^{n+1} = \Theta(2^n)$ but $2^{2n} \neq \Theta(2^n)$.
11. Prove: $f(n) = o(g(n))$ implies $f(n) = O(g(n))$.
12. A recursive algorithm satisfies $T(n) = T(n/3) + T(2n/3) + n$. Prove $T(n) = O(n \log n)$.
13. If $A \leq_p B$ and $B \in \mathbf{P}$, what can you conclude about A?
14. Explain why HAMILTONIAN CYCLE is in **NP**.
15. Outline a reduction from CLIQUE to VERTEX COVER.