COMPSCI 70 Self-Study Lecture Notes

Dun-Ming Huang

Contents

Chapter 0

Review of Sets and Mathematical Notation

In this chapter, we provide a brief review of fundamental mathematical notations that will be used throughout the coursework.
This involves set notation, set properties, as well as convenient mathematical notations that aid to simplify, abbreviate, as well as clarify the different conditions and clauses that we used to literally write out in mathematical statements.

0.1 Set Theory

This chapter describes the fundamental properties of mathematical objects called Sets. Sets are widely applied throughout and beyond the subfields of mathematics, whose origins and characteristics shared across those applications are described by the mathematical definition of sets.

0.1.1 Fundamental Properties of Sets

As you may have guessed, in set theory, we mainly discuss a mathematical object called "set".

Definition 0.1.1. Set

A set is a collection of objects.
The mathematical objects that are members of a set are called elements.
Sets are normally expressed in the format of:

$$\{\text{element 1}, \text{element 2}, \ldots, \text{element n}\}$$

The symbol representing membership in a set works as follows:

Symbol 0.1.1. Membership Symbol

If a mathematical object x is a member of a set A, then we may mathematically express that $x \in A$.
Otherwise, for a mathematical object y that does not belong to A, $y \notin A$.

There are several more properties to sets:

- The size of a set is defined as the number of elements in it. This property of set is called **cardinality**. The cardinality of a set A can be mathematically denoted as $|A|$.
- The equality of a set is held when two sets have exactly same elements; order and repetition does not matter, albeit in computers, many implementations of sets do not allow repetition.
- Sets whose cardinality are 0, or in other words empty sets, are denoted as as well as $\emptyset$.

To denote a set of mathematical objects that all belong to some other set but suffices some property, we may utilize a notation called **set-builder notation**:

Symbol 0.1.2. Set-Builder Notation

A set A whose members all follow expression exp, and the terms of exp all fit some list of conditions can be written as:

$$\{exp : \text{condition 1}, \ldots, \text{condition n}\}$$

The colon above can be replaced by a vertical bar. Since we will be using vertical bars for other purposes in later chapters, a colon is perhaps the least abusive notation to play with.
But, to conform the formatting of official lecture notes, let us continue with vertical bars.
For example, the list of all rational numbers can be written as:

$$\mathbb{Q} = \left\{ \frac{p}{q} | p, q \in \mathbb{R}, q \neq 0 \right\}$$

0.1.2 Relationships between Sets

We often compare sets in terms of their sizes and elements they contain. Out of these standards for comparison, we can define the relationship between a set and another larger set that contains all elements of the former as follows:

Definition 0.1.2. Subset

If every element of a set A is also in a set B, then A is a subset of B.
Mathematically, we write it as $A \subseteq B$.
Or, stating the equivalent in an opposite direction, $B \supseteq A$, and this would state B as a superset of A.

And a stricter similar relationship follows:

Definition 0.1.3. Strict Subset

If $A \subseteq B$ but A excludes at least an element of B, then we say that A is a strict subset (proper subset) of B.
Mathematically denoted, $A \subset B$.

Utilizing the definitions above, we may form some observations:

- The empty set is a proper subset of any nonempty set.
- The empty set is a subset for any set, including itself's.
- While every set is not a proper subset of itself, every set is a subset of itself.

While we compare sets based on their members, we can also "add", create sets based on the members of two sets. There are two ways of doing so, being **intersection** and **union**.

Definition 0.1.4. Intersection

The intersection of set A and B, written as $A \cap B$, is the set containing all elements that are both in A and B.
In set builder notation:

$$A \cap B = \{x | x \in A \wedge x \in B\}$$

Definition 0.1.5. Union

The union of set A and B, written as $A \cup B$, is the set of all elements contained in either A or B.
In set builder notation, it pronounces very similarly with the notation for intersections:

$$A \cap B = \{x | x \in A \vee x \in B\}$$

Regarding the way empty sets work in the above arithmetic, for an arbitrary set A:

- $A \cap \emptyset = \emptyset$
- $A \cup \emptyset = A$

At last, let us regard the notion of **"complements", the difference between sets**.

Definition 0.1.6. Complement

Imagine the arithmetic of sets to work solely upon their members, and let the difference of sets be defined such that:

$$A - B = A \backslash B = \{x \in A | x \notin B\}$$

This set A B is called the set difference between A and B, or alternatively, the relative complement of B in A.

This also indicates that the "subtraction arithmetic" for sets is not commutative. Using empty sets as examples:

- $A \setminus \emptyset = A$
- $\emptyset \setminus A = \emptyset$

And the last arithmetic is the **"multiplication of sets": Cartesian Products**.

Definition 0.1.7. Cartesian Products

The Cartesian Product (cross product) of sets A and B is defined such that:

$$A \times B = \{(a,b) \mid a \in \mathbb{R}, b \in \mathbb{R}\}$$

And last but not least, we have mathematical operations that generate a set of sets (nested) based on other sets:

Definition 0.1.8. Power Set

The power set of S can be written as:

$$\mathbb{P}(S) = \{P | P \subseteq S\}$$

as one of its various denotations, the one on lecture note using a Weierstrass P.
The power set of a set is the set of all of its possible subsets.

In addition, if the cardinality of S in the above definition is $|S| = k$, then we can state that $|\mathbb{P}(S)| = 2^k$.

0.2 Mathematical Notation

Here we describe and explore mathematical notations that will abbreviate equations, statements, as well as facilitate us to view mathematical expressions in different perspectives. Mathematical notations introduced in this section will be very frequently employed in coming courseworks.

0.2.1 Summation and Products

The summation of some expression dependent on some other variable can be stated with a summation symbol, such that:

Symbol 0.2.1. Summation

Let $f(x)$ be an expression based on some input x, then the following expressions are equivalent:

$$f(m)+f(m+1)+\cdots+f(m+n)=\sum_{k=1}^{n} f(m+k)$$

While the summation expression is highly applicable to numerous theorems, there is another notation similar in nature to the summation expression– the product expression:

Symbol 0.2.2. Product

Let $f(x)$ be an expression based on some input x, then the following expressions are equivalent:

$$f(m)\times f(m+1)\times\cdots\times f(m+n)=\prod_{k=1}^{n} f(m+k)$$

0.2.2 Quantifiers

Quantifiers are symbols that help us target a range of elements from some set when we need to write a conditional statement for it. There are two specific quantifiers we will use here.

Symbol 0.2.3. Universal Quantifier

The universal quantifier $\forall$, pronounced as "for all", targets all elements from a set.
For example, the phrase "$\forall x \in A$" is equivalent to the phrase "for all objects x that belongs to the set A". This can serve as the first clause of a statement, while the second clause would refer a statement for the elements targeted by the quantifier.

Meanwhile, we have a different quantifier that selects a different range from the universal quantifier:

Symbol 0.2.4. Existential Quantifier

The existential quantifier $\exists$, pronounced as "there exists (at least one)", states that there should exist at least one element from some set that fits a condition stated in the second clause. The syntax, therefore, is as follows:

$$\exists x \in A : \text{statement}$$

This means: there exists at least an object x belonging to the set A such that the "statement" holds. There are some more variants to this overall syntax, such as replacing $\in$ with $\notin$... etc.

There also exists some variations to the existential quantifier:

- There exists at least one: $\exists$
- There exists only one: $\exists!$
- There does not exist: $\nexists$

0.3 Personal Regard on this Topic

Personal log for mathematical topics, because why not! Just don't treat this section as educational content.
Sets are very useful objects for mathematical generalizations, which we'd do a TON in discrete mathematics, hence its practicality. There will, however, come times where statements are hard to write out.
In that case, don't bother with making the statements multi-part! There's a clear trend in homework questions and solutions for that.

Also, my reader didn't grade one of my homework questions and I missed my regrade deadline :v

Chapter 1

Propositional Logic

This chapter introduces the fundamentals of propositional logic while enhancing the student's understanding of quantifiers, as well as discussing popular uses of propositional operations as implications and negations.

1.1 Propositional Logic

1.1.1 Definition of Proposition

To speak the language of mathematics, we must familiarize ourselves with objects of the mathematical language. One fundamental block of this mathematical language is a **proposition**.

Definition 1.1.1. Proposition

A proposition is a logical statement that is either true or false.
In other words, it is a mathematical statement that could be correct, but also could be incorrect.

Propositions need to be either true or false, which means it cannot be statements that can yield some ambiguous result. And because they usually are not known to be true or false, we often need to prove propositions in discrete mathematics.

Think Brandon! Think! 1.1.1: What should a proposition be like?

Which of the following qualifies as a proposition?

(a) $\sqrt{3}$ is rational.

(b) $2 + 2$

(c) I often never give up or let down

Option (a) is a statement that has to either hold true or false, because this value $\sqrt{3}$ is either rational or irrational.
Option (b) is a mathematical expression, and there is no true or false aspect to it. It is not a proposition.
Last but not least, option (c) is just a statement, because the word "often" is not properly defined. If propositions are supposed to clearly hold true or false, it should not contain blurrily defined words.
The answer, therefore, is option (a).

1.1.2 Logical Operations of Proposition

There are approximately four methods of using preexistent propositions to make new propositions, and we do so to produce complicated propositions from simpler subparts.
These four ways being:

- **Conjunction**: $P \wedge Q$, works similarly as the boolean "and".
- **Disjunction**: $P \vee Q$, works similarly as the boolean "or".
- **Negation**: $\neg P$, works similarly as the boolean "not".
- **Implies**: $P \implies Q$, which states a familiar phrase of "if P, then Q". Here, the proposition P is called a hypothesis, and Q a conclusion.

To demonstrate the behaviors of these operations, we may use a **truth table**. A truth table example for the operation "Conjunction" is attached below:

Figure 1.1.1. Truth Table for Conjunction

A truth table is a table that points out the result of an operation on statements P and Q, for a set of given boolean values that P and Q each result in.

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

Now we may discuss some other properties of propositions.
First of all, there is a fundamental principle called **Law of the Excluded Middle**, which states what is as follows:

Axiom 1.1.1. Law of Excluded Middle

For any proposition P, $P \vee \neg P$ always holds true, because P is either true or false (not true).

Here, propositions like $P \vee \neg P$ that always hold true are called **tautologies**.
Meanwhile, propositions that are always false, such as $P \wedge \neg P$ (which cannot be true, since P is either true or false), are called **contradictions**.
Implications also have interesting truth table properties:

Figure 1.1.2. Truth Table for Implication

A truth table for implications is as follows:

P	Q	$P \implies Q$	$\neg P \vee Q$
T	T	T	T
T	F	F	F
F	T	T	T
F	F	T	T

While this provides that the implication is only false when P is true and Q is false, this also provides a not so intuitive insight: An implication is trivially true when the hypothesis is false.
This situation is known as **"vacuously true"**.
The reason why this insight seems to hold is because while we cannot confirm the conclusion is false when the hypothesis is false, this is also the nature of how if-then statements are expected to be defined.

Indicated via the above truth table, as $P \implies Q$ and $P \vee Q$ have the exact same boolean behavior on truth table, they are logically equivalent. This means they are statements that perform the logical behavior and meaning.

Interestingly, there are also various ways to pronounce implications, such as "if P, then Q", as well as "Q if P", and many more.
Sometimes we also come across words that seem to connect two propositions in a tighter implication, such as "iff".
An **iff** relationship is also standardly pronounced as **if and only if**, which is mathematically expressed as $P \iff Q$.
This "iff" holds if both $P \implies Q$ and $Q \implies P$ are true. Consequentially, and as encountered in previous coursework, such as EECS 16A, the proof of an "iff" statement is at its nature the proof of two smaller statements.

1.1.3 Implications from Implications

An implication can have logically equivalent statements that are based on it. This means that for an implication A and a logically equivalent implication B based on A, if A is true, then so is B expected to.
There can also be implications that are based on A yet doesn't necessarily hold true when A does. After all, implications are, in nature, propositions.

For an implication $P \implies Q$, we may define two other implications out of it:

- **Contrapositive**: $\neg Q \implies \neg P$, which is logically equivalent to $P \implies Q$.
- **Converse**: $Q \implies P$, which does not always hold true. In some cases, converse statements can hold true, but usually accompanies some extra conditions. Some examples can be observed in the contents of MATH 53.

When two propositions are logically equivalent, we may mathematically denote it with the $\equiv$ symbol. An example of usage would be stating an implication the logical equivalent of its contrapositive, in the forms:

$$(P \implies Q) \equiv (\neg Q \implies \neg P)$$

An application of this is a new approach of performing proofs. To prove a prompt that states a theorem as an implication $P \implies Q$, by proving its contrapositive, which is in some cases easier than proving the prompt directly, we can manage to indirectly prove the prompt.
This application will be introduced in the next chapter as "proof by contrapositive".

1.2 Quantifiers

1.2.1 Use of Quantifiers

In the previous chapter, we introduced the usage of quantifiers as an abbreviation and selector of elements in sets to which a condition holds, in the format of:

<quantifier selecting clause> <second clause containing condition to hold for selected values>

The second clause is effectively a proposition, which the entire sentence containing the first and second clause is also effectively a proposition.

There is a more sophisticated way to express the first clauses of these forms of propositions.
The first clause has a quantifier, and throughout a "universe" we are working with (abstractly, a wide variety of mathematical objects following some conventions), the statement is quantified (holding true) for a selected range of objects in this universe.
So syntactically, the quantifier serves to involve this range of quantification for our proposition.

In a finite universe, there is also another way of conceptualizing the quantifiers we know:

Think Brandon! Think! 1.2.1: Propositional Logic in Quantifiers

Let us work with a universe $\mathbb{U} = U_1, \ldots, U_n$, where n is some finite natural number and elements of this universe are some mathematical object.

Then:

- **Universal Quantifiers**: $(\forall x \in \mathbb{U}(P(x))) \equiv (P(U_1) \wedge \cdots \wedge P(U_n))$
- **Existential Quantifiers**: $(\exists x \in \mathbb{U}(P(x))) \equiv (P(U_1) \vee \cdots \vee P(U_n))$

To express more complicated propositions, we must expand beyond the previous first-clause-second-clause format for propositions. This is especially when we are selecting two ranges of values to describe a multi-variable proposition with.

Think Brandon! Think! 1.2.2: How to read complicated propositions?

Say we are dealing with the proposition:

$$(\forall x \in \mathbb{Z})(\exists y \in Z)(x < y)$$

Here, I can observe two quantifying clauses and the last clause that stands for a proposition. In fact, propositions involving quantifiers will always have the last clause as a proposition, due to the syntactical limits and customs of mathematical language.
Now let us try translating each clause into the English language:

1. $\forall x \in \mathbb{Z}$: For all objects x that belongs to $\mathbb{Z}$, the set of all integers.
2. $\exists y \in \mathbb{Z}$: There exists an object y that belongs to $\mathbb{Z}$, the set of all integers.
3. $x < y$: Object x is smaller than y.

Piecing the clauses together: For all objects x that belongs to $\mathbb{Z}$, the set of all integers, there exists an object y that belongs to $\mathbb{Z}$ such that x is smaller than y.
This proposition, in fact, holds true, since we can always find a larger integer.
However, the proposition with a reversed order for quantifier clauses:

$$(\exists y \in Z)(\forall x \in \mathbb{Z})(x < y)$$

has to hold False.

With the above translating process, this statement states that: there is an object y that belongs to $\mathbb{Z}$, the set of all integers, such that for all objects x that belongs to $\mathbb{Z}$ such that x is smaller than y.
In other words, it assumes the existence of one largest integer, which does not exist!

1.3 Negation

1.3.1 Meaning of Negation

If a proposition P is false, its negation is true.
This inspires another approach for proofs (proof by contradiction), which usually work for simpler prompts since their contradictions would only be simpler to prove, if provided that this method is optimal to begin with.
However, when P appears to be a more complicated proposition involving other smaller propositions, we should look for some way to make our understanding of it more concise.
A logical law, called the **De Morgan's Laws**, facilitates us with such matter:

Axiom 1.3.1. De Morgan's Law

The De Morgan's Law states the two following logical equivalences:

$$\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$$

$$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$$

In fact, negations of propositions involving quantifiers follow analogous laws, due to the similarity of universal quantifier with conjunction and existential quantifier with disjunction:

Axiom 1.3.2. Extension of De Morgan's Law

Based on the similiarity of quantifiers with operations of propositions:

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

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

These above equivalences can provide flexibility in coming mathematical proofs.

1.3.2 Complex Examples of Negation

Let us discuss the example attached on the lecture notes from Summer 2022.

Prompt 1: Write a proposition that states, "there are at least three distinct integers x that satisfies $P(x)$.

And the proposition our textbook provided was:

$$\exists x \exists y \exists z (x \neq y \neq z \wedge P(x) \wedge P(y) \wedge P(z))$$

Think Brandon! Think! 1.3.1: Proposition for Prompt 1

Let us analyze this proposition via some translation:

$\exists x \exists y \exists z (x \neq y \wedge x \neq z \wedge y \neq z \wedge P(x) \wedge P(y) \wedge P(z))$
$\rightarrow$ "there exists a x, y, z" such that
"x, y, z are not equal to each other, and all of P(x), P(y), and P(z) holds".
$\rightarrow$ "there exists at least one possible combination of three distinct integers"
such that "all of P(x), P(y), and P(z) holds".
$\rightarrow$ there are at least three distinct integers x that satisfies $P(x)$.

In the above translation process, we first analyzed each phrases independently. Then, having each clauses translated, we move on to combine them together semantically.
For example, the fact that the three integers x, y, z are not equal to each other means they are distinct, and so we may summarize the quantifying clause from "there exists a ..." into "there exists at least one combination of three integers such that ...".
This helps us imply that, if there exists at least one combination, then there are at least three integers to satisfy $P(x)$.
This logic of inter-language and inter-context translation allows us to convert a proposition from its mathematical form to pure English text. It should be familiarized via practice and experience.
For purposes of practicing, let us analyze another proposition and attempt to translate it:

$$\exists x \exists y \exists z \forall d (P(d) \implies d = x \vee d = y \vee d = z)$$

And in the following portion, let us perform again the same procedure of directly translating the proposition from symbols to English, and then rephrase each English clause into more concise descriptions.

Think Brandon! Think! 1.3.2: Explain this proposition for me, will you?

Translate the proposition to English: $\exists x \exists y \exists z \forall d(P(d) \implies d = x \vee d = y \vee d = z)$

This proposition first has a quantifying clause that states: "there exists a x, y, and z for all values d", or that there exists a set of three numbers x, y and z for all values d.
The last clause is an implication stating: "if $P(d)$ holds true, then $d = x \vee d = y \vee d = z$". That means d is equal to either x, y, or z. So, there exists three numbers out of all possible values in the universe such that if $P(d)$ holds, then d is one of the three numbers we observed before.
Meanwhile, we are not provided that x, y, and z are distinct values, so there are at most three values for d such that $P(d)$ holds.
Let us switch a perspective and view the contrapositive of the later clause and verify our previous interpretation.
If d is equal to neither of x, y, z, then $P(d)$ will not hold.
Assuming they are distinct, this statement would provide that if d is equal to neither of some (up to three) distinct numbers, $P(d)$ will not hold.
Therefore, the proposition is stating "**P(d) holds for at most three distinct integers**".

Here is one demonstration about conjunctions.
By performing conjunction for the propositions above: that $P(d)$ holds for at least and at most 3 integers, we successfully create a new proposition that states "$P(d)$ holds for exactly 3 integers".
Mathematical propositions that come with limits, or quantifying clauses, for some other proposition are useful in helping us locate a range of values for which a condition holds.

If we combine propositions that hold for same conditions but with different quantified limits, we managed to combine the limits together and form some tighter statement providing a proposition with tighter limitations.

1.4 Personal Regard on This Topic

Propositional logic is one aspect of the universal mathematical language that our community will continue to work, especially in fields as artificial intelligence, where many propositions need to hold for some operation(s) to be optimizied and conducted correctly. So, we will indeed come across complicated propositions.
The most intuitive way to deal with it is practice translating them into English, and figure how to structurally present them the best in our process!

For now, we are still in the “Let’s have a taste of discrete math” stage, and we will have to study just two or three more topics to look at more specific mathematical objects. However, the skills we build here are worthwhile, so make sure to stick around and motivate yourself to read more propositions!
My friend, for instance, tried to familiarize himself with propositional logic by playing tabletop games. That might be something you’re interested in doing during spare time?

Chapter 2

Forms of Proofs

This chapter provides an introduction and demonstration for the lines of thoughts behind numerous examples of proofs. The reader will also be exposed to the variety of proof techniques, and be encouraged to explore them via examples provided in the lecture notes.
In this chapter, we concentrate specifically on mathematical proof for its breadth of application and significant role as the foundational concept and tool of understanding and observing the corusework's contents.
Proofs are known to be more complicated concepts, and as well the first barricade for students when studying discrete mathematics. The creativity and "intuition" for generating proofs usually comes with practice and experience with variety of prompts. Readers are encouraged to explore beyond the examples of this lecture note for that reason.
But before we start, let me preface with this. Proofs have learning curves. My advice for you, sincerely, is to have fun and enjoy tackling challenges.

2.1 Introduction to Proofs

2.1.1 Definition of Proofs

In many occassions, we aim to provide some explanation towards the trueness and falseness of mathematical propositions, statements. The process of doing so is called "proving", and produces a **mathematical proof**.

Definition 2.1.1. Mathematical Proof

A mathematical proof provides a means for guaranteeing the truthfulness of a statement.
Concretely, it is a finite sequence of steps that construct an entire logical deduction, such that this deduction proves the statement for the range of values it is quantified over. In some cases, this means guaranteeing a proposition for infinite cases.

Proofs are widely applied not only on the mathematic fields, but also those of computer science.
For instance, we might want to prove the correctness of a program as in following its contract, or even on the famous "halting problem". In that regard, for the variety of fields the concept of "proof" is applied on, it is not necessarily mathematic. Still, it is completely based on logic.

Proofs that can successfully guarantee the trueness of a statement, or provide conclusive evidence for it, helps us be certain of a proposition. Such proof is welcomedly named a **rigourous** proof.
A rigorous proof has some structural convention. Proofs usually begin with **axiom**, or self-evident propositions, and the proof proceeds with a sequence of simple logical steps from axioms to further statements. In some sense, it is human thinking put on paper without any jumps.

2.1.2 Basic Notations

There are some basic mathematical notations we have not introduced in previous chapters, but will prove useful and foundational in discrete mathematics, including the symbols of some universes that we have discussed before. It will be listed in the following table:

Symbol 2.1.1. Table of Basic Notations in Proofs

$\mathbb{N}$	Set of all natural numbers
$\mathbb{Z}$	Set of all integers
$\mathbb{Q}$	Set of all rational numbers
$\mathbb{R}$	Set of all real numbers
$\mathbb{C}$	Set of all complex numbers
$a\|b$	Number a divides number b ($b = aq$)
$a \coloneqq b$	Definition: number a is defined to have value b

2.2 Proof Technique

2.2.1 Direct Proof

Each techniques of proofs will have to do with guaranteeing some form of implication, say $P(x) \implies Q(x)$. The main approach of a Direct Proof user is to prove the legitimacy of $Q(x)$ via assuming $P(x)$.

We assume an **arbitrary**, another word for "general, any", value x such that $P(x)$ holds. We then write logical deduction that guarantees $Q(x)$ assuming that $P(x)$ is true, therefore proving that if $P(x)$ holds then $Q(x)$ works.

That above is a very literal description of direct proof, and in fact, might have sound like it doesn't offer much about the supposedly esoteric arts of mathematical proofs. Direct Proof is known as the foundational technique of proofs, and most of the time (as encountered in other courseworks), while the approach of direct proof is concise and simply comprehensible, finding the mathematical rules to support this approach is often difficult. It requires creativity.
To demonstrate the Direct Proof Technique, let us consider the following prompt:

Think Brandon! Think! 2.2.1: Example of Direct Proof

Prove that: for any $a,\ b,\ c \in \mathbb{Z}$, if $a|b$ and $a|c$, then $a|(b+c)$

The proposition above is quantified over some three integers a, b and c. The hypothesis is that a divides b and b divides c. Let us write the hypothesis in mathematical notation:

$$b = aq_1, c = aq_2, q_1, q_2 \in \mathbb{Z}$$

Let us observe the value of $(b+c)$, and see how it could guide us to proving the prompt:

$$b + c = aq_1 + aq_2 = a(q_1 + q_2), (q_1 + q_2) \in \mathbb{Z}$$

Then, since $(b+c)$ can be written as a product of integer a with some other integer $q_1 + q_2$, a does divide $(b+c)$. Therefore, $a|(b+c)$.

From the above we can observe that direct proof is a very straightforward approach for proving theorems, especially when the logical process guiding from hypothesis to conclusion is clear. This might also explain why direct proof is a very instictive and popular approach for proofs in the previous courses. From here on, we will utilize direct proof in many other proof techniques as well.
Derivations are often forms of direct proof as well. This means direct proofs can also be algebraically heavier than

seen above. In the above example, we also did not attempt to translate the proposition into mathematical symbols, for the sake of brevity. Therefore, in the example that follows, we will demonstrate a much fuller, complicated example for executing direct proof on a theorem.

Think Brandon! Think! 2.2.2: Show me another example of direct proof

Prove that: For an integer between 0 and 1000 exclusively, if such integer is divisible by 9, then the sum of its digits is also divisible by 9.

Let us organize the proposition into mathematical symbol:

$$(\forall n \in \mathbb{N})(n < 1000) \implies (9|n \implies 9|(\text{sum of digits of n}))$$

And now, let us initiate another logical deduction, assuming that the hypothesis, $9|n$, is true:

$$\begin{aligned}
&9|n \implies n = 9k, k \in \mathbb{Z} \\
&\quad \text{(Let the numeric expression of } n \text{ be } abc\text{, then:)} \\
&\quad \implies 9k = 100a + 10b + c \\
&\quad \implies 9k = 99a + 9b + (a + b + c) \\
&\quad \implies (a + b + c) = 9k - 99a - 9b \\
&\quad \implies (a + b + c) = 9(k - 11a - b) \\
&\quad \text{(Since k, a, b are all integers:)} \\
&\quad \implies (a + b + c) = 9l, l = k - 11a - b \in \mathbb{Z} \implies 9|(a + b + c)
\end{aligned}$$

2.2.2 Proof by Contraposition

This is the second proof technique we will discuss, called **Proof by Contraposition**. We should first recap on what exactly is a contraposition.
The contraposition to an implication $P \implies Q$ is another logically equivalent proposition to the original implication, in the form of $\neg Q \implies \neg P$. Since the implication is logically equivalent to its contrapositive, by proving the contrapositive statement of the prompt, we have indirectly proved the prompt.

Think Brandon! Think! 2.2.3: Demonstrating the Proof by Contraposition Technique

Prove the theorem: *Let n be a positive integer and d|n. If n is odd, then d is odd.*

We will use proof by contraposition to prove this theorem.
Stating the prompt in mathematical symbol:

$$(\forall n \in \mathbb{Z})(d|n) \implies (n \pmod 2 = 1 \implies d \pmod 2 = 1)$$

The contrapositive of this prompt assuming the provided information for n and d would be:

$$(\forall n \in \mathbb{Z})(d|n) \implies (d \pmod 2 \neq 1 \implies n \pmod 2 \neq 1)$$

Provided so, in the contrapositive proposition, our hypothesis is $2|d$. Meanwhile, by prompt, $d|n$, thus for some $k \in \mathbb{Z}$, $n = dk$.

Combining the above knowledge:

$$n = d \times k = 2(\frac{d}{2} \times k)$$
$$2|d \implies \frac{d}{2} \in \mathbb{Z}$$
$$(\frac{d}{2} \times k) \in \mathbb{Z} \implies 2|n$$

In the above proof, we first attempted to find a contrapositive proposition to the original prompt. After so, we directly apply the Direct Proof technique in our current technique.
In other words, proof by contraposition is just finding the contrapositive proposition to the prompt and using direct proof on it: it is a variant of direct proof.

2.2.3 Proof by Contradiction

Proof by Contradiction proves a proposition P by first assuming its negation, $\neg P$. After so, attempt to directly prove that this situation, $\neg P$, can lead to both results of R and $\neg R$. In other words, we prove the implications $\neg P \implies (\neg R \wedge R)$ simultaneously.
This is contradictory.
Furthermore, applying the definition of implications, this provides the insight $\neg P \implies (\neg R \wedge R) \equiv$ False (referring to the Law of Excluded Middle). The contrapositive of such proven proposition is True $\implies P$, proving that proposition P is true.

This process is slightly similar to "disproving": via showing that the negation of the prompt to be proven is a ridiculous, impossible situation to be real.
Let us begin with an example then:

Think Brandon! Think! 2.2.4: Now, an example of Proof by Contraposition Technique

Prove that: There are infinitely many prime numbers.

Now, let us consider the negation of the prompt: there are finitely many prime numbers.
If so, then let us also enumerate these finite amount of (let us say k) prime numbers: $p_1 < p_2 < \cdots < p_k$.
Step 1: For the sake of Contradiction...
Let us then have a number:

$$q \coloneqq p_1 p_2 \dots p_k + 1$$

This number cannot be prime, for the largest prime number is $p_k < q$. Since q is not prime, it then has a prime divisor p that is among the enumerated list of prime numbers.
Step 2: However, if we follow the assumption...
Provided $p|p_1 p_2 \dots p_k$ and supposedly $p|p_1 p_2 \dots p_k + 1$, we reach a proposition that would prove fatal:

$$p|((p_1 p_2 \dots p_k + 1) - (p_1 p_2 \dots p_k))$$

(This comes from another proof in the lecture, stating that $(a|b \wedge a|c \wedge (b > c)) \implies (a|(b-c))$)
The above proposition tells us that $p|1$, and this requires that $p \leq 1$. Since there does not exist such prime divisor p, q has no prime divisor and is therefore prime itself.
We have created a contradiction, stating that if there are finitely many prime numbers, then such defined number q is prime and not prime.
Step 3: We have formed a contradiction:
From this contradiction, we clearly see that there cannot just be finitely many prime numbers.
Rather, as the prompt says, there must be infinitely many prime numbers!

Let us continue the series of demonstration with another example to enhance our understanding towards the process of disproving:

Think Brandon! Think! 2.2.5: Another example for Proof by Contraposition Technique

Prove that: $\sqrt{2}$ is irrational.

To prove by contradiction, we should first present a contradiction in the proposition "$\sqrt{2}$ is rational".
If that above negation of the prompt works true, then the following proposition holds by the definition of rational numbers:

$$\exists p \exists q \left(\sqrt{2} = \frac{p}{q} \wedge p \text{ and } q \text{ are coprime} \right)$$

And therefore, squaring both sides of the later clause for this proposition:

$$2 = \frac{p^2}{q^2}$$

And therefore, $2q^2 = p^2$.
Now, let us enter a smaller proof: "For an integer a, if a^2 is even, so is a even".
Let us consider the contrapositive of that minor prompt, which would be: "For an integer a, if a is odd, so is a^2 odd". Any odd integer, meanwhile, can be considered as some number $a = 2n$ for another integer n. Squaring such odd integer:

$$\begin{aligned} a^2 &= (2n+1)^2 \\ &= 4n^2 + 4n + 1 \\ &= 2(2n^2 + 2n) + 1 \\ (2n^2 + 2n) &\in \mathbb{Z} \\ a^2 \% 2 = 1 &\implies a^2 \text{ is odd} \end{aligned}$$

Having proven this smaller property, let us move back to the original demonstration. The minor proof indicates to us that if p^2 is even, so should p. Therefore, $p = 2c$ for some integer c.
This leads us to $2q^2 = p^2 = 4c^2$, which shows $q^2 = 2c^2$, and provided from there that q^2 is even, so should q.
If p and q are both even, then they are not coprime. Yet, by the definition of irrational number, it is defined that p and q are coprime.
There exists a contradiction formed from the hypothesis "$\sqrt{2}$ is rational": it is wrong.
Via proof by contradiction, we have disproved the negation of prompt and in turn proved $\sqrt{2}$ is irrational.

2.2.4 Proof by Cases

The technique of proof by cases is slightly more complex.
For some claim with an unknown variable, let us set finite amount of cases for that unknown variable, without knowing which of the possible cases is true. Then, prove for each possible cases that for either one to be a legitimate proposition, the general prompt still holds.
Therefore, in turn, for any possibility of the unknown variable in our prompt, the prompt holds.
Let us demonstrate this underlying line of thought with an example below:

Think Brandon! Think! 2.2.6: An example for Proof by Cases

Prove that: $((\exists x \notin \mathbb{Q})(\exists y \notin \mathbb{Q}))(x^y \in \mathbb{Q})$

Let us attempt the proof by cases.
Since the prompt's quantifying is involved with existential quantifiers, showing one set of numbers x and y would suffice to prove the prompt.
Let us use the set $x = y = \sqrt{2}$ to proceed with the prompt. Our cases to prove would then be:

1. **Case 1**: $\sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$
2. **Case 2**: $\sqrt{2}^{\sqrt{2}} \notin \mathbb{Q}$

Case 1
If this case holds, then the prompt is already proved the case's existence.
Case 2
If this case holds, then we may apply a new set: $x = \sqrt{2}^{\sqrt{2}}, y = \sqrt{2}$.
However:

$$\begin{aligned}(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} &= (\sqrt{2})^{\sqrt{2}\times\sqrt{2}} \\ &= (\sqrt{2})^2 = 2 \in \mathbb{Q}\end{aligned}$$

Therefore, we still manage to prove that there exists irrational numbers x, y such that x^y is rational, via proving every possible cases of our prompt's unknown variable letting our prompt guaranteed to be true.

Albeit the introduction said "finitely many cases", we in reality are expected to deal with just 2 or 3 cases for this technique. If we manage to encounter a prompt that requires more cases for so, revise the categorization of cases or try some othe rtechnique of proofs.

Applying the above claim to a broader horizon, every proof has some techniques to prove by, and some are to be more efficient than others. The selection of method is purely context-based.
It is therefore encouraged of mathematicians to explore some different techniques, and encouraged of students to familiarize with each of the four to provide flexibility during problemsolving.

2.2.5 General Advice in Proofs, Bullet Points

- Justification can be stated without proof only if it is correct or will be automatically agreed with by any reader.
- If a step cannot be justified and explained clearly, there exists a jump, and some intermediate steps must be added.
- A subsidiary result useful in a more complex proof is called a lemma. It has to be already proved somewhere else. People also break proofs into proofs of lemmas to provide organization and structure.
- The line between lemma and theorems is not clear cut, but theorems are propositions that usually should be exported to the mathematical world, while lemma exists on a more local standing to just facilitate the proof.
- Still, there are some famous lemmas in the world that serve almost the same roles as theorems. This is why the boundary is said to not be that clear cut.

Side Note from Future: You might find another form of proof, **Proof by Construction**, useful as we proceed into constructions and considerations of larger mathematical objects, such as those that can span several dimensions or levels. However, they are structurally very similar to mathematical inductions, which is the topic of discussion for the next chapter.

2.3 Personal Regard on This Topic

Proofs are difficult, and we all need to practice it. In fact, most of the problems in computer science are resolved or concluded via proofs.
Why are proofs so difficult? Here are some of my own difficulties and my own methods of resolving it:

- Where do I start on a proof? The prompt looks so abstract! It looks like the Minecraft enchantment texts!

  The worst thing you can do to your brain during an exam is making it panick. Minecraft enchantment texts prespective will mystify the prompt, and possibly discourage you from approaching it. At these times, try just listing out every single thing the prompt has provided you. Work with these pieces of puzzles, and paraphrase the prompt mathematically.

- Ho-ho. Are you approaching the proof prompt? Then, what's the next step after writing out all the ideas that trivially exist?

  Think about equivalent statements. Let's take *Think! 2.2.4* as an example. If there are infinitely many prime numbers, then that means there doesn't exist a smallest or largest prime numbers. So the contradiction would be that.

- Now that I have equivalent statements, what do I head for next, if I just have very insufficient information to work with?

  If you don't have anything, try crafting something. In this case, **for the sake of contradiction** saved our lives in 2.2.4 by providing us an imaginary case to work with, where there exists a largest prime number p_k and a number that is some value more than it and must be non-prime. In my view, everything afterwards really just depends on creative work.

- How do I get better at creative work?

  Read more proofs. Read how people approach different problems with different perspectives, internalize homework solutions and online forum proofs, the criticisms on them. You will only broaden your horizon via looking at how other people create proofs (well, at the cost of playing less Smash Bros Ultimate).

- What do you really mean by a perspective?

  In COMPSCI 61B, we have approached a topic called "view", which means an abstraction of a complicated system. For example, the view of a social network might be a graph, the view of a playlist might be a queue. In this case, we can manage and switch between views when encountering proofs with different mathematical objects.
  View, or perspectives (which is my preferred word choice), usually deals greater effort on mathematical object proofs. Number theory related proofs take more insights and approaches that rely on the properties of numbers, such as division results and rationality of numbers.
  And as mentioned before, these efforts can be improved by reading more proofs and learning more about conducting proofs by some individual learning.

In summary, everything starts with making a first step.

Chapter 3

Mathematical Induction

In this chapter, we introduce and discuss a new form of deduction called "Mathematical Induction". In the examples offered by the chapter, we discuss the power and practicality of mathematical induction, as well as the different forms of inductions that come with different tips of using.
Meanwhile, as the chapter does not include the lecture note's description on a connection between COMPSCI 61A's "Leap of Faith" and mathematical induction, the readers are encouraged to nurture an understanding towards this connection in their vision.
Hopefully, it will facilitate the understanding towards mathematical induction and provide a firmer ground on recursive functions/programs.

3.1 Introduction to Mathematical Induction

3.1.1 Definition of Mathematical Induction

Previously, we introduced techniques and directions we can take to produce mathematical proofs. Here, we will introduce a new powerful tool, a new proof technique, a form of deduction called "mathematical indcution". Such form of logic establishes a statement holds for all natural numbers.
This is an important contribution.
First of all, many mathematical objects are closely associated with natural numbers, especially those that we will encounter in COMPSCI 70.
And second of all, when we are asked to prove statements for all natural numbers, there are an infinite number of values to be checked for the statement. It poses the problem: while mathematical proof requires a finite sequence of logic, the prompt requires us to guarantee a proposition for infinite values. This implies how strong a tool mathematical induction can be for mathematicians, and furthermore, computer scientists.

How exactly does mathematical induction work? Or, how does it guarantee the statement to be right for all natural numbers? Here is a structural summary of mathematical induction to answer this doubt:

- **Base Case**: that the statement works for an initial value k. In the case we are proving for all natural numbers, $k = 0$ is a popular choice. Therefore, we attempt to prove at this step that the prompt proposition $P(n)$ works for the value 0.
- **Induction Hypothesis**: We here make the assumption that for an arbitrary value $n \geq k$, $P(n)$ is true. Building on that, we proceed into the next step:
- **Inductive Step**: With the induction hypothesis, we here attempt to prove that $P(n+1)$ is true.

Let us see an example of mathematical induction utilized in a direct proof:

Think Brandon! Think! 3.1.1: Mathematical Induction in Direct, Algebraic Proof

Prompt: $\forall n \in \mathbb{N}(\sum_{i=0}^{n} i = \frac{n(n+1)}{2})$

For brevity, let the proposition $P(x)$ be the second clause of the prompt: $\sum_{i=0}^{x} i = \frac{x(x+1)}{2}$.
Base Case: Prove $P(0)$.

$$\sum_{i=0}^{0} i = 0 = \frac{0(0+1)}{2}$$

Hereby, $P(0)$ has been proven.
Induction Hypothesis: Assume for an arbitrary k, $P(k)$ holds, such that $\sum_{i=0}^{k} i = \frac{k(k+1)}{2}$.
Inductive Step: Assuming $P(k)$ is true, prove $P(k+1)$.

$$\begin{aligned}
\sum_{i=0}^{k} i &= \frac{k(k+1)}{2} \\
\sum_{i=0}^{k+1} i &= \frac{k(k+1)}{2} + (k+1) \\
&= \frac{k^2+k}{2} + \frac{2k+2}{2} \\
&= \frac{k^2+3k+2}{2} = \frac{(k+1)(k+2)}{2} = \frac{(k+1)((k+1)+1)}{2}
\end{aligned}$$

As seen in the above proof, we assumed the induction hypothesis to hold when attempting to complete the inductive step. Often, the induction hypothesis even serves as a critical step towards the completion of an induction.

3.1.2 Graphical Mathematical Induction

Mathematical Inductions not only work from the algebraic perspective. Since it can prove statements to hold for all natural numbers, these statements ought to not only be propositions for algebraic identities. It can also be geometrical phenomenons. Let us view an example from the lecture notes:

Think Brandon! Think! 3.1.2: Mathematical Induction in Geometric Proof

Define n-colorable as able to color a map with n colors such that no adjacent areas share the same color.
Prompt: Let $P(n)$ denote the statement *"Any map where the border of all areas are marked by within n lines is two-colorable"*, then prove that $\forall n \in \mathbb{N}(P(n))$.

Base Case: Consider $P(0)$. In this case, a map with 0 lines to mark the border of areas is just one huge area, which is definitely two-colorable.
Induction Hypothesis: Assume for some arbitrary $k \geq 0$, $P(k)$ holds.
Inductive Step: Let us assume that the induction hypothesis is true. In the inductive step, our map will provided a new line to mark the borders of newly created regions. These newly created regions of the map must come as a partition of previously whole areas.
Now, let us set the line onto the map. Let us defined the side left or above the new added line (the $(k+1)^{th}$ line) as the left hand side (LHS), and the other (RHS).
We notice that the current map would not be two-colorable, because the newly created regions must have the same color with their other partition.
Let us save this proof with a small maneuver. By swapping all colors existing on the LHS to their opposite, then while the two-colorable property is preserved under any color swapping, all original whole areas will now hold partitions of different colors, for their partitions are splitted across the LHS (the swapped side) and RHS

(the unswapped side).
Consequentially, for the k-line map is two-colorable as the induction hypothesis states, the $k+1$-line map must also be. Therefore, under such assumption, the inductive step works and $P(k+1)$ holds.

Not only are mathematical inductions useful on such graphical proofs, it can further provide strong proofs towards theorems that are considered really hard to prove in algebraic means.
Graphical proofs rely more on geometry and logic than mere algebra, which makes it different in property from the conventional mathematical inductions we see in the lecture notes dealing with number theory and arithmetic manipulations. This comes very clear when we start using abstractions for graphical objects in discrete mathematics.

3.1.3 The Strength of Induction Hypothesis

However, mathematical induction is not all-able. If the prompt is too difficult to prove, or if the induction hypothesis is too weak for us to proceed the inductive step with, we will be unable to complete a proof.
Let us visit an example from the lecture notes regarding this problem.

Think Brandon! Think! 3.1.3: When the Induction Hypothesis is SUSpicious

Prompt: for all $n \geq 1$, the sum of the first n odd numbers is a perfect square.

Base Case: Prove the prompt for $n = 1$. The first odd number is 1, and $1 = 1^2$ is a perfect square.
Induction Hypothesis: Assume that for an arbitrary $k \geq 1$, the sum of the first k odd numbers is a perfect square.
Inductive Step: The $(k+1)^{th}$ odd number would be (2k + 1). Considering the sum of the first k odd numbers is a perfect square m^2, the sum of the first $k+1$ odd numbers would be m^2+2k+1.
What next.....? Proof incomplete.

We were not able to complete the above induction because our induction hypothesis was not powerful enough to drive us towards completing the inductive step. This induction hypothesis, or a prior assumption that we may call **priori**, needs to be a different proposition that speaks the same thing as an induction hypothesis does. Or, it has to be a stricter, stronger proposition for us to assume.
Let us try a different induction hypothesis for the same prompt:

Think Brandon! Think! 3.1.4: When the Induction Hypothesis is Reliable

Prompt: for all $n \geq 1$, the sum of the first n odd numbers is n^2.

Base Case: Prove the prompt for $n = 1$. The first odd number is 1, and $1 = 1^2$.
Induction Hypothesis: Assume that for an arbitrary $k \geq 1$, the sum of the first k odd numbers is a k^2.
Inductive Step: The $(k+1)^{th}$ odd number would be (2k + 1). Considering the sum of the first k odd numbers is a perfect square k^2, the sum of the first $k+1$ odd numbers would be $k^2+2k+1 = (k+1)^2$.
What next.....? Proof incomplete.

While this prompt allows us to prove the previous, it also has a stricter proposition to prove with, which comes with a more convenient statement to prove with.
In other words, the original prompt was too vague, causing our Induction Hypothesis to be weaker.
As for how we can discover a stronger hypothesis, we can usually manually demonstrate the proposition of prompt and attempt to see the pattern in which it was achieved:

Think Brandon! Think! 3.1.5: Finding A Stronger Prompt

Prompt: for all $n \geq 1$, the sum of the first n odd numbers is a perfect square.

$$n = 1 : 1 = 1^2$$
$$n = 2 : 1 + 3 = 2^2$$
$$n = 3 : 1 + 3 + 5 = 3^2$$
$$n = 4 : 1 + 3 + 5 + 7 = 4^2$$

We found there seems to be a pattern that: Prompt: for all $n \geq 1$, the sum of the first n odd numbers is n^2.
We then performed mathematical induction on it as an attempt to find a stronger hypothesis to prove the original prompt with, which worked!

3.2 Simple Induction and Strong Induction

In this section, we describe strong induction as an idea similar to simple induction, demonstrate the TPO of using either techniques, and will form an analogy of both inductions as tools that perform the same purpose with different equipped power.
We might recognize a strong induction as a TI-84, if simple induction is a TI-83. How so? Let us attempt to answer this by exploring the contents of this section.

3.2.1 The Strength of Induction: Simple or Strong

Up until now, we have used the three-part format of mathematical induction known as "simple induction", or "weak induction".
Notably, there is another notion of induction called "strong induction", which expects a slightly different induction hypothesis:

$$\wedge_{i=0}^{k} P(i) \text{is true}$$

While both forms of induction hold the same purpose and accomplish the same result, the amount of power needed in completing objectives is different across these two tools. Strong induction provides us a stronger hypothesis to work with, thus is more simple to use. Let us demonstrate so with an example:

Think Brandon! Think! 3.2.1: New Technique: Strong Induction

Prove: $(\forall n \in \mathbb{N})((n > 12) \implies (\exists x, y \in \mathbb{N}(n = 4x + 5y)))$

Base Case:

$$n = 12 = 3 \times 4 + 0 \times 5$$
$$n = 13 = 2 \times 4 + 1 \times 5$$
$$n = 14 = 1 \times 4 + 2 \times 5$$
$$n = 15 = 0 \times 4 + 3 \times 5$$

Induction Hypothesis: Assume that the prompt holds for all $12 \leq n \leq k$ for some arbitrary $k \geq 15$.
Inductive Step: We need to prove the claim for numbers beyond such limit k. We will prove for the case $n = k + 1 \geq 16$, or with some algebra: the case $(k+1) - 4 \geq 12$.
Since $12 \leq (k+1) - 4 \leq k$, by the induction hypothesis, $\exists a, b \in \mathbb{N}((k+1) - 4 = 4a + 5b)$.
It would then provide: $(k+1) = 4a + 4 + 5b = 4(a+1) + 5b$. Therefore, the prompt holds for value $(k+1)$

Let us provide another example to witness the efficiency in strong induction, as well as to explore an example on the lecture note.

Think Brandon! Think! 3.2.2: New Technique: Strong Induction, ex2

Prove that: every natural number $n > 1$ can be written as a product of one or more primes.

Base Case: The number $n = 2$ itself is prime, therefore can be written as a product of a prime number (itself).
Induction Hypothesis: Let us assume the prompt holds for all numbers $2 \leq n \leq k$, for an arbitrary k.
Inductive Step: We face the number $k+1$ to prove the prompt for. Here are two cases: either $k+1$ is prime, or not prime.
If $k+1$ is prime, then the problem is resolved as it can be written as a product of one prime number (itself).
If it is not prime, however, then we would assume it is the product of two natural numbers x, y such that $1 < x \leq y < k+1$, as multiplication should work.
However, such numbers x, y should by the induction hypothesis be able to be written as a product of one or more prime numbers. If so, then so can $k+1$. Via proof by case and mathematical induction, we have proved the prompt to hold true over the range of natural numbers it quantifies over.

3.3 Build-Up Error

However, it is very easy to make mistakes on mathematical inductions if we blindly assume the induction hypothesis to be non-vacuously true.
In other words, we can dangerously assume for an arbitrary k and the prompt's proposition P that $P(k)$ is true, while it might not be!

Think Brandon! Think! 3.3.1: When Mathematical Induction is SUSpicious

Prompt: Every horse has a same color

Wrong Proof:
Base Case: Let there be only one horse. One horse has one color, prompt is trivially true.
Induction Hypothesis: Let a set of n horses always have the same color. (Hey, this is MEGA-SUS! We'll explain why later.)
Induction Step: Let the new set of horses have $n+1$ members. Then, any n-element subset of this set of horses must have the same color, thus all horses must have the same color!

Where did it go wrong? Or, more exactly, why is mathematical induction sus? Well, there suddenly exist counterexamples to the induction hypothesis, and we blindly assumed that there isn't.
This is known as a **build-up error**, where we falsely assume the prompt to work for any natural number input already and thus falsely prove the prompt as right.

Let us take a stricter look at the above proof.
In the induction step, let us first remove an arbitrary horse from the set of $n+1$ horses. The induction hypothesis can tell us that this leftover set of n horses all have a same color.
But when we apply $n = 1$, this entire claim becomes suspicious: we, if following the induction step, actually get two disjoint sets (which means without common elements) of horses, each of cardinality 1. So the horses are never in the same color group, and cannot necessarily just have the same color!

The build-up error occurs: it shows some valid reasoning via a reasoning that is fundamentally flawed, and it is up to the prover to recognize the flaw.

3.4 Personal Regard on This Topic

Mathematical Induction is difficult to deal with, because induction hypotheses are sometimes hard to seek and in many cases difficult to configure and debug around.
But as always, practicing will help a lot in familiarizing with the content and coming with esoteric and maverick ways to prove diverse prompts!
For more advices regarding proofs, try searching in the previous chapter and your experiences.

Chapter 4

The Stable Matching Problem

In this chapter, we introduce the Stable Matching Problem and the Propose-And-Reject Algorithm, and use it as a case study of how proofs can be applied to problems of the computer science field to guarantee the efficiency and legitimacy of an algorithm.

4.1 Description of Stable Matching Problem

In this section, readers will be introduced to a frequently occuring problem of instutitions, called the "Stable Matching Problem", as well as learn an algorithmic solution of it that we will analyze the inner workings of in the next following section.

4.1.1 Stable Matching Problem and Propose-And-Reject Algorithm

Say we have an employment system, having to match n jobs to n candidates. Each job has a list of n candidates ordered based on preference, while a preference-ordered list of n jobs also exist for each candidates.
Our problem is, how do we find the best match, defined as in "switching jobs does not benefit each candidate"?
The algorithm that achieves this is called the "Propose-and-Reject algorithm", or more formally known as the Gale-Shapley algorithm. Let us see how this algorithm works in the hypothetical context we presented above, in "phases":

Think Brandon! Think! 4.1.1: How does the Gale-Shapley Algorithm Work?

At phase 1, every job position proposes to the most preferred candidate on its list who has not yet rejected their position.
At phase 2, each candidate collect all offers they receive, and responds "maybe" to their favoirte positions while "no" to other offers.
At phase 3, each rejected position crosses off the candidate who rejected them.
We repeat the loop from phase 1, as we have completed phase 3. We stop looping when no offers are rejected, as that means each candidate has a job offer of their "maybe". The candidates will then accept their offers.

In the following section, let us analyze this algorithm and answer the essential query: "How on Earth does this simple approach solve the significant problem?"

4.2 The Properties of Propose-And-Reject Algorithm

In this section, we will analyze the properties of Gale-Shapley Algorithm, as in what makes it a working algorithm and provide the proof that it works. This section will involve many mathematical induction, and we will finally see

induction being applied to problems of the computer science fields!

4.2.1 The Properties of Propose-And-Reject Algorithm

When discussing whether an algorithm is good or not, we mind about two properties: whether it halts, and whether it outputs a stable (good) matching.
The former can be proven true using a proof:

Think Brandon! Think! 4.2.1: Does the Gale-Shapley Halt?

On each day the algorithm doesn't halt, at least one job must eliminate some candidate from its list. Or else, it means that every candidate accepted an offer, and therefore the matching has ended.
Assuming this worst case for lists of n jobs and n candidates, this algorithm has a worst case runtime of $O(n^2)$ for it halts in at most a finite number, n^2, of iterations.

How do we define a "stable" match then? What is the heuristic, the metric, or the standards for deciding how good a matching is?
In the context we work with, let us define the metric as:

- A matching is unstable if there is a job and a candidate who both prefer working with each other over a current matching.
- A matching is stable if there are no couples like above, called "rogue couples".

First of all, we want to assess whether there could always be stable matchings, since if not, then the algorithm could still lead us astray from the combinations we want.
Let us observe how the algorithm's list of choices develops. While each jobs begin with its first choice, their choice prime candidates could betray and reject them. Therefore, jobs have worse choices over time. However, candidates will reject any choices that they do not like among all they receive and "maybe", therefore only having better choices over time.
This can be formalized into a lemma:

Lemma 4.2.1. Improvement of Choices in GS Algorithm

If job J makes an offer to candidate C, then on every subsequent day C has a job offer in hand which candidate C likes at least as much as J.

Time to do proofs for this lemma:

Think Brandon! Think! 4.2.2: And Prove the Improvement Lemma above

Let us perform induction on a day $i \geq k$

Base Case ($i = k$): on day k, candidate receives at least one offer from job J. Since the candidate C chooses whichever of the current favorite and offer from J is better, in the case J is not chosen, the offer C holds is better than J, and in the case J is chosen, C has an offer as good as J per se.
Induction Hypothesis: Let us assume the lemma holds for some arbitrary $n \geq k$.
Inductive Step: Let the current favorite offer of candidate C be B, and the offer received today be from job J. Since C must choose the better among B and J, if B is chosen, then B is at least more favored and liked than J, while if J is chosen, the offer that C holds is as good as J. Considering both cases, the offer can only be as good as or better than J.

Or, there is an alternative way to prove this lemma, utilizing another mathematical principle that makes its debut in this chapter.

Think Brandon! Think! 4.2.3: Proving the Improvement Lemma, Alternative

Before moving on, let us consider:

Definition 4.2.1. Well-Ordering Principle

If $S \subseteq \mathbb{N}$ and S is not empty, then S has a smallest element. Or, put in mathematical symbols as an implication:

$$S \subseteq \mathbb{N} \wedge S \neq \emptyset \implies \exists x \in S(\forall y \in S(x \leq y))$$

Now, let us assume that there exists an $n > k$ after our base case in the previous induction proof such that at day n, a first counterexample to the lemma occurs. The candidate C has either no offer or an offer from some other job H less favorite than the current favortie J.
On day $n-1$, candidate then receives an offer from some job K and liked it at least as mucha s J, which is as the lemma proposes. Therefore, this offer K still exists on day n, the exception day, which must make the new favorite be better than K, thus J, thus H.
We see from the above logic that the proposition $\neg(P(i) \implies P(i+1))$ cannot hold for any values i. Therefore, by the law of excluded middle, $(P(i) \implies P(i+1))$ will hold for any arbitrary value i. A Proof by Contraposition guides to the completion of a mathematical induction.
The above proof is secured by the Well-Ordering Principle because the order of proof in induction that assumes the smaller natural number to be ordered before the larger natural number is required for both the hypothesis of some day n being a first counterexample and the basic hypothesis of induction on proving the proposition to an incremented number.

Now, we only need to know two more things. First of all, the propose-and-reject algorithm does halt, and furthermore, terminate with a matching. Second of all, the amtching produced by the algorithm is always stable, now that we have the improvement lemma to help proving this statement with.

Think Brandon! Think! 4.2.4: Does the GS Algorithm always terminate with a matching?

Let us suppose it does not, such that there is a job J left unpaired when the algorithm terminates. That means the job J has offered to and been rejected by all possible candidates.
By the Improvement Lemma, this means each candidate must have had a better offer than what job J provided. Therefore, there in fact exists n jobs better than J, leaving us with a list of $n+1$ jobs. However, we only assumed there to be n jobs.
Therefore, via proof by contradiction, we have proved the lemma:

Lemma 4.2.2. Termination of GS Algorithm

The Gale-Shapley algorithm always terminates with a matching.

Taking this to the next step, we verify the stability of each matching.

Think Brandon! Think! 4.2.5: The Stability of Matching by Termination

Let us consider an arbitrary couple (J,C) in the final result of matching.
Let us suppose that this arbitrary job J prefers some other arbitrary better candidate C^* to C. However, C^* is currently paired with some other job K, and by the Improvement Lemma, this must be because that K is at least as favorable as J, making C^* having fovored K more than J.
Job J also wouldn't want to switch to a worse candidate, so the best it can have now is C. The job J can therefore never be involved in a rogue couple.
We have proved there cannot be any job J involved in a rogue couple, such that for any candidate J prefers better, that candidate doesn't want J more.

Theorem 4.2.1. The Property of GS Algorithm

The matching produced by the Gale-Shapley algorithm is always stable.

4.2.2 Optimality of Propose-And-Reject Algorithm

We would also like to discuss whether the matches this algorithm provides is optimal. However, before discussing so, to make good mathematical propositions, we must define optimality.

Definition 4.2.2. Optimality of Match in GS Algorithm

For a given job J, its optimal candidate is the highest rank candidate on the preference list of J that J could be paired with in any stable matching.
For a given candidate C, its optimal job is the highest rank job on the preference list of C that C could be paired with in any stable matching.
Put more straightforward, optimal is the highest ranked choice posssible for the resulting match to be stable.
A matching where each job is paired with its optimal candidate is known as a job optimal matching. On the other hand, a matching where each candidate is paired with their respective optimal job is known as a candidate optimal matching. The opposite of this concept is "pessimal".

And now, let us decide whether the matching output by our GS Algorithm is job-optimal:

Think Brandon! Think! 4.2.6: Is the GS Algorithm providing a job-optimal matching?

For the sake of contradiction, let us propose that the algorithm doesn't.
Let there exist a day on which a job has its offer rejected by the optimal candidate, and the first day of this counter be on day k (mind that we are again using the well-ordered principle here via having a "first counterexample").
Suppose on that day, J was rejected by the optimal candidate C^* in favor of an offer from J^*. Then, this couple (J, C^*) should exist in some stable matching. Therefore, the matching would look something like:

$$T = \{\dots, (J, C^*), \dots, (J^*, C'), \dots\}$$

First of all, if C^* rejected J in favor of J^*, then C^* must have liked to work on J^* more. Furthermore, J^* is assumed to have made an offer to C^*, but accepts C' in turn. This means C^* is at least as good as C', and that J^* would've wanted to work with C^* instead.
We have came to see that (J^*, C^*) is in fact a rogue couple, making T an unstable match rather than a stable one as we assumed with the above logic.
Therefore, via proof by contradiction, we found:

Theorem 4.2.2. The Optimality of GS Algorithm I

The matching produced by the Gale-Shapley algorithm is always Job-Optimal.

as well as candidate-optimal:

Think Brandon! Think! 4.2.7: Is the GS Algorithm providing a candidate-optimal matching?

Let us utilize the Job-Optimality of GS Algorithm and let T be an employer-optimal matching:

$$T = \{\dots, (J, C), \dots\}$$

For the sake of contradiction, let us propose that there exists a stable matching:

$$S = \{\dots, (J^*, C), \dots, (J, C'), \dots\}$$

such that job J^* is ranked lower than job J on the preference of C.
However, if so, then C must have wanted to work with J more and J's current candidate C' is not preferred more than C given the stable amtching T.
We managed to show that the matching S is both stable and unstable, which itself is a contradiction. Therefore, the matching from GS Algorithm cannot be candidate-optimal.
If so, then the employer-optimal pairing is already the worst possible pairing for candidates, and therefore:

Theorem 4.2.3. The Optimality of GS Algorithm II

The matching produced by the Gale-Shapley algorithm is always Candidate-Pessimal.

Notably, this algorithm can still be modified in finer details to provide a priori in preference from jobs, as well as converting the algorithm into possessing candidate-optimality instead.

4.3 Personal Regard on This Topic

When doing proofs regarding stability, it is always advised to operate with imaginary entities. This almost always offers a more concrete insight to the proof of operation.
This algorithm is an interesting topic that demonstrate the importance of mathematical induction, as well as showcases the discussion and analytics we may perform on an algorithm.
Personally, here are a few topics that I chose to highlight during revision:

- **Improvement Lemma**: Candidates will only hold better or equal offers.
- **Optimality**: Candidate-pessimal and job-optimal for our current settings. However, by switching the places of candidates and jobs, this optimality reverses.
- **Stability**: Definition is customizable across different short answer settings, but the commonality is there should not be rogue couples (matchings), where the second and first element of some different pair wants to be with each other instead.
- **Uniqueness of Stable Match**: Stable matches can be generated via other algorithms. This is just one of them. Therefore, there can exist more stable matchings than we knoW.

Imaginary entities, or denoting entities, is VERY useful. It allows you to form arguments from constructions and produce a proof, which graphs, a mathematical object that we work with in the future, will require you to perform.
Please use it wisely.

Chapter 5

Graph Theory

In this chapter, we will discuss Graph Theory, a highly applicable set of mathematical theories that help with algorithmical operations via representational abstractions in computational and mathematical notions.

5.1 An Introduction to Graph Theory

5.1.1 Motivation

Abstraction. It is the universal theme of many things computer science. We attempt to capture the essence, simple representation of a complex situation, from concepts to functions.
And graphs happen to be a very popular abstraction. Graphs are sets of connections between customizable individuals, working from neural networks (individuals are neurons) to your mom (individuals are cells, connections are the chemical interactions of cells).
This mathematical object, graphs, have been developed by mathematicians and scientists. Eventually, we are provided a framework for the object, and this framework is known as **graph theory**.

Definition 5.1.1. Graph Theory

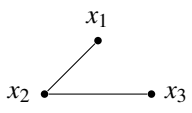
The mathematical theory of properties and applications of Graphs.

Albeit I realize that grades would be a popular motivation for studying, understanding graphs will provide good fluency across many advanced computational concepts. It is necessary, and is essential to the language of mathematics and computers.

5.1.2 Foundations and Formation of Graphs

Formally, a **graph** is defined by a set of **vertices** V and a set of **edges** E. Vertices are the individuals and edges are the connections.
Graphically, each vertex corresponds to the small circles in the following example of graphs, while edges are line segments connecting these vertices:

Figure 5.1.1. Example of a Graph



The *LaTeX* code is shown as follows:

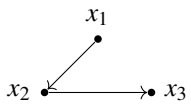
```
\begin{tikzpicture}
    [point/.style = {circle, fill, inner sep = 1pt}]
    \node[point] (1) [label=above:$x_1$] {};
    \node[point] (2) [below left of=1, label=left:$x_2$] {};
    \node[point] (3) [below right of=1, label=right:$x_3$] {};
    \draw (1) -- (2);
    \draw (2) -- (3);
\end{tikzpicture}
```

Think Brandon! Think! 5.1.1: How do you mathematically represent the graph in Figure 5.1.1?

The set of its vertices would be $V = \{x_1, x_2, x_3\}$.
Its edge, meanwhile, can be contained in the set $E = x_1, x_2, x_2, x_3$

Notice that here, E is a multiset, meaning there can be an element appearing multiple times if an edge is repeated. But we would really like E to be a set instead.
To let repeated edges that might have had different directions be counted distinctly, let us also define a directed graph, where each edge must have a direction:

Figure 5.1.2. Example of a Graph



The *LaTeX* code is shown as follows:

```
\begin{tikzpicture}
    [point/.style = {circle, fill, inner sep = 1pt}]
    \node[point] (1) [label=above:$x_1$] {};
    \node[point] (2) [below left of=1, label=left:$x_2$] {};
    \node[point] (3) [below right of=1, label=right:$x_3$] {};
    \draw[->] (1) -- (2);
    \draw[->] (2) -- (3);
\end{tikzpicture}
```

The example shown above is known as a **directed graph**, graphs equipped with directed edge.

Definition 5.1.2. Directed-ness of Edge

If an edge models a one-way path from one vertex to another, such an edge is known as a **directed edge**. Mathematically, it is represented as an edge with an arrowmark.
For the edge set from the above example,

$$(x_1, x_2) \in E \wedge (x_2, x_1) \notin E$$

If an edge is not arrowmarked, such that it allows travel between two vertices in both directions, then the edge is an **undirected edge**.

We can use a directed edge on multiple abstractions. For example, in a circuit diagram, current flows from one point to another in a single direction.
But, an undirceted graph is also useful for modeling, say a metro map, where it is free to commute between stations

for any direction.

5.1.3 Edge and Degree

For an undirected edge $e = (u, v)$ (or in equivalent notation, $\{u, v\}$), such edge e is said **incident** on vertices u and v. Meanwhile, these vertices connected by a same edge would be neighbors, or in more formal terms, **adjacent**. If a vertex is involved in no edge, it is called an **isolated vertex** and would also be disconnected from the graph.
A directed graph, however, has one-directional edges. In that case, vertices will still be incident, but the way to calculate visits slightly differs. We will see when we discuss the metric of visit frequency of vertices.
We may also use a metric to see how frequently a vertex is involved in the travelings of graphs: **degree**.

Definition 5.1.3. Degree

In general, **degree** measures the amount of edges a vertex has been visited by or involved in.
Such measure would differ slightly in the context of graph: whether it is directed or undirected.
If the graph is undirected, then degree of a vertex u is calculated as the amount of edges that involve u:

$$deg(u) = |\{v \in V : u, v \in E\}|$$

If the graph is directed, there are thwn two measures of degree:

- In-degree of u: The amount of edges that directs towards u.
- Out-degree of u: The amount of edges that directs from u.

If an edge originates from one vertex to itself, then this edge u, u is known as a **self-loop**. Again, depending on abstraction, this can either be utterly useless information or very useful information.
Since it is difficult to analyze graphs with self-loops, we will talk about graphs without self-loops in the following sections, and will not discuss the occassion of multiple edges between a pair of vertices (unless their directions are all distinct).

5.1.4 Traversing a Graph, Mathematically

We traverse graphs, algorithmically most of the time. How do we mathematically characterize a traverse?

Definition 5.1.4. Path

A **path** in a graph G is a sequence of edges. A path would, then, start from one vertex and end at a vertex.
Essentially, it is an itinerary, a projectory of traverse!
A path is *simple* when the vertices it travels are distinct.

Definition 5.1.5. Cycle, Tour

A **cycle** is a sequence of edges that starts and ends at the same vertex.
More specifically, all vertices traveled are distinct except the starting and ending vertex, and no edges are repeated.
If a sequence of edges simply starts and ends at the same vertex, it is known as a **tour**.

Let‘s make a brief summary:

Figure 5.1.3. Walk vs Path vs Cycle vs Tour

Type	No Repeated Vertices	No repeated edges	Start is End
Walk			
Path	◯	◯	
Tour			◯
Cycle	except start and end	◯	◯

5.1.5 Connectivity

Previously, we discussed the notion of isolated vertices. How do we, then, mathematically describe the connected-ness of a graph?

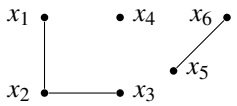
Definition 5.1.6. Connectivity

A graph is **connected** if there is a path between any two distinct vertices.
On the contrary is a **disconnected** graph.

Let me demonstrate with some examples:

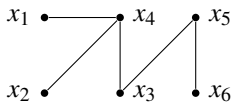
Figure 5.1.4. Connected vs Disconnected

Disconnected:



As you may have seen, vertex 4 is an isolated vertex that no one visits (just like me).
Meanwhile, the subgraph involving vertex 5 and 6 is disconnected from the rest of the graph, as well as the isolated vertex.

Connected:



In such a connected graph, there is a path from every vertex to another vertex.

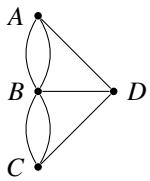
5.2 Koenigsberg's Seven Bridges and Eulerian Tour

5.2.1 The Seven Bridges of Koenigsberg

Imagine that there are seven bridges connecting four cities, abstractable to the graph as seen below:

Figure 5.2.1. Seven Bridges of Koenigsberg

The Seven Bridges of Koenigsberg can be represented by the graph as follows:



Where the seven edges each are distinct bridges, and the four vertices are distinct cities.

Now, the question is: is there a route that would traverse each seven bridges precisely once and return to the starting point.

5.2.2 Euler‘s Take

Euler, having contributed a lot to graph theory, came with some terminologies regarding itineraries of traversals:

Definition 5.2.1. Eulerian Itineraries

Eulerian Walk is a walk of graph G that uses each of its edge exactly once.
Eulerian Tour is a closed Eulerian walk where the starting and ending point are the same vertex.

This rephrases the Seven Bridge Problem to: ”Is there an Eulerian tour for its representative graph?”
The answer lies in the Eulerian Theorem:

Theorem 5.2.1. Eulerian Theorem

An undirected graph $G = (V,E)$ has an Eulerian tour iff G is even degree, and connected (except isolated vertices).
An **even-degreed** graph is a graph where all vertices have even degree.

Its proof is as follows:

Think Brandon! Think! 5.2.1: Prove the Eulerian Theorem!

Proving the forward direction of theorem:
Assume G has an Eulerian tour. Since it has to use every single edge, every vertex that has an adjacent edge will be involved in this tour, therefore connected with all other vertices on the tour.
Meanwhile, because everytime a tour enters a vertex along an edge, it exits along a different edge, each vertex will have a pair of edge adjacent to it. The start vertex would also have the same phenomenon, since we begin from and end at the starting vertex.
In that case, the graph would be even-degreed for the pairs of edge that occurs on vertices (which must all be used up), and would be connected.
Proving the backward direction of theorem:
Let us first have a subroutine called FIND_TOUR that finds a tour from a graph G.
Notice that while the tour is not necessarily Eulerian, it must get stuck at the vertex it started at, because it would then deplete the pairs of edges it can continue its travel with. This does not serve as a formal proof, but hopefully the conceptual summary of lecture notes' claim is good enough.
Then, let‘s also involve another subroutine SPLICE, which outputs a single tour T' provided a tour T that intersects each of the other arguments $T_1, \ldots, T_i$. Last but not least, these tours are merged together. The input tours are all edge-disjoint.
Last but not least, EULER, which once provided a graph G and starting vertex s, will attempt to find a spliced tours $G_1, \ldots, G_i$ from an arbitrary found tour T of $EULER(G_i, s)$, and its edge-disjoint intersected tours.
Let us use a shortened mathematical induction to prove the functionality of EULER:

Base Case:
The graph has 0 edges, there is no tour to find.

Induction Hypothesis:
EULER outputs an Eulerian Tour for any even degree, connected graph with at most $m \geq 0$ edges. This is a strong induction!

Induction Step:
Suppose G has $m+1$ edges. Removing the edges of T from G, since the tour would remove pairs of edges from vertices, we would be left with a remaining even-degree, connected graph with less than m edges.
This T intersects each of the edge disjointed but connected subtours G_i, at some first vertex of intersection.
The induction hypothesis allows us to assume these resultant subtours to have Eulerian tours. Therefore, our end result of EULER is in fact a splice of individual Eulerian tours, into a large tour whose union is all edges of G and thus also an Eulerian tour.

Such is the general description of this proof.

In this case, since the graph of Seven Bridge is not even-degreed, it cannot have an Eulerian Path. Therefore, the solution is impossible.

5.3 Planarity, Euler's Formula, Coloring

5.3.1 Trees

Unfortunate. Regardless, let's talk about trees.
As mentioned in CS61B, a graph is a tree if it is connected and acyclic.

5.3.2 Planar Graphs

What is a planar graph?

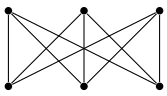
Definition 5.3.1. Planarity

A graph is planar if it can be drawn on a plane without crossings.

In that case, all trees are planar.

Symbol 5.3.1. Three Houses-Three Wells

The graph $K_{3,3}$ is a graph where there are two sets of vertices, each of size three, and all edges between the two sets of vertices are present:



Shown in above is an example of a nonplanar graph, $K_{3,3}$.
Another example of a nonplanar graph would be a complete graph with five nodes, notated as K_5. A complete graph is a graph where every possible edge is present.
And from here, let us introduce another useful notion in the discussion of graphs by its edges:

Definition 5.3.2. Bipartite Graphs

A bipartite graph, $G = (V, E)$, is a graph where vertices are split into two groups and edges only exist between groups.
Mathematically, let the groups of vertices be L, R, then $V = L \cup R$ and $E \subseteq L \times R$.

5.3.3 Euler's Formula

Planar graphs, when drawn on the plane, separates a plane into multiple regions. How do we express this fact mathematically?

Definition 5.3.3. Faces

The faces of a graph are the regions which the graph subdivides the plane into.
Faces are distinguishable for a planar graph, but not much so for nonplanar graphs!

Euler has contributed a great formula for planar graphs, which are great generalizations of polyhedra.

Theorem 5.3.1. Euler's Formula

Let v, f, e be respectively the amount of vertices, faces, and edges in a connected planar graph.
Then, $v + f = e + 2$.

And let's proceed with a proof:

Example Question 5.3.1: Prove Euler's Formula

We can perform mathematical induction on value of e.
Base Case $e = 0$:
In this case, since there is no edge, the amount of vertex and face are all 1. The formula holds.
Induction Hypothesis For all connected planar graphs, the prompt holds.
Induction Step Let's consider the following cases:

- If it is a tree, then the amount of faces is 1 and the amount of edge e in a tree is equal to $v - 1$.
- If it is not a tree, find a cycle of the graph and delete any edge of the cycle to break it. This will reduce both e and f by one. By induction hypothesis, formulas work in the smaller graph, and the arithmetics we just performed would prove the formula true in a graph with $e - 1 + 1 = e$ edges.

For a non-connected planar graph, the situation is slightly different.

Theorem 5.3.2. The Sparsity of Graph

Take its connected subpart, a planar graph that has $f > 1$ faces, and count the number of sides on each face. In the end, as we double count each side, we double count each edge of the graph, resulting in the conclusion:

$$\sum_{i=1}^{f} (\text{Number of sides in face } i) = 2e$$

Knowing that there cannot be parallel edges between two same nodes, and assuming that the graph has at least two edges (to secure at least three vertices), then each face has at least three sides.

Number of sides in face $i \geq 3$

$$\sum_{i=1}^{f} 3 \leq \sum_{i=1}^{f} (\text{Number of sides in face } i) = 2e$$

Since we are considering a connected planar graph, let us transform the above inequality according to Euler's Formula:

$$\begin{aligned} 2e &\geq 3f \\ 2e &\geq 3(e+2-v) \\ 2e &\geq 3e-3v+6 \\ e &\leq 3v-6 \end{aligned}$$

This implies that planar graphs are sparse, or, they cannot have too many edges.
Therefore, specific graphs whose number of edges and vertices do not follow the above inequality can be seen as non-planar:

Definition 5.3.4. Planarity

For a connected planar graph, $e \leq 3v-6$.
Mind that this does not serve as a planarity test, as such property is satisfied by $K_{3,3}$ as well.
However, its contrapositiv can serve as a non=planarity test.

In that case, both K_5 and $K_{3,3}$ are non-planar. We can explore these graphs further: these are the only non-planar graphs, made precise by the mathematician Kuratowski (where the capital K came from).

Theorem 5.3.3. A graph is non-planar iff it contains K_5 or $K_{3,3}$

Let us define contain as where the nodes in the graph K_5 or $K_{3,3}$ are identifiably connected as their corresponding graph rules via paths, such that no two of those paths share vertices.
In otherwords, let the graphs noted in this theorem be a subgraph of the larger inspected graph.
This mathematical result is sometimes also known as Kuratowski's theorem, and its direction imply that if a graph contains one of these two non-planar graphs, itself must also be non-planar.

5.3.4 Duality and Coloring

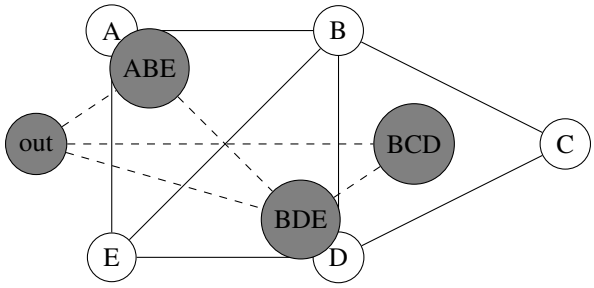
The Greeks knew everything, from the section above to the next notion:

Definition 5.3.5. Dual Graph

The dual graph $G*$ of a graph G is a graph formed by converting faces of G into vertices and drawing edges between adjacent faces.

And the CS70 Lecture Note will ask you to *think about it :::::D*. So do it, maybe search a few Google Images up. I can't LaTeX this. Skill issue.
But Brandon, **I hear your thoughts**. Why did you put a LaTeX-ized dual graph on the screen already

Figure 5.3.1. Dual Graph



and why is Duality important at all?
That's because it reduces area-coloring map into a planar graph. Since a map's faces ought to be distinguishable, its dual graph can serve as a planar graph for us to work with.
Now let's talk about coloring. Essentially, two-coloring a graph is working with a bipartite graph, where one group of vertex has one specific color, the other group the other color.
Or, we may proceed with a breadth-first algorithm that colors any uncolored neighbor with alternative colors. This will let us discover cycles of odd lengths in the event of failing to color. Notably, we have the four color theorem, which states that you may graph any map with four colors. This implies that any planar graph can be vertex-colored in four or more colors!

5.4 Classes of Graphs

5.4.1 Complete Graphs

Starting with a definition,

Definition 5.4.1. Complete Graph

A complete graph contains all possible edges between its vertices.
A complete graph on n vertex is unique and denoted as K_n, and has $\frac{n(n-1)}{2}$ edges.

5.4.2 Trees

Starting with a definition… well, in fact, many definitions.

Definition 5.4.2. Trees

Trees can be defined as any of the following:

- A graph that is connected and acyclic (as spoken in last section).
- A graph that is connected and has $n-1$ edges (this guarantees it to be acyclic).
- A graph that is connected, and the removal of an arbitrary edge disconnects the graph.
- An acyclic graph that becomes cyclic upon the addition of edge on any possible position.

Why do we want trees? I don't know, maybe retaking CS61B will provide you some insights.

In a rooted trees, which algorithms are mainly concerned with, there is a designated node called the root at the topmost level of a tree. The bottom-most nodes are called leaves, and the intermediate nodes are called internal nodes.
In a rooted tree, a root should never be a leaf; and throughout types of trees, leaves ought to be degree-1 vertices.

But why does this matter? That's a nice question. James. I have been tested on this when practicing Leetcode. I think that shows something.

5.4.3 Hypercubes

We would like a graph to be strongly connected, and a complete graph is such example. But, complete graphs take a lot, a lot of edges. Since graphs are abstractions of real-life objects, this implies huge operational and development costs. Our alternative solution would be a hypercube.

Definition 5.4.3. Hypercube

A hypercube is a graph whose vertex set is given by:

$$V = \{0,1\}^n$$

which is equivalent of the set of all n-bit strings.
The edge set is then defined as:

$$E = \{x, y : \text{x and y differ in one bit}\}$$

Such a hypercube is also called an n-dimensional hypercube.

Notably, we may deifne hypercube recursively.
The vertex set of a $n-1$ dimensional hypercube happens to be $\{0x | x \in V_{n-1}\}$. The n-dimensional hypercube is obtained by placing an edge between each pair of vertices in the 0-subcube, which we described via an $n-1$ dimensional hypercube, and the 1-subcube where every leftmost digit of such vertex group is 1 instead!
Well, let's take a further investigation into its connectivity, so to make sure that we are working with the alternative solution we have wanted:

Lemma 5.4.1. Total Number of Edges in Hypercube

By the definition of hypercube, $E(n) = 2E(n-1) + 2^{n-1}$, where $E(1) = 1$.
Using mathematical induction, we may show that $E(n) = n2^{n-1}$.
In a simpler approach, since the degree of each vertex is n and n bit positions can be flipped in any of the 2^n vertices, there is a total of $n2^n$ edges (non-disticnt) that we can count between cubes.
We counted each edge twice, so we divide that total number by 2 to attain the result that the total number of edges in a n-dimensional hypercube is still $n2^{n-1}$.

Theorem 5.4.1. The number of discarded edges from disconnected vertices

Let $S \subseteq V$ such that $|S| \leq |V - S|$, and let E_S denote the set of edges connecting S to $V - S$:

$$E_S := \{\{u, v\} \in E : u \in S \wedge v \in (V - S)\}$$

Then, it must be that $|S| \leq |E_S|$.

Example Question 5.4.1: Prove the above theorem

Let is perform mathematical induction on the dimension of hypercube.
Base Case: $n = 1$
We may only separate 1-dimensional hypercubes into one vertex and another. Either way, there will be only one edge connecting S and $V - S$, and the cardinality of such two sets would be 1. Theorem holds.
Induction Hypothesis: Assume claim holds for $1 \leq n \leq k$; we perform strong induction.
Induction Step: Prove the claim for $n = k + 1$.
Let us assume that among the two separated sets, it would be $|S| \leq 2^k$.

Let S_0 represent the vertices from 0-subcube in S, and respectively for S_1. Then, either S has a fairly eqaul intersection size with the two subcubes, or it does not.
Case 1: $|S_0| \leq 2^{k-1} \wedge |S_1| \leq 2^{k-1}$
The induction hypothesis works on both subcubes, which means there are at least $|S_0|$ edges between S_0 and its complement, and similarly for $|S_1|$ and S_1. If so, the total number of edges between S and $V - S$ would be at least $|S_0| + |S_1| = |S|$, as the theorem attempts to claim.
Case 2: $|S_0| \geq 2^{k-1}$
In this case, we cannot apply the induction hypothesis as wanted before on the 0-subcube, but only on the 1-subcube. But we can apply this onto $|V_0 - S_0|$ instead and infer the amount of edges between them is at least $2^k - |S_0|$. The current total of edges is calculated as $2^k - |S_0| + |S_1|$.
The edges that cross between the 0-subcubes and 1-subcubes would be at least $|S_0| - |S_1|$ because there is an edge between every pair of vertices $(0x, 1x)$. Therefore, finally, the number of edges crossing is indeed at least $2^k \geq |S|$ as desired.
(The $|S_0| - |S_1|$ was derived from the cardinality of complement from $\{x : 0x \in S_0\} - \{x : 1x \in S_1\}$).

5.5 Personal Regard on This Topic

This is probably the most abstract topic you'll encounter in CS70, just because everyone is new to graphs.
Here is also an organization of results we have proven during discussion and homework:

- Every even-degreed connected graph has an Eulerian Tour, a tour on which every edge is used. (Eulerian Theorem)
- If a graph contains $K_{3,3}$ or K_5, it is non-planar. (Kuratowski)
- For a connected planar graph, $v + f = e + 2$. (Euler's Formula)
- For a connected planar graph, $e \leq 3v - 6$. This inequality is even tighter on disconnected planar graphs. (Proven in Lecture)
- Every planar graph can be vertex-colored with four colors. (Four Color Theorem)
- Hypercubes of n dimension has $n2^{n-1}$ edges. (Proven in Lecture)
- The number of vertices in an undirected graph with odd-degree is even. (Discussion 2B)
- An n-dimensional hypercube can be edge-colored with n colors. (Discussion 3A)
- Every hypercube is bipartite. (Discussion 3A)
- A Hamiltonian Tour is a tour where each vertex appears only once, each pair of consecutive vertices are connected by an edge, and the first and last vertex of this path are connected by an edge. (Definition)

Chapter 6

Modular Arithmetics

In this chapter, we will discuss a very foundational aspect of modular arithmetics that supports a wide majority of computational methods. This aspect of mathematics is modular arithmetics: the arithmetics of remainders and divisions widely researched, boding influence from number theory, and potentially linear algebra!

6.1 Introduction to Modular Arithmetics

6.1.1 Motivation

Across several computational notions, we would utilize number lines that wrap around. What do we exactly mean? Think back to Project Enigma, where we increment letters by numbers, and we would like to increment the letter Z by one position. That would provide us A, but because Z is computational represented as number 25, and A 0, we need to wrap a number around back to 0 whenever it exceeds 26.
The mathematical operation that allows it is called a number circle.
On a number line, we proceed from one number to the next, until perhaps infinity; in a number circle, counting works like the hours of a clock: whenever we reach the maximum possible hour (12), we notate the next hour as the minimum possible hour (1). Essentially, the next number to the largest of a circle is the smallest number of a circle.
Mathematically, such arithmetics are known as **Modular Arithmetics**.

6.1.2 Foundation

Let us take alphabets as an example. To computationally represent them, we would use the integers in the following range:

$$\{0,1,\ldots,24,25\}$$

There is a one-to-one correspondence of the 26 digits towards 26 alphabets. To consider alphabets whose representation is temporarily larger than the maximum of this range (number circle), 25, since there are a total of 26 consecutive integers from 0 in the range: **the true numerical representation of a letter is the remainder of division from its current representation divided by the size of range**.

Symbol 6.1.1. Modulo

The Modulo operator, (mod), is used in the manner:

$$x \pmod{m} = \text{remainder of x divided by m}$$

For example:

$$26 \pmod{26} = 0$$

And as a review of the rules of remainder, let $r = x \pmod m$, it is ruled that $0 \leq r \leq m-1 \wedge r \in Z$.
Therefore, symbol wise:

$$29 \pmod{26} = -23 \pmod{26} = 3$$

Symbol 6.1.2. Properties of $\pmod{}$

- $(a+b) \pmod n = [a \pmod n + b \pmod n] \pmod n$
- $(a-b) \pmod n = [a \pmod n - b \pmod n] \pmod n$
- $(a \times b) \pmod n = [a \pmod n \times b \pmod n] \pmod n$

6.1.3 Set Representation

For a modulo k, it happens that we can categorize all integers into k sets S_i, such that:

$$S_i = z : z \pmod i = i$$

These sets are called residue classes $\pmod{} k$. And, to mathematically express their comraderies, we may use the following symbol:

Symbol 6.1.3. Congruency

$$x \equiv y \pmod k \iff (x \pmod k = y \pmod k)$$

For example,

$$29 \equiv -23 \pmod{26}$$

Let us attempt to prove a relevant theorem:

Think Brandon! Think! 6.1.1: Prove this theorem in the following box

Prove:

$$(a \equiv c \pmod m \wedge b \equiv d \pmod m) \implies (a+b \equiv c+d \pmod m \wedge ab \equiv cd \pmod m)$$

Let $c = a + k \cdot m$, $d = b + l \cdot m$:

$$\begin{aligned} c + d &= a + b + (k+l) \cdot m \\ &\equiv (a+b) \pmod m \end{aligned}$$

$$\begin{aligned} c\dot{d} &= ab + alm + bkm + klm^2 \\ &= ab + (al + bk + klm) \cdot m \\ &\equiv (ab) \pmod m \end{aligned}$$

6.2 Exponentiation

How can we compute $x^y \pmod m$?
Well, for instance, we can compute the sequence $x \pmod m, x^2 \pmod m, \ldots$ up to the y^{th} term, but that takes time exponential to the number of bits in y, which is less efficient than wanted.
Instead, let us not calculate all items of the sequence, but just some:

Theorem 6.2.1. Algorithm for Modular Exponentiation

We solve the value of $x^y \mod m$ with the following procedure:

- If y is 0, then return 1. This is trivial solution!
- Else, we'd use the following procedure:
  - Compute a new variable $z = x^{\frac{y}{2}} \pmod m$.
  - If y is even, then return $z^2 \pmod m$.
  - Else, return $z^2 x \pmod m$

Essentially, we use the mathematical facts that:

$$\begin{cases} x^{2a} = (x^a)^2 \\ x^{2a+1} = x \times (x^a)^2 \end{cases}$$

The concrete analysis of the above algorithm would be performed in CS170, but this algorithm holds essentially an $O(n)$ runtime, where n is the number of bits in y.
This is very efficient, much more efficeint than the exponential-time naive approach.

6.3 Bijections

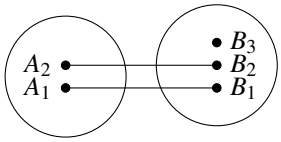
Take this as a side track!
A function is, in its nature, the relationship of sets: it is a relationship of elements from set A to set B. Mathematically expressed,

$$(\forall x \in A)(f(x) \in B), f : A \to B$$

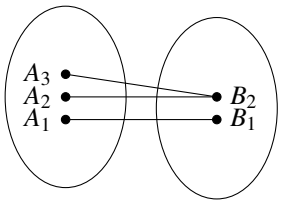
Definition 6.3.1. Injective, Surjective, Bijection

Injective, Surjective, and Bijective are all properties of functions, relationship between two sets A and B.
We call the A the set of *pre-images*, and B the set of *images*.

- **Injective**: Each image (output) has at most one pre-image (input). This type of function is thus called a "one-to-one" function.



- **Surjective**: Each image (output) has at least one image (input). This type of function is "onto".



- **Bijective**: A function is Bijective if the function is both injective and surjective. In other words, each image (output) has exactly one pre-image (input) and vice versa.

Let us practice this concept with some modular functions:

Example Question 6.3.1: Bijectivity of Modular Functions

Show whether the following functions are bijective:

$$\begin{cases} f(x) \equiv x+1 \pmod{m} \\ g(x) \equiv 2x \pmod{m} \end{cases}$$

Where both functions map $\{0,\ldots,m-1\}$ onto themselves.

The function f is bijective: every image, which are nonnegative integers from 0 to $m-1$, have a corresponding pre-image that is themselves.
The function g, however, would suffer under an even m as odd nonnegative images will have no preimages. In that case, g is only bijective for an odd m.

6.4 Inverse Operations

6.4.1 Multiplicative Inverse

We have discussed adding as an inverse operation of subtraction, but what is the inverse operation of multiplication in a modulo sense?
In normal arithmetics, multiplying a number by x is equivalent of dividing it by $\frac{1}{x}$, and for an equation $xy = 1$, we would call y the multiplicative inverse of x (because multiplying y reduces the product into 1). In linear algebra, we also had "inverse matrices" that can form multiplicative inverses via yielding a product of identity matrices $I_n = A^{-}1 \cdot A$.
In a modular sense, then, the multiplicative inverse would be defined as follows:

Definition 6.4.1. Multiplicative Inverse of Modular Arithmetics

For x, y, m such that $xy \equiv 1 \pmod{m}$, y is the multiplicative inverse of x modulo m.

There turns out to be conditions under which the multiplicative inverse of a number x modulo m exists:

Theorem 6.4.1. Condition of Existence for Multiplicative Inverse in Modular

Theorem: The necessary condition of multiplicative inverse of x modulo m is that $gcd(m,x) = 1$.

If there exists an inverse y of x modular m, then $xy \equiv 1 \pmod{m}$. Therefore, $xy = km + 1$.
In that case, $xy - km = 1$.
Let c be a common divisor of x and m, then that divisor must be the factor of xy and km too, thus the factor of $xy - km = 1$. However, the only number that can be a factor of 1 is 1.
Therefore, assuming there exists a multiplicative inverse, then the only possible common divisor of x and m is 1. Therefore, m and x are also coprime to each other.

And the multiplicative inverse would be unique:

Theorem 6.4.2. Uniqueness of Multiplicative Inverse under Modular

Theorem: The multiplicative inverse of x modular m is unique.

Let us state a as the multiplicative inverse of x modular m, then $ax \equiv 1 \pmod{m}$.
For the sake of contradiction, let's assume another distinct multiplicative inverse b, such that $bx \equiv 1 \pmod{m}$.
In that case, $bx - ax = x(b-a) \equiv 1 \pmod{m}$, so if two or more distinct multiplicative inverse of x modular m exist, then x and m are not coprime to each other. This causes a contradiction with the condition of existence, which states that such inverses can only exist for coprime x and m.
Therefore, by contradiction, there can only exist one multiplicative inverse of x modular m.

While the inverse of x can be written as $y = x^{-1} \pmod{m}$, this is generally considered an abuse of notation, as x^{-1} can also stand for $\frac{1}{x}$ under such ambiguous context.
Another way to express the multiplicative inverse can be $y = (x)_m^{-1}$

But we cannot just settle with knowing that an inverse exist. Let us attempt at finding a way to compute it; namely, let us produce an algorithm that can compute the multiplicative inverse of x modulo m.
Let us begin our passage by thinking about greatest common divisors. Suppose that for any pair of numbers x, y, then the greatest common divisors can be expressed as:

$$d = gcd(x,y) = ax + by$$

Let's express the relationship of numbers m and x when there exists a multiplicative inverse of x modulo m, in the above mathematical format:

$$1 = gcd(m,x) = am + bx$$

This would imply that $bx \equiv am + bx \equiv 1 \pmod{m}$, such that b is a multiplicative inverse of x modulo m.
Euclid's Algorithm, which is for computing greatest common divisors, helps us to find the integers a and b in the above equation. Thus, let's explore Euclid's Algorithm as a method of computing modulo inverses!

6.4.2 Euclid's Algorithm

The Euclid's Algoirithm, which is more of a folk algorithm by ancient engineers, relies on the theorem below:

Theorem 6.4.3. Reduction of GCD Computation via Modular Arithmetics

Theorem: Let $x \geq y > 0$, then $gcd(x,y) = gcd(y, x \pmod{y})$

Let us express $x = qy + r$, where $q \in \mathbb{Z}$ and $r = x \pmod{y}$. Note its an equal sign, not congruence.
If their greatest common divisor $gcd(x,y) = d$ divides x and y, then it also divides x and qy, thus also for $r = x - qy$.
Therefore, $gcd(x,y) = gcd(y, x \pmod{y})$.

The Euclid's Algorithm is the process to keep applying $gcd(x,y) = gcd(y, x \pmod{y})$ until the second argument becomes 0. At termination, the first argument is the greatest common divisor of original inputs x and y.
Now, let's verify whether it works:

Theorem 6.4.4. Correctness of Euclid's Algorithm

Prove that Euclid's Algorithm correctly computes $gcd(x,y)$.

Let us perform strong induction on the second argument (which should be the smaller of two inputs).
Let the proposition for functionality of Euclid's Algorithm be that:

$$P(n) : \text{The algorithm computes } gcd(x,n) \text{ for all } x \text{ correctly, and } x \geq y > 0$$

Base Case: $n = 0$
In this case, $gcd(x,0) = x$, which the algorithm correctly computes; meanwhile, the inequality stated in proposition regarding input size is maintained.

Induction Hypothesis: Assume P holds for all values $k < y$.
Induction Step: Now's the highlight, prove that $P(y)$ holds.
According to the steps of Euclid's Algorithm, $gcd(x,y) = gcd(y, x \pmod y)$. This equality is proven in an above cube, and it is also guaranteed that $y > x \pmod y$ by the nature of *mod*. Therefore, $P(y)$ holds!

The runtime of Euclid's Algorithm is $O(n)$, where n is the total number of bits in input (x,y). This is because for every two recursion call, the first initial input is reduced by at least one bit!

6.4.3 Extended Euclid's Algorithm

The extended Euclid's Algorithm provides us the coefficients a and b such that $d = gcd(x,y) = ax + by$:

Think Brandon! Think! 6.4.1: Extended Euclid's Algorithm

```
def extended_gcd(x, y):
    if y == 0:
        return (x, 1, 0)
    d, a, b = extended_gcd(y, x % y)
    return (d, b, a - x // y * b)
```

Let us now consider the inner workings of it: how the return for non-base-case functions?

Think Brandon! Think! 6.4.2: Inner Working of EEA

To establish some foundation, $d = gcd(x,y) = ay + b(x \pmod y) = Ax + By$.
We will need to find the expressions for new coefficients A, B, in terms of previous coefficients a, b throughout the calls of recursions.

$$\begin{aligned} d &= ay + b(x \pmod y) \\ &= ay + b(x - \lfloor \frac{x}{y} \rfloor y) \\ &= bx + (a - \lfloor \frac{x}{y} \rfloor)y \\ A &= b \\ B &= a - b \cdot \lfloor \frac{x}{y} \rfloor \end{aligned}$$

Or, instead of having to compute the weird arithmetics of new second coefficient, just alternate the secondary coordinate b into the primary of the next traced case (as algorithm demonstrates), and autonomously find a value of new b that works to satisfy $ax + by = d$.
This usually works for exams because d is usually 1. If it is not and the situation is complicated, stick to the algorithm.

6.4.4 Brief Notes: Division via Inverse

We have learned that to find the multiplicative inverse of x modulo m, we would just need to solve b from the equation $1 = ak + bn$, where $k = max(x,m)$ and n the other.
Let us attempt to apply this onto solving a congruence:

Example Question 6.4.1: Solve the following congruence using modular inverses

Solve: $8x \equiv 9 \pmod{15}$

The inverse of 8 modulo 15 is 2. How? $1 = -1 \times 15 + 2 \times 8$
So, let us multiply both sides of the congruence by $8^{-1} \pmod{15}$:

$$x \equiv 9 \times 2 \equiv 3 \pmod{15}$$

6.5 Chinese Remainder Theorem

The Chinese Remainder Theorem is another craftsman algorithm developed by Sunzi for counting the number of soldiers. The simplest case of the theorem is as follows:

Definition 6.5.1. Chinese Remainder Theorem, Claim

For m, n with $gcd(m,n) = 1$, there is exactly one $x \pmod{mn}$ such that:

$$x \equiv a \pmod{n} \wedge x \equiv b \pmod{m}$$

– Sunzi, The Art of War

Example Question 6.5.1: Prove the Simplest Case of Chinese Remainder Theorem

The proof follows from the existence of multiplicative inverse for n, m, respectively modulo m, modulo n. By prior discussions, this holds when $gcd(n,m) = 1$.
Let us follow from this fact and declare some variables:

$$\begin{cases} u \equiv m(m^{-1} \pmod{n}) \\ v \equiv n(n^{-1} \pmod{m}) \end{cases}$$

Here, for analysis on u:

$$m(m^{-1} \pmod{n}) \equiv mm^{-1} \equiv 1 \pmod{n}$$
$$u \equiv 0 \pmod{m} \text{ (because it is a multiple of } m)$$

Applying a similar logic onto the value of $n(n^{-1} \pmod{m})$, we will attain the following:

$$\begin{cases} u \equiv 1 \pmod{n} \wedge u \equiv 0 \pmod{m} \\ v \equiv 0 \pmod{n} \wedge v \equiv 1 \pmod{m} \end{cases}$$

And therefore, let $x = au + bv$:

$$\begin{cases} x = au + bv \equiv a(1) + b(0) \equiv a \pmod{n} \\ x = au + bv \equiv a(0) + b(1) \equiv b \pmod{m} \end{cases}$$

Such a solution can be argued unique:

Think Brandon! Think! 6.5.1: Might the solutions of CRT be unique?

Consider for the sake of contradiction a second distinct solution y exists, then due to the modular properties, $x-y \equiv 0 \pmod{m}$ and $x-y \equiv 0 \pmod{n}$.
Since $gcd(m,n) = 1$, and $x-y$ is some multiple of m,n, and $x \neq y$, then either $|x-y| = 0mn$ or $|x-y| \geq mn$.
The former implies $x = y$, while the latter suggests that one of x and y is not in $\{0,\ldots,mn-1\}$.

And therefore, it's not exactly that the solution is unique. It's that distinct solutions must have a difference that is a multiple of mn. In other words, it potentially has infinite solutions.
We have now figured that CRT helps to solve systems of congruences, and now we are able to solve systems of 2 congruences. What is left to us is to solve systems of more congruences.
And so, let us now move onto the more complex cases of CRT:

Theorem 6.5.1. Chinese Remainder Theorem

Let $n_1, n_2, \ldots, n_k$ be coprime natural numbers, then for any sequence of integers a, there is a unique integer $0 < x < N = \prod_{i=1}^{k} n_i$ such that:

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ \vdots \equiv \vdots \\ x \equiv a_i \pmod{n_i} \\ \vdots \equiv \vdots \\ x \equiv a_k \pmod{n_k} \end{cases}$$

where

$$b_i = \frac{N}{n_i}\left(\frac{N}{n_i}\right)^{-1}_{n_i}, \; x \equiv \left(\sum_{i=1}^{k} a_i b_i\right) \pmod{N}$$

– Sunzi, The Art of War

Think Brandon! Think! 6.5.2: Prove the Existence of Solution for CRT

The proof of solution existence follows a very simular framework to part (b).
Let us first abreviate an expression in terms of a function:

$$f(b) = \prod_{j=0, j\neq b}^{k} n_j = \frac{N}{n_b}$$

Let there be values:

$$x_i = a_i f(i) \cdot f(i)^{-1}_{n_i}$$

Then, following the similar logic of part (b), we can dictate that the solution would be:

$$x = \sum_i x_i$$

As for why the abvoe works:
For each congruence under modulo n_i we deal with, the only term left to interpretation due to not being a multiple of n_i would be x_i; however, due to how the term x_i is constructed via involving a number and its multiplicative inverse under n_i:

$$x_i \equiv a_i \pmod{n_i}$$

The uniqueness of this solution follows the fact that, for the sake of proof, let y be a distinct solution to the system from x:

$$\forall n_i (x - y \equiv 0 \pmod{n_i})$$

This follows from the observation that they are both equivalent to a same value under any modulo n_i to qualify as a solution.
But, the above claim would hint that $x - y \equiv 0 \pmod{N}$, stating that each solution is of distance N from each other, and thus there would be a unique solution if under modulo N.

Now, a linear algebra perspective:

Think Brandon! Think! 6.5.3: The set of solutions is a Vector Space

Here, each b_i is congruent to $1 \pmod{n_i}$ and $0 \pmod{n_j}$ by the theorem, so a multiple of b_i can be used to satisfy congreunce $\pmod{n_i}$ while not integering with other congruences of other modulo.
In that sense, we can construct multiple unit vectors from b_i across the indices of congruence to construct linear combinations that become a solution by itself. The set of all solutions is then that span.
This might be a convenient point of vectorizaton for compuation.

The uniqueness condition is very similar: distinct solutions must have a distance that is a multiple of N as defined in theorem.

6.6 Personal Regard on This Topic

Just like my friend who plays Smash Ultimate says when I first tried to play against a tournament player who plays an overpowered character:

My advice for you, is to have fun.

No, just kidding. Here are the advises I meant to put instead:

- Be aware of the timings at which you use $\equiv$ and $=$. These are different. If you are using *mod* as an operator, note it clearly.
- The proof for CRT is useful, and the reason why its solution works associates extremely closely to its proof. This is applicable for polynomials, surprisingly.
- Extended Euclid Algorithm will save you in the next chapter.
- If you are like most of the readers and this author might be, the original name of CRT in its native langauge is in case you want to yell some move names during your midterm at Pimentel 1 to cheer yourself up when using CRT.

Chapter 7

Public Key Cryptography

In this chapter, we discuss public key cryptography: a convenient means of confidential conversation, and explore it as an application of modular arithmetics, which was the explained subject of previous chapter.

7.1 Introduction to Cryptography

Cryptography is the study that develops the most secure ways of delivering messages.
The basic setting for cryptography is typically described over a plot involving three characters:

- Crewmate A: Trying to communicate confidentially over a link.
- Crewmate B: Trying to communicate confidentially over a link.
- Sus person: Trying to listen in crewmates' conversations to do sussy things.

Or, to formalize what is a 'sus person', feel free to use the word 'spy' or 'evesdropper' too.

To let the communication be confidential and secure between crewmates, each crewmate will apply an Encryption function E to a message x, and send $E(x)$ to the other crewmate. Upon the receipt, the other crewmate applies a decryption function D onto $E(x)$ such that $D(E(x))$ provides information resembling the original message x. In most cases, $D(E(x)) = x$.
These functions E and D are not necessarily single-variable. We will touch on this soon. But, without knowing anything about the inner workings of E and D, the 'sus person' cannot know what the original function x at all.

For many centuries, humans study **private-key protocols**, which also requires a codebook, which contains all future correspondence between encrypted and decrypted messages. This requires a codebook, to begin with.
On the other hand, **public-key schemes** allow crewmates to communicate, encrypt, and decrypt without the need of codebook. This seems off. If crewmate B and the sus person both have equal information on the encrypted message $E(x)$, are they not equally capable of solving the encryption?
The essence of **public-key schemes** hides in implementing a digital lock, which only the decrypter can have a key for. Therefore. the decrypter send the encrypter a method of sending secured message that the decrypter has the key for. While the sus person can attempt finding the key, it should be difficult to do so, so arduous that it is almost impossible.

That's the point of cryptography.

Now, since we also need a key to decrypt the message, the function D is indeed multivariate rather than single-variable on the encrypted message $E(x)$.

7.2 Cryptography in Modular Arithmetics Theorems

7.2.1 The RSA Scheme of Cryptography

The RSA scheme is based heavily on modular arithmetics.

Let p and q be two large primes (typically with 512 bits, so larger than 2^{511}).
Let $N = pq$. Messages to the decrypter are numbers modulo N, excluding 0 and 1.
Then, let e be any number relatively prime to $(p-1)(q-1)$.
The public key of decryption is the pair of numbers (N,e). While this key is published to the whole world, the numbers p and q are not public.
The private key of decryption, meanwhile, is $d = e^{-1}_{(p-1)(q-1)}$, which would exist by the definition of e.
Now, let us explore the encryption and decryption functions:

Definition 7.2.1. Encryption and Decryption functions for RSA Scheme

Provided the values:

$$\begin{aligned} &p,q \text{ are large prime numbers} \\ &N = pq \\ &e \text{ is coprime with } (p-1)(q-1) \\ &d = e^{-1}_{(p-1)(q-1)} \end{aligned}$$

Among these values, only N and e are known.

Encryption: $E(x) \equiv x^e \pmod N$

Decryption: $D(y) \equiv y^d \pmod N$

Example Question 7.2.1: Demonstration of RSA Scheme

Let $p = 5$, $q = 11$, so $N = pq = 55$. In this case, $(p-1)(q-1) = 40$. For a number coprime with it, let's choose $e = 3$.
The public key of this scheme is $(N,e) = (55,3)$, and the private key is $d \equiv 3^{-1}_{40} \equiv 27$.
For any message x that the encrypter will send, the encryption of x is $y \equiv x^e \pmod{55}$. Meanwhile, the decryption of y is $x \equiv y^d \pmod{55}$.
Therefore, for a message $x = 13$, the encryption is $y = 13^3 \equiv 52 \pmod{55}$, and the decryption of y is $x = 52^{27} \equiv 13 \pmod{55}$.

7.2.2 The Supporting Pillars: Theorems

Let us first introduce a significant theorem of modular arithmetics:

Theorem 7.2.1. Fermat's Little Theorem

Theorem: For any prime p and any $a \in \{1,2,\ldots,p-1\}$, $a^{p-1} \equiv 1 \pmod p$.

Let S denote the set of possible results from mod p.
Provided that p and a are coprime, $gcd(p,a) = 1$.
Consider the set of numbers $\{a,2a,\ldots,(p-1)a\}$. Because a and p are coprime, for two numbers of the prior set to be congruent under $\pmod p$, it must be their multiple is the same.

Consequentially, the following sets are equal:

$$\{1,2,\ldots,p-1\} = \{a \pmod p, 2a \pmod p, \ldots, (p-1)a \pmod p\}$$

Following from that equality of sets:

$$1 \times 2 \times \cdots \times (p-1) \equiv (p-1)! \pmod p$$
$$a \times 2a \times \cdots \times (p-1)a \equiv a^{p-1}(p-1)! \pmod p$$
$$1 \times 2 \times \cdots \times (p-1) = a \times 2a \times \cdots \times (p-1)a$$
$$a^{p-1}(p-1)! \equiv (p-1)! \pmod p$$

Here, multiply both sides by $(p-1)!_p^{-1}$

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

As a side note, the value $(p-1)!_p^{-1}$ exists because p is prime and must therefore be relatively prime to the components of $(p-1)!$.
Now, let us ensure that the message will get encrypted and decrypted correctly:

Theorem 7.2.2. Correctness of Encryption and Decryption

Theorem: Under the above definitions of the encryption and decryption functions E and D,

$$\forall x \in \{0,1,\ldots,N-1\}(D(E(x)) \equiv x \pmod{()N})$$

Mathematically expressing the prompt:

$$\forall x \in \{0,1,\ldots,N-1\}(D(E(x)) \equiv (x^e)^d \equiv x \pmod N)$$

By the definition of $d = e^{-1}_{(p-1)(q-1)}$, $ed \equiv 1 \pmod{(p-1)(q-1)}$. Let us then express $ed = 1 + k(p-1)(q-1)$, leading to:

$$x^{ed} - x = x^{1+k(p-1)(q-1)} - x = x(x^{k(p-1)(q-1)} - 1)$$

Let us inspect whether $pq | x^{ed} - x$, or in other words, $p | x^{ed} - x \wedge q | x^{ed} - x$.
For the argument of divisibility with p, we consider two cases: the trivial case where $p|x$ and the harder case where $p \neg | x$.
If x is not a multiple of p, then since p is prime, by Fermat's Little Theorem:

$$x^{p-1} \equiv 1 \pmod p$$
$$x^{k(p-1)(q-1)} - 1 \equiv 0 \pmod p$$
$$p | x^{k(p-1)(q-1)} - 1$$

If $p|x$, then the case is trivial. We are just subtracting two multiples of p.
We can prove under similar logic that $q | x^{ed} - x$, so in the end, $pq | x^{ed} - x$.

Chinese Remainder Theorem can also be used to provide a similar proof, where we construct a system of congruences undero modulo p and q exploiting $x^{ed} \equiv x$ under these modulos, and finally find $x \pmod{pq}$ to be the only viable solution by the uniqueness property of CRT.
So, why is RSA secure? It is based on the assumption:

> Given N, e, and $y \equiv x^e \pmod N$, there is no efficient algorithm for determining x.

This is because to guess x, we would have to try on the order of N values; however N is the product of two 512-bit prime numbers, which makes thi unrealistic.

Or, even if we factor N into its prime factors, this is still a majorly impossible problem to deal with computationally.
So in summary, the security of RSA depends on the difficulty to solve a message from it.

7.2.3 Prime Number Theorem

Now, all that is left is for the crewmates to:

1. Find huge prime numbers p and q.
2. Compute exponentials mod N.

Both of which are not so shrimple.
But, to find large prime numbers, we can attempt to find efficient algorithms of determining whether a number is prime, and fortunately, a large portion of numbers are prime.

Theorem 7.2.3. Prime Number Theorem

Theorem: Let $\pi(n)$ denote the number of prime numbers that are less than or equal to n.

$$\forall n \geq 17 \left(\pi(n) > \frac{n}{\ln(n)} \right)$$

For exponentiation, the exponentiation algorithm provided in Chapter 6 is actually good enough.
So RSA is quite plausible for implementation and usage!

7.3 My Personal Regard on This Topic

Takeaway: Fermat's Little Theorem is goated.
Otherwise, make sure you understand that in RSA Scheme...:

- The known values are N and e, where e is coprime with $(p-1)(q-1)$ and $N = pq$.
- The reason your algorithm makes the eavesdropper mald is because p and q are hard to find.
- If it works properly, $D(E(x)) \equiv x \pmod{N}$. You use the public key to encrypt and decrypt, it's the private key that depends on $(p-1)$ and $(q-1)$.
- This algorithm can work if you find two or more prime factors to form N, and you'd just adjust the formula for e and d accordingly. But, if you just have a prime number N as your public key, your private key is very findable because it's just a multiplicative inverse of a known number undero a known modulo.

If you're interested in cryptography, the mathematics of making hackers mald, make sure to tune in for cryptography classes!

Chapter 8

Polynomials

In this chapter, we discuss the application of polynomials and their mathematical properties, as well as a concise instance of its performance in cryptography.

8.1 Re-Introduction to Polynomial

Polynomial is a category of functions that are easy in their algebra expression while also widely applicable to various mathematical techniques. While we have already encountered polynomials in high school mathematics, let us define polynomial in a formal fashion that we might not have done before:

Definition 8.1.1. Polynomial

A polynomial of single variable is a single-variable expression with an associated function:

$$p(x) = a_d x^d + a_{d-1} x^{d-1} + \cdots + a_1 x + a_0$$

Here, while the variable and coefficients are usually real numbers, the exponents need be integer.
An n-degree polynomial is a polynomial whose largest exponent is n.

The root of a polynomial is the value x for a polynomial function p such that $p(x) = 0$.
And following the existence of "roots", we now discuss some properties of roots and polynomials.

Symbol 8.1.1. Properties of Polynomials

1. A non-zero n-degree polynomial has at most n roots.
2. Given $n+1$ pairs $(x_1, y_1), \ldots, (x_{d+1}, y_{d+1})$ with all x_i distinct, there is a unique polynomial $p(x)$ of degree n such that $p(x_i) = y_i$ for $1 \leq i \leq d+1$.

The implication of property 2 would be that a line is define-able by two points.

8.2 Polynomial Interpolation

Polynomial Interpolation is the property that helps locate the n-degree polynomial described in property 2, and the name of the algorithmic method for it is Lagrange Interpolation:

Think Brandon! Think! 8.2.1: A Startpoint

Suppose that we will construct a polynomial $p(x)$ where $y_1 = k$, $y_j = 0$, where $2 \leq j \leq d+1$.
Let us begin with an arbitrary d-degree polynomial:

$$q(x) = (x-x_2)(x-x_3)\dots(x-x_{d+1})$$

Here, since $(x-x_j)$ is involved as a factor of $q(x)$, we can confirm $q(x_j) = 0$.
Meanwhile, as $q(x_1) = (x_1-x_2)(x_1-x_3)\dots(x_1-x_{d+1})$, we may define $p(x)$ as follows:

$$p(x) = \frac{kq(x)}{q(x_1)} \rightarrow p(x_1) = \frac{kq(x_1)}{q(x_1)} = k$$

Let us generalize the above example into the construction of polynomial $\Delta_i(x)$, the d-degree polynomial where $y_i = k$, $y_j = 0$:

$$\Delta_i(x) = \frac{\prod_{j\neq i}(x-x_j)}{\prod_{j\neq i}(x_i-x_j)}$$

So, for a set of $d+1$ points $\{(x_1,y_1),\dots,(x_{d+1},y_{d+1})\}$, we will respectively produce $\Delta_1(x),\dots,\Delta_{d+1}(x)$. We can acquire the polynomial p:

$$p(x) = \sum_{i=1}^{d+1} y_i\Delta_i(x)$$

We have put our coefficients y_i back at this point, so the definition of $\Delta(x)$ is easier to handle.
Let us discuss why such a p work. First of all, p is still d-degree, since we are just adding multiple d-degree polynomials together. Second of all, when evaluated at x_i, the *Delta* polynomials that contain $(x-x_i)$ will not contribute to the sum due to being zero, and will let $\Delta_i(x)$ alone contribute!
The discussion of second property above will probably remind you of constructing solutions for Chinese Remainder Theorem!

Theorem 8.2.1. Property 2 of Polynomials from Section 8-1

Theorem: Given $d+1$ pairs $(x_1,y_1),\dots,(x_{d+1},y_{d+1})$, with all x_i distinct, there is a unique polynomial $p(x)$ of degree at most d, such that $\forall i(p(x_i) = y_i)$.

The existence of such a polynomial is secured by Lagrange Interpolation. We now just perform a separate proof for uniqueness.
For the sake of contradiction, assume the existence of $q(x)$ such that $q(x_i) = y_i$ for all listed pairs.
Then, consider a polynomial $r(x) = p(x) - q(x)$. Since p and q are, for the sake of contradiction as aforementioned, considered different polynomials, $r(x)$ must be a non-zero polynomial of degree at most d. Property 1 suggests it can only have at most d roots.
However, $\forall i \in 1,2,\dots,d+1(r(x_i) = p(x_i) - q(x_i) = 0)$. In this case, r has at least $d+1$ roots, producing a contradiction.
Therefore, there can only exist one polynomial to satisfy the listed points.

8.3 Re-Introducing Polynomial Division

For a polynomial $p(x)$ of degree d, we may divide it by another polynomial $q(x)$ of degree $\leq d$ via long division:

$$p(x) = q'(x)q(x) + r(x)$$

This technique has been employed in Algebra 2 as well as integration in MATH 1A (or 1B, somewhere in the Calculus BC range to be sure).
We would define the quotient of such division $q'(x)$, and the remainder $r(x)$. The degree of $r(x)$ must be smaller than

that of $q(x)$.
Instead of taking this space to re-introduce long division, let's leave that as an exercise for the reader to review high-school mathematics, and move on to the proof of Property 1 as listed in Section 8-1:

Theorem 8.3.1. Property 1 of Polynomials from Section 8-1

Theorem: A non-zero polynomial of degree d has at most d roots.

This proof can be constructed via the proof of two smaller claims.
Claim 1: If a is a root of $p(x)$ with degree ≥ 1, then $p(x) = (x-a)q(x)$, where $q(x)$ is a $d-1$-degreed polynomial.
To prove this claim, we can use the property of remainders in polynomial division. If we divide $p(x)$ by $(x-a)$, then the remainder of this division must be smaller than the degree of $(x-a)$, which would make the remainder a constant term c.
Menahilwe, since a is a root of $p(x)$, and $p(x) = (x-a)q(x)+c$, $p(a) = c = 0$.
Therefore, there doesn't really exist a remainder.
Claim 2: A polynomial $p(x)$ of degree d with distinct roots $a_1, \ldots, a_d$ can be written as $p(x) = c(x-a_1)\ldots(x-a_d)$, where $c \in \mathbb{R}$.
Time for mathematical induction to be back!
Base Case: $d = 0$.
If $p(x)$ is degree-0, then it is a constant c to be written in the above form.
Induction Hypothesis: The prompt holds for some $d \geq 0$.
Induction Step: prove the case for $d+1$.
Let $p(x)$ be a polynomial of degree $d+1$ with distinct roots, then Claim 1 supports that $p(x) = (x-a_{d+1})q(x)$ for some degree-d $q(x)$.
For all $i \neq d+1$, $p(a_i) = (x-d_{a+1})q(a_i) = 0$. It will be helpful if we reference the induction hypothesis, notice that $q(x)$ has d distinct roots, and thus is a polynomial of degree d.
Therefore, $q(x) = c(x-a_1)\ldots(x-a_d)$, and following the product form of p:

$$p(x) = c(x-a_1)\ldots(x-a_{d+1})$$

Claim 2 implies Property 1 if we can show that the only roots of $p(x)$ are $a_1, \ldots, a_d$, but that also can be confirmed from the factorizaton of such polynomial $p(x)$.

8.4 Interlude: Finite Fields

Property 1 and 2 of polynomial expression as noted above would both work when coefficients and bases are chosen complex or rational, other than the real cases we have assumed along the way. This is because the proofs of properties rely on legal arithmetics that recieve arguments of one set (rational, complex... etc.) and return arguments of a same set.
This would imply that if we have chosen to conduct the proof of properties 1 and 2 via, say natural numbers and integers, then since we cannot secure the property of recieving and returning arguments of same set, the proofs cannot function.
Is there a workaround for natural numbers then? If we work with numbers modulo some prime number m, then legal arithmetics will still return us any number from $\{1, \ldots, m-1\}$, so properties 1 and 2 would still hold! So interestingly, properties 1 and 2 hold under a finite set like that of numbers modulo m as well.
We describe such situation as we are working over a finite field.

8.4.1 Counting

How many possible polynomials are there modulo m? Well, for a n-degree polynomial who has m values to choose from per n coefficient, it would be m^n.

How many possible polynomials are there that satisfy k points modulo m?
Well, for each point we neglect, we gain m possibilities for that empty coordinate that can bode any value of y. Therefore, considering n points, the total number of possible polynomials modulo m would be m^{k-n}.

8.5 Secret Sharing

Let us attempt to devise a secret sharing scheme such that, for the condition of secret decryption to be having a k-member faction:

1. Any group of k of all members can pool their information to configure the secret.
2. No group of $k-1$ or fewer members have any information about the secret, no matter how they pool their knowledge.

So now, suppose there are n officials indexed from 1 to n, and the secret is a natural number s. Let q be a prime number lrager than n and s, let us work over the finite field of modulo q.
Pick a random polynomial $P(x)$ of degree $k-1$ such that $P(0) = s$, and provide $P(i)$ to each official indexed i.
Now, since you need k points to figure out the explicit expression of a $k-1$ degree polynomial $P(x)$, you must need at least k officials who hold one point of information to contribute.

8.6 Personal Regard on This Topic

The best way to remember that $n+1$ pairs interpolate a n degree polynomial is using your nose (if you have one; if you don't, I suggest you use other people's nose).
Brandon. How? Are you out of your mind?
Look at your nose, which has two points (or holes if you will?). These two points on your nose form a line, because we can interpolate a 1-degree polynomial (a line) via two points on your nose.
The reason why I chose nose is because I see noses everyday and that reminds me of Lagrange Interpolation.

When asked to develop a secret-sharing mechanism via polynomial, you'd like to pay attention to how many pairs/roots to give to each individuals for interpolation.
This can also associate to the notion of how much power someone has in a voting system to decrypt a secret to a specific party.
For instance, if person A has more pairs of the polynomial than person B has, then person A has a larger power in this voting system because he has more portions of the decryption key.

Chapter 9

Error Correcting Codes

We can deliver messages all the way we want, but what if the message is lost, dropped, or corrupted?
In this chapter, we learn about error correcting codes: a major study of mathematics and EECS with many underlying theories and applications on media playback devices. If you have guessed from the combination of disciplines involved, these applications contain significant linear algebra based on finite fields!

9.1 Erasure Errors

Let us consider a corrupted message as follows:

Figure 9.1.1. Erasure Error, Graphically Presented

These errors are erasure errors, referring to the above situation(s) where a message with n packets has lost at most k packets during transmission.
To restore such messages, we can use some mathematical procedure.
Let us assume the contents of each packet is a number modulo q, where q is prime. The properties over $GF(q)$ will then resolve the erasure error via polynomials. We are now applying note 8 contents!

Let us denote the message to be sent as $m_1, \ldots, m_n$, where m_i is a number in $GF(q)$. Then:

1. There exists a unique polynomial $P(x)$ of degree $n-1$ such that $P(i) = m_i$. Relevant properties were the focus subject of discussion in previous chapter.

2. For each message to be sent, $m_i = P(i)$. We may then generate additional packets via evaluating this polynomial P at points $n+k$. The interpolation would work fine as long as $n+k \leq q$, and thus we have attained a trasmitted codeword, which would have more packets than messages to account for packet loss.

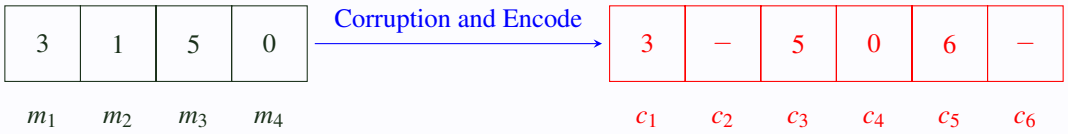
3. This polynomial $P(x)$ can be reconstructed from its values at any n distinct points, since it is degree $n-1$. Therefore, as long as n packets are transmitted from the codeword, we can recover the original message.

So in summary, we'd transmit a $n+k$ packet codeword, which would have at most k packages lost. However, the codeword is based on the original message, which has n packets and thus is degree $n+1$, and since we would have at least n packets left after any errorneous transmission or disk-biting, disk-crushing, using my DVD as a frisbee... you get the point: at least n packets are left for the polynomial interpolation to occur.
And the polynomial is characteristic of the message. As long as this polynomial can be retained from the successfully transmitted packets, we know the original message's individual packets via recovering the polynomial.

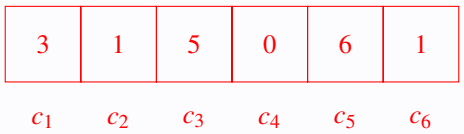
The uniqueness of our polynomial is thus CRUCIAL to the error correcting code!!!!!!!

Think Brandon! Think! 9.1.1: Example of Recovering Erassure Errors

Let us review the example from lecture notes. Suppose our final result of transmission was something like this:



The original message, notated as terms of m, will form a polynomial interpolation $P(x) = x^3 + 4x^2 + 5$.
Therefore, the codeword would actually be a rectangular array of:



But, we are able to interpolate that exact polynomial $P(x)$ even if we are left with just c_1, c_3, c_4, c_5, since we have these n points to interpolate a $n-1$ degree polynomial.
Therefore, computing $P(2)$ will recover m_2.

9.2 Polynomial Interpolation in Error Correction

There are ways to perform polynomial interpolation other than the Lagrange Interpolation, which will be useful in the next section.
The goal of polynomial interpolation, again, is to take input of $d+1$ pairs (points) and return a polynomial $p(x)$ such that it contains every point inputted.

Theorem 9.2.1. Polynomial Interpolation for General Error

Let us disucss Polynomial Interpolation in this alternative method.
We may first write a system of $d+1$ linear equations in $d+1$ variables, where the variables are the coefficients of the polynomial.
Then, each equation is obtained by fixing x to be one of $d+1$ values (first item of per inputted tuple), with the rightside of equation (which is constant for a system solving context) be the second item of corresponding tuple.
This polynomial interpolation method is fundamentally linear-algebra based, and only requires very fundamental skills (Gaussian Elimination) that we might just make it a new EECS 16A question.

9.3 General Errors

The context of general error is, in summary, the general case of erasure errors. In this case, erased packets are replaced by corrupted packets, and our problem setting is once again, to work with a length-n message with at most k packets corrupted.
By the similarity of context, you may probably have guessed that our settings for solution once again follows these rules:

- Each character is a number modulo q for some prime number q.
- We can describe a message as a polynomial $P(x)$ of degree $n-1$ over $GF(q)$, such that each pair of message (i, m_i) is contained by $P(x)$.
- To cope, with general errors specifically, we will transmit additional characters (so, sending a codeword, not a message).

In this case, we transmit $2k$ additional characters. Therefore, at least $n+k$ of the received packets in a codeword is non-corrupted.
Now that we have the settings of our riddle, we must discuss: does Bob have sufficient information for recovering, or interpolating, the characteristic polynomial of original message?

First of all, for any $n+k$ packets of the codeword, there should exist a unique polynomial that fits all $n+k$ packets. So it is at least possible to recover the polynomial.
The proof of its uniqueness follows: Among these $n+k$ points there can be at most k errors. Let $P'(x)$ be another polynomial of degree $n-1$ that goes through these $n+k$ packets, then at least n of the packets would be contained by both P and P' However, our previous chapter has already revealed that a polynomial of degree $n-1$ is defined by its values at n points, so P and P' are in fact same polynomials.
Now that we know it is possible to find a polynomial, what is an efficient method of finding such polynomial, amidst the possible unlucky case of encountering some of the k errors?

Theorem 9.3.1. Error-Locator Polynomial

An error locator polynomial is designed to locate errors:

$$E(x) = (x-e_1)(x-e_2)\dots(x-e_k)$$

We do not know this polynomial explicitly, since we don't know where errors are.
However, we have found that:

$$P(i)E(i) = r_iE(i)$$

where r_i is the value of packet contained at index i of the corrupted message.
Since the value of i has the constraint $1 \leq i \leq n+2k$, we can indeed produce a system of $n+2k$ linear equations with $n+2k$ unknowns.

Let us reflect on the approach.
Define $Q(x) := P(x)E(x)$, which has degree $n+k-1$ and therefore is described by $n+k$ coefficients.
Meanwhile, $E(x)$ is described by $k+1$ coefficients as a k-degree polynomial.
Therefore, we interpolate $n+k$ coefficients from $Q(x) = r_xE(x)$ and k coefficients from $E(x)$ (which is a polynomial of $(x-e_i)$, and we find that e_i and there are k of these).
This forms $n+2k$ linear equations for us to configure the coefficients of $Q(x)$ and $E(x)$.
At last, we perform long division to acquire $P(x)$:

$$P(x) = \frac{Q(x)}{E(x)}$$

Theorem 9.3.2. Correctness of Error-Locator Approach

Let us for the sake of proof assume the existence of alternative solutions Q', E', such that:

$$P(x) = \frac{Q(x)}{E(x)} = \frac{Q'(x)}{E'(x)}$$

Well, since we can follow from the definition of Q to determine that $Q(x) = r_x E(x)$, we may conclude ahead:

$$\begin{cases} Q'(x) = r_x E'(x) \\ Q(x) = r_x E(x) \end{cases}$$

Following the above equations:

$$\begin{aligned} Q'(x)E(x) &= r_x E(x)E'(x) \\ Q(x)E'(x) &= r_x E(x)E'(x) \\ Q'(x)E(x) &= Q(x)E'(x) \\ \frac{Q(x)}{E(x)} &= \frac{Q'(x)}{E'(x)} \end{aligned}$$

Therefore, any possible set of polynomials would function to find us the right polynomial!

9.4 Personal Regard on This Topic

It really probably is just about reading the notes a few more times and knowing how to perform polynomial interpolation :p
For now, there's nothing I'd like to add on top of the notes for this section.

Chapter 10

Counting

10.1 Counting be like

But Brandon, I hear your thoughts again.
Why in the holy Spaghetti Monster Shall we learn how to count?

The "counting" we address here in a discrete mathematical sense is not about counting from number to number in an arithmetic sequence. It's about estimating the number of possibilities.
For instance, consider a deck of 40 cards. What is the probability from which you draw all 5 parts of the inexorable Exodia? What is the total number of cases about the combination of 5 cards you draw from your deck, and from there within, what are all possible cases of drawing five cards of some same category (number 8, club, diamond...)?
Most importantly, what is the probability of it?
The techniques we employ to estimate and learn the total number of possible cases according to some conditions is known as **counting**.
In this chapter, let us begin from counting in simpler circumstances, like counting sequences.

10.1.1 Counting Sequences

Let there be a set $S = \{1, 2, \cdots, n\}$.
We may pick $k \leq n$ elements from this set S, one at a time, while removing what we have sampled (selected) from S.
This is known as **sampling without replacement**: once we select something from a set, it is removed from the set.
When counting the number of different ways to do this, either order matters, or order doesn't. In the case that order does matter, we are counting ordered sequences.

The most daily-life example would be Poker (without money, that's the relevant part .-.).
When we deal a card, that card dissappears is sampled from, as well as removed the deck. Meanwhile, when the sequence is ordered, the sequence of dealing card A then card B is different from the sequence of dealing card B then card A.
Therefore, at the first card, we have 52 disctinct possibilities for the card we now deal.
The second card, however, depends on the first card, since it cannot coincide with the first card. Your hand cannot have two exactly same card. That is cheating, and COMPSCI 70 does not condone cheating.
This means the second card has 51 possible choices, whichever card we picked from the first. Notice again that albeit the number of possibilities is the same, the second card you can get comes from a different set of cards depending on your first card.

Therefore, the total amount of possibilities for drawing 5 cards from a 52-card poker deck is in fact:

$$52 \times (52-1) \times (52-2) \times (52-3) \times (52-4)$$

This brings us to the **First Rule of Counting**:

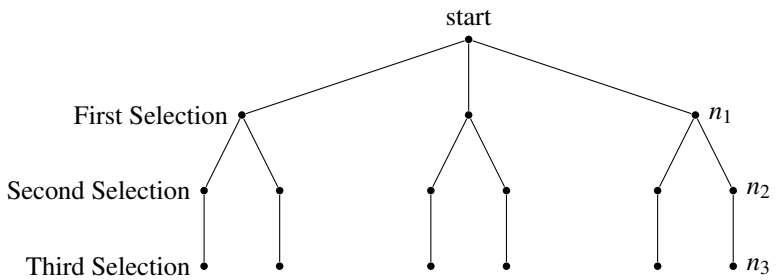
Axiom 10.1.1. The First Rule of Counting

If an object can be made by a sucession of k choices, where there are n_i ways to make the i^{th} choice, then the total number of distinct objects that can be made in this way is:

$$\pi_{i=1}^{k} n_i$$

Let us use this opportunity to introduce another instinct:

Figure 10.1.1. The Tree of Counting



Say we are looking for the number of ordered sequences produced from sampling from a three-card deck. Then, we will once again have 3 possibilities for the first card, 2 for the second card, and 1 for the third card (provided it's the last card chosen).
In the above figure, each leaf node is a termination of the sampling process: a possibility. Therefore, we just count the number of leaf nodes that exist in this tree: in this case:

$$n_1 \times n_2 \times n_3 = 3 \times 2 \times 1 = 6$$

10.1.2 Counting Sets

Sets are slightly different stories from sequences, since there is no order in set.
Put in a deck drawing context as discussed in prior subsection, we would then be counting the number of distinct hands: where hands do not all have same members (cards).
To count the distinct subsets of a set S (in this case our deck) with 5 cards, we can let each sequence get placed into some bin corresponding to the set of 5 elements in the sequence. Each bin must contain 5! ways of ordering the 5 elements in the set.
This means we just have to find the number of bins (distinct sets), which is essentially dividing the number of ordered sequences with 5! for this context:

$$\frac{52!}{(52-5)!5!}$$

This quantity, $\frac{n!}{(n-k)!k!}$, stands for the number of k-element subsets from an n-element set S, the result of our counting. The alternative but more popular expression of this is $\binom{n}{k}$:

Theorem 10.1.1. Second Rule of Counting

Assume an object is made by a succession of choices, and the order in which the choices are made does not matter.
Let A be the set of ordered objects, B the set of unordered objects. If there exists an $m-to-1$ function $f : A \to B$, then we can count the number of ordered objects and divide by m to obtain the number of unordered objects.

So in the hand-counting context of this subsection, we had a 5!-to-1 function from set of all ordered sequences to set of all distinct hands (which are the unordered sequences we count for).

10.2 Sampling with Replacement

Sampling with replacement is another different scenario of sampling we'd like to consider.
Perhaps the most daily-life example would be gacha games, where in each trial players have a constant probability of obtaining a specific in-game item.
To avoid counterarguments: if your game doesn't have constant probability and is also not sampling without replacement, it's a third type of scenario where we have a separate probability model and is not simplistic enough for the scope of this section. I don't know, you may have played the wrong game.
But perhaps, a slightly more relatable example would have been tossing coins. In each coin toss, the only two possible results are heads and tails. You can sample a head, but cannot remove the possibility of rolling a head from the total possible consequences of tossing a coin. Therefore, this is sampling with replacement.
I also don't want to talk about instances where your coin does not land on head or tails. As COMPSCI 70 emphasize, we do not condone cheating.

What are the mathematical properties of sampling with replacement?
Well, since no elements are removed from S, the number of possible choices across each selection will be constant, provided that the numebr of elements in S is always constant and does not decrease.
Since we have n choices in each trial, across k selections in the process of creating a k-length sequence,

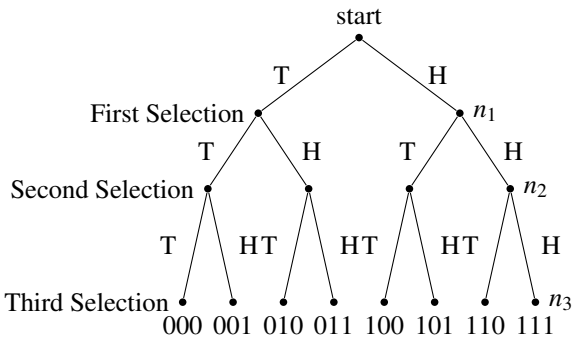
$$n_1 = n_2 = \cdots = n_k = n$$

Therefore, the total number of possible ordered k-length sequences from an n-element set S under sample with replacement is n^k.

10.2.1 Here is an Example

We may visit coin tosses first. The total possible outcomes of tossing a coin $k = 3$ times can be portrayed by the following tree:

Figure 10.2.1. The Tree of Counting on Coins



In the above graph, each vertex corresponds to a possible result, where each bit stands for whether the toss resulted in a head (1) or a tail (0).

10.2.2 But Sometimes, Order Doesn't Matter

Suppose we would like to select 5 cards from a set of three cards $S = \{1,2,3\}$, where ordering doesn't matter.
In this context, Second Rule of Counting does not help, because we do not have a m-to-1 relationship between ordered sequences and distinct sets. We would need to apply a different perspective. Let us discuss below:

Let us first generalize back to the original setting. We have a set $S = \{1,2,\cdots,n\}$ and would like to know the number of ways to choose multisets (sets with repetition) of size k.

Assume we have one bin for each element of S: providing n bins in total. We then count the number of ways to fill these bins with k elements, without minding the order of the bins themselves but just how many of the k elements they each contain.
Each allocation to the bin can be written as a bit-string, where 1 represents the change of bin and 0 represents the existence of one element.
For example, provided 5 bins and 5 elements where the leftmost bin has 1 element, the middle bin has 3, adn the last has 1, the representation is

$$011000110$$

If there exists a space between the 1s, it would imply that the bin contains no elements.
The length of our binary string, provided n elements and k bins, is $n+k-1$. We now choose which n locations should contain 0s in this arbitrary zeros, or alternatively, which of the remaining $k-1$ containing 1s.
A position corresponds to one specific value: once it is used, you may not reuse the position for another value and the position will be removed from the set of all possible options. The ordering also does not matter.
Therefore, we are essentially sampling k locations from $k+n-1$ without replacement where order does not matter, which provides the total number of multisets as:

Theorem 10.2.1. Balls and Urns

Generally, the number of ways to place n balls into k bins is:

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

Let us also describe the intuition we have used in generating the balls and urns approach in producing a match between bitstrings and ball-urn allocations:

Theorem 10.2.2. Zeroth Rule of Counting

If a set A can be placed into a one-to-one correspondence with a set B (in other words, find a bijection $f : A \to B$), then $|A| = |B|$.

10.3 Combinatorial Proofs

In short, proofs on combinatorials. They are mostly identity proofs. Let us explore a few examples/categories from below sections:

10.3.1 Binomial Theorem

Theorem 10.3.1. Binomial Theorem

$$\forall n \in \mathbb{N}((a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k})$$

The LHS is a product of n terms $(a+b)$.
Let these factors be labeled sets $f_1, \ldots, f_n$ for generality.
For multiplying factors f_1, f_2, a total of 4 two-term combinations is made via matching each term of f_1 to each term of f_2:

$$f_1 \times f_2 = \{(a_{f_1}, a_{f_2}), (a_{f_1}, b_{f_2}), (b_{f_1}, a_{f_2}), (b_{f_1}, b_{f_2})\}$$

$$(a+b)^2 = a^2 + ab + ba + b^2$$

Every term in such polynomial is in some form $a^k b^{n-k}$. It would be reasonable to see how this term contributes to the over sums of polynomial.

We produce terms $a^k b^{n-k}$ when k copies of a and n copies of b are multiplied, and there are $\binom{n}{k}$ ways to form such terms.
Therefore, for each index k possible (which spans from 0 to n, standing for the number of a apparent in the monomial), the sum of such monomials is the product of its value and number of such pairing:

$$\binom{n}{k} a^k b^{n-k}$$

Sum this product over all possible k, we will get the value of $(a+b)^n$, which is the sum of all possible monomials (products) achieved:

$$\sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$

The Binomial Theorem would then contribute to an interesting identity. Let us set $a = 1, b = -1$ in binomial theorem. Then:

$$\begin{aligned}(-1+1)^n &= \sum_{k=0}^{n} \binom{n}{k} (-1)^k (1)^{n-k} \\ &= \sum_{k=0}^{n} \binom{n}{k} (-1)^k = 0\end{aligned}$$

Zamn! Did you just use binomial theorem to prove that $-1+1=0$ to the n^{th} power is still 0?
Yeah, we did. But the last line of the proof, which is the combinatorial identity we have produced, is arugably more valuable.

10.3.2 Combinatorial Identities

Here are two interesting combinatorial identities to capitalize off in several proofs.

For now, let's talk about the first of two:

Definition 10.3.1. Hockey-Stick Identity

The Hockey-Stick Identity states that:

$$\binom{n}{k+1} = \sum_{i=n-1}^{k} \binom{i}{k}$$

The LHS of this identity is the number of ways to choose a $(k+1)$-length subset from an n-element set S.
For generality, let us denote S such that:

$$S = \{S_1, \cdots, S_n\}, \;\; S_1 < \cdots < S_n$$

If we let the lowest-value element of the subset to be S_1, then we will have k elements to choose from the rest of $n-1$ elements from S. Therefore, it contributes $\binom{n-1}{k}$ to the total number of combinations.
If we let the lowest-value element of the subset to be S_i, then we will have k elements to choose from the rest of $n-i$ elements from S (because i elements of S are smaller than or equal to the smallest element of subset). Therefore, it contributes $\binom{n-i}{k}$ to the total number of combinations.
This sum will continue until we let the lowest-value element of subset to be S_{n-k}, where we would be left with $n-(n-k)=k$ elements to choose from: the bare minimum.
Totaling all above contribution leads to the Hockey-Stick Identity:

$$\binom{n}{k+1} = \sum_{i=n-1}^{k} \binom{i}{k}$$

The second identity I'd like to introduce to you is:

Definition 10.3.2. Sum of combinations from n items

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

This identity can be proved via setting $a = 1, b = 1$ from the binomial theorem:

$$\begin{aligned} 2^n &= (1+1)^n \\ &= \sum_{k=0}^{n} \binom{n}{k} 1^k 1^{n-k} \\ &= \sum_{k=0}^{n} \binom{n}{k} 1^k = \sum_{k=0}^{n} \binom{n}{k} \end{aligned}$$

To discuss this via interpretation, suppose we have a set S with n distinct elements.
The LHS of this identity counts the number of ways of choosing an i-lengthed subset.
The RHS, meanwhile, counts the total number of possible subsets (see the Chapter for Set Theory in this document).
Therefore, the LHS and RHS of this identity indeed denote the same value!

10.3.3 Permutation and Derangements

In combinatorics, we have a famous problem. Here it goes.
Assume that we collect the namecard of n people, and redistribute them into the n people. On average, how many people recieve their own namecard?

Let us look at this in a combinatorics perspective.
Label the namecards as $1, 2, \cdots, n$ and let π_i denote the namecard that is returned to the i^{th} student.
Then, the vector $\vec{\pi} = \begin{bmatrix} \pi_1 & \cdots & \pi_n \end{bmatrix}^T$: a permutation of the set $\{1, \cdots, n\}$. There are then $n!$ distinct permutations of $\{1, \ldots, n\}$ since it is not a multiset.
In this setting, the situation at which a person receives his own namecard is when $\pi_i = i$.
Therefore, the number of people who receive their own namecard is the amount of index at which $\pi_i = i$. These points are called "fixed points".

A permutation with no fixed points is called a **derangement**. There is a formula for the number of derangements provided a set of n distinct elements:

Theorem 10.3.2. Number of Derangements

Theorem: For an arbitrary positive integer $n \geq 3$, the number D_n of derangements of $\{1, \ldots, n\}$ satisfies:

$$D_n = (n-1)(D_{n-1} + D_{n-2})$$

Proof:
In a derangement $(\pi_1, \cdots, \pi_n)$, suppose $\pi_n = j \in \{1, \cdots, n-1\}$. Then, there are $n-1$ choices for j. We also know that $j \neq n$ because this would then not produce a derangement.
Now, if $\pi_j = n$, then we recognize that j and n get swapped, and the number of possible derangements with the remaining $n-2$ numbers is D_{n-2}.
Or, if $\pi_j \neq n$, then n satisfies the same constraint as j (in not being able to be the value j). Therefore, we have $n-1$ values left to form derangements with, which provides D_{n-1} possibilities.
This completes the derivation of the above formula.

Notably, with the boundary conditions $D_1 = 0$ and $D_2 = 1$, we can then determine D_n computationally via recursion and memoization.

The derivation is not shown here, but the recursion would then yield that:

$$D_n = n! \sum_{k=0}^{n} \frac{(-1)^k}{k!}$$

10.4 The Principle of Inclusion-Exclusion

Overcounting is a big problem in combinatorics, but here is a technique to prevent most of such situations.
Let A_1 and A_2 be two subsets of the same finite set A and we want to count the number of elements in $A_1 \cup A_2$.
If A_1 and A_2 are disjoint, it is very lucky:

$$|A_1 \cup A_2| = |A_1| + |A_2|$$

If A_1 and A_2 are not disjoint, however, then we need to subtract the number of elements from their intersections:

$$|A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2|$$

This brings us to the generalization of this logic over an arbitrary number of subsets:

Theorem 10.4.1. Principle of Inclusion-Exclusion

Theorem: Let $A_1, \cdots, A_n$ be arbitrary subsets of the same finite set A:

$$|A_1 \cup \cdots \cup A_n| = \sum_{k=1}^{n} (-1)^{k-1} \sum_{S \subseteq \{1,\cdots,n\}:|S|=k} |\cap_{i \in S} A_i|$$

Proof:
Note that if a, an element, is not included by the union of all subsets of A considered in this scenario, then it is not possible for a to be counted anyways.
For a to be counted, it must belong to some subset A_i, such that $i \in \{1, \ldots, m\}$.
For the sake of clarifying the proof, define an index set $M \subseteq \{1, \ldots, n\}$ as:

$$M = \{i \in \{1, \ldots, n\} : \ a \in A_i\}$$

applying the prior logic, $a \in A_i$ iff $i \in M$. Here, if the subset S we find is not contained by M, then $a \notin \cap_{i \in S} A_i$.
Therefore, for a to be counted, $a \in \cap_{i \in S} A_i$ for any non=empty subset $S \subseteq M$. The number of subsets that have counted a on RHS is thus:

$$\sum_{k=1}^{n} (-1)^{k-1} \sum_{S \subseteq \{1,\cdots,n\}:|S|=k} 1 = \sum_{k=0}^{m} \binom{m}{k} (-1)^{k-1} = 1$$

And applying this to the derangement problem:

Example Question 10.4.1: Count Derangements

Let A_i denote the set of all permutations with i being a fixed point.
The number of permutations with at least one fixed point is $|A_1 \cup \cdots \cup A_n|$. Therefore,

$$D_n = n! - |A_1 \cup \cdots \cup A_n|$$

Meanwhile, the cardinality of A_i would be $(n-1)!$ because it just has to permute $n-1$ other elements in an arbitrary way.
The intersection of A_i and A_j would thus need to permute $n-2$ arbitrary elements, and has cardinality $(n-2)!$.
Let us follow the clue: for a subset $S \in \{1, \cdots, n\}$ such that $|S| = k$,

$$|\cap_{i \in S} A_i| = (n-k)!$$

The rest of algebra follows:

$$\begin{aligned}
|A_1 \cup \cdots \cup A_n| &= \sum_{k=1}^{n} (-1)^{k-1} \binom{n}{k} (n-k)! \\
&= \sum_{k=1}^{n} (-1)^{k-1} \frac{n!}{k!} \\
&= n! \sum_{k=1}^{n} (-1)^{k-1} \frac{1}{k!} \\
D_n &= n! - |A_1 \cup \cdots \cup A_n| \\
&= n!(1 - \sum_{k=1}^{n} (-1)^{k-1} \frac{1}{k!}) \\
&= n! \sum_{k=1}^{n} (-1)^{k} \frac{1}{k!}
\end{aligned}$$

Chapter 11

Countability

11.1 More on Bijections

Recall upon the Zeroth Rule of Counting:

> Two finite sets have the same size iff their elements can be paired up, where each element of one set has a unique partner in another.

This is oddly familiar. It is, in fact, formalized via the concept of bijection, which we will re-introduce here after its brief appearance when discussing modular arithmetics.

Definition 11.1.1. Bijections

Consider a function $f : A \to B$, which since it is a function, would satisfy that $\forall x \in A, f(x) \in B$.
Then f is a bijection if such relation holds true in a reverse direction. Therefore, for a bijection $f : A \to B$:

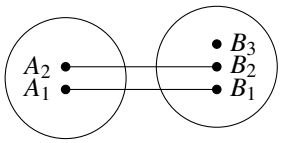
$$[\forall x \in A(f(x) \in B)] \wedge [\forall x \in B(f(x) \in A)]$$

In this case, as discussed before, f would satisfy two other significant properties simultaneously: injective and surjective.
Let me, for the sake of easing my own toil and reducing my masochistic tendency of typing 5 hours a day in LaTeX, just copy the description from Chapter 6:

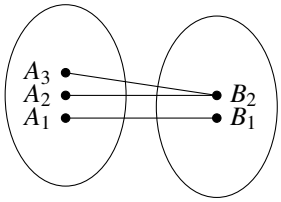
Definition 11.1.2. Injective, Surjective, Bijection

We call the A the set of *pre-images*, and B the set of *images*.

- **Injective**: Each image (output) has at most one pre-image (input). This type of function is thus called a "one-to-one" function.



- **Surjective**: Each image (output) has at least one image (input). This type of function is "onto".



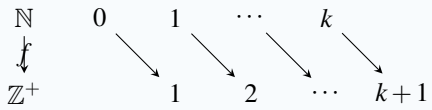
And once again, the inverse of a bijective function must also be bijective. While this is listed as an exercise to the reader, you may imagine bijections as relationship between sets A and B; then, notice that inverse functions of $f : A \to B$ are infact relationships $f^{-1} : B \to A$.
Since f^{-1} is the inverse of f, it carries the exact same pairings of f and must also be bijective as f is.

11.1.1 Cardinality and Bijection

Cardinality is the size of a set.
To show that two sets have the same cardinality, we can attempt to prove the existence of a bijection between them.
For example, let us consider a problem of two sets:

Example Question 11.1.1: The cardinality of nonnegative integers and positive integers

For this course, $0 \in \mathbb{N}$. Essentially, we ask if the cardinality of $\mathbb{N}$ is equal to that of $\mathbb{Z}^+$.
To prove the equality, let us attempt to form a bijection between these sets:



We can see that there is indeed a bijection $f\mathbb{N} \to \mathbb{Z}^+$, since every value of each set is paired with another value from the other set.
Meanwhile, we can notice that the relationship is one-to-one because no two natural numbers have the same image.

A very similar case can be made for $f : \mathbb{N} \to 2\mathbb{N}$, and can be found on the lecture note.

We are left with another interesting case where bijection can be certified, between $\mathbb{N}$ and $\mathbb{Z}$. Our initial thought here would have to do with the properties of both sets. For each pair of number $(-k,k)$ in $\mathbb{Z}$, where k is a natural number, we also need to take two respective numbers from $\mathbb{N}$ to match against them.
If so, then let us attempt with the function:

$$f(x) = \begin{cases} \frac{x}{2} & \text{if x is even} \\ -\frac{x+1}{2} & \text{if x is odd} \end{cases}$$

And let us see how this pairing works out:

$$\begin{array}{ccccccccc} \mathbb{N} & 0 & 1 & 2 & \cdots & 1000 & \cdots \\ f\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \mathbb{Z} & 0 & -1 & 1 & \cdots & 500 & \cdots \end{array}$$

Let us proceed with a proof of its bijection by demonstrating its injectiveness and surjectiveness:

Example Question 11.1.2: Prove the One-to-One and Onto of above function

One-to-One:
Suppose there exists two arbitrary values x, y such that $f(x) = f(y)$.
Then, both values must have the same sign, so either $f(x) = \frac{x}{2} = f(y) = \frac{y}{2}$ or $f(x) = -\frac{x+1}{2} = f(y) = -\frac{y+1}{2}$.
And in either case, we obtain that $x = y$, thus obtaining that:

$$f(x) = f(y) \implies x = y$$

Therefore, f is injective.
Onto:
Suppose that y is non-negative, then $f(2y) = y$. Suppose that y is negative, then $f(-(2y+1)) = y$ as well.
Since every image has a pre-image, f is onto.

Therefore, while $\mathbb{N}$ has the same size with $\mathbb{Z}^+$, it also has the same size with $\mathbb{Z}$. Interesting!

11.2 Countability

Now we come to an imporatant question. What does it mean for a set to be countable?

Definition 11.2.1. Countability

A set S is countable if it is finite, or if there is a bijection between S and $N \subseteq \mathbb{N}$.

Well, just knowing countability by definition would yield fun, but what would yield more fun? Explaining if $|\mathbb{Q}| = |\mathbb{N}|$!

Theorem 11.2.1. Proof that $|\mathbb{N}| = |\mathbb{Q}|$

To do so, let us first observe that $|\mathbb{N}| \leq |\mathbb{Q}|$, since we would see that every natural number is a rational number while the reverse does not necessarily hold true.

To continue off this proof, we now need to recognize if also $|\mathbb{Q}| \leq |\mathbb{N}|$. To do so, we will exhibit an injection $f : \mathbb{Q} \to \mathbb{N}$.
Now, let us consider a counterclockwise spiral that rotates its direction of travel 90 degrees counterclockwise. In such a figure, findable from the lecture notes (Around page 4), we observe that each rational number is represented by the point (x, y) on the path of such spiral, which is placed on the two-dimensional space $\mathbb{Z}^2$.
If we map each lattice point on the spiral to the numeric order of its precedence on the spiral, we then thus prove that every rational number (characterized as a lattice point, again) must correspond to one natural number.
Therefore, knowing that all points on the spiral must be presentable as a natural number: $|\mathbb{Q}| \leq |\mathbb{N}|$.

We will not go over the next few examples on the lecture note for now (and I expect this opinion to be revised around November's Thanksgiving break, where I drown with my ideals in mathematics), but please acknowledge such a technique of countability.
The set of all binary strings is countable. As for why, each binary strings has a representative number, and if we must consider the empty string, then we would assign it to the value 0 and assign other bit-strings to the number they represent in binary base added by 1.
This exact reasoning would suffice for ternary strings, quad-ary strings (is this a thing?), hexadecimal strings... the

list goes on, but this holds a significant implication. If we can manage to map mathematical objects, like polynomials and graphs (representable by adjacency matrices, for example), then sets of matheamtical objects can indeed also be proven as countable!
The lecture note proposes an example proving the set of all polynomials to be countable, via formatting a polynomial as a ternary string, where each coefficients and exponents are formatted in their binary bit-string form and separated by a character "2". The former logic has stated that the set of all ternary strings is countable, which would then, via convenience offered from the definition of countability, make the set of all polynomials be as countable as the set of all ternary strings are.
I promise to offer a much more thorough explanation of this as we go onto RRR week and revise the formatting of our lecture notes.

11.3 Cantor's Diagonalization

In this chapter, we deal with two great questions.
First of all, why am I denser than the set of rational numbers? You see, the set of rational numbers is dense, as between any two rational numbers, we can find another rational number. In fact, between any two real numbers there is always a rational number (which we used as an example in Chapter 2 when demonstrating proofs). It's Real Dense.
And second of all, is there a bijection between the rationals and the reals, provided the above fact of the density of set (that real numbers and rational numbers are all dense)? We will resolve this question via an argument from Cantor, known as "Cantorś diagonalization".
Before presenting the proof, I'd like to remind that any real number in the interval $[0,1]$ can be written as an infinite decimal with no trailing zeros, and rational numbers can be represented as recurring decimals, while irrational ones non-recurring decimals. Most importantly, they are all unique and infinitely long. For numbers like 0.5, they will be represented as $0.4\overline{9}$.

Theorem 11.3.1. Uncountability of Real Numbers

Theorem: The real interval $\mathbb{R}[0,1]$ is uncountable.

Proof: For the sake of contradiction, let us consider there is a bijection $f:\mathbb{N}\to\mathbb{R}[0,1]$. We may then enumerate real numbers in an infinite list $f(0),\cdots$ as follows:

$$\begin{aligned} f(0) &= 0.5214\cdots \\ f(1) &= 0.1416\cdots \\ f(2) &= 0.9478\cdots \\ f(3) &= 0.5309\cdots \\ &\vdots \end{aligned}$$

The basic rule of this construction is that for the number of leftup-rightdown diagonal (in this case $0.5479\cdots$) r, its $(i+1)^{th}$ digit after decimal point must be same with that of $f(i)$ for any $i\in\mathbb{N}$.
However, we can see that a value, such as r whose digit is incremented by 2 under modulo 10, cannot fit into this enumeration because of its $(i+1)^{th}$ digit not being equal to that corresponding digit of r.
Therefore, we see at least one item not counted in the "bijection" that should've existed between $\mathbb{N}$ and $\mathbb{R}[0,1]$, thus deeming $\mathbb{R}[0,1]$ uncountable.

The way this technique worked ultimately roots in how we defined a real number as an infinite decimal expansion that can be either recurring or non-recurring. Say today we attempt to apply this argument on $\mathbb{Q}$, which is a set of recurring decimal expansions, the proof wouldn't gain progress because we cannot confirm whether the number of diagonal is indeed a recurring decimal representation.

11.4 Cantor's Set

The Cantor's Set then argues about the mathematical properties of interval $\mathbb{R}[0,1]$. This set is defined by repeatedly removing the middle thirds of line segments infinitely many times.

Definition 11.4.1. Cantor's Set, Construction

A Cantor's Set can be constructed iteratively with infinite steps.
In each step, we remove the middle one-third of line segments we currently have. For example,

$$[0,1] \to [0,\frac{1}{3}] \cup [\frac{2}{3},1]$$

And then we proceed to remove the middle third of each line segments we deal with:

$$[0,\frac{1}{3}] \cup [\frac{2}{3},1] \to [0,\frac{1}{9}] \cup [\frac{2}{9},\frac{1}{3}] \cup [\frac{2}{3},\frac{7}{9}] \cup [\frac{8}{9},1]$$

You would think that the Cantor's Set faces an inevitable emptiness (just like most of human life), but it is merely that the measure of the set is zero, as in it now contains many almost trivial intervals with many isolated points across the interval and "looks mathematically like being empty". It is the realistic equivalent of looking at a white paper with many atomic black points, which we just cannot perceive with human eyes.
But the Cantor's Set is uncountable. This is proven by forming a bijection between the Cantor's Set and an uncountable set:

Theorem 11.4.1. The Uncoutnability of Cantor's Set

Theorem: The Cantor's Set is Uncountability.

Proof: Each element in the Cantor's Set can in fact be expressed as a ternary string, at last containing only those whose ternary representations have 0s and 2s (because those that involve 1s must have been in the middle third of some interval and thus deserves removal).
We will also proceed to write any finite decimal expressions into its infintie form (for example, $0.1 \to 0.\overline{2}$).
This leaves us with a set builder representation of the Cantor's Set:

$$\{C = \{x \in [0,1] : \text{x has a ternary representation of only 0s and 2s}\}\}$$

Which allows us to map the set C onto the interval $\mathbb{R}[0,1]$ via substituting every trit 2 with value 1 and forcing the ternary string into a binary presentation that corresponds to some value in $\mathbb{R}[0,1]$.
This way, we formed a bijection between C and an uncountable set, so using the Zeroth Rule of Counting, C the Cantor's Set is also uncountable.

11.5 Power Sets: Higher Order Infinity

Here is a review on Power Sets:

Definition 11.5.1. Power Set

The power set of S can be written as:

$$\mathbb{P}(S) = \{P | P \subseteq S\}$$

as one of its various denotations, the one on lecture note using a Weierstrass P, $\wp$.
The power set of a set is the set of all of its possible subsets.

In addition, if the cardinality of S in the above definition is $|S| = k$, then we can state that $|\mathbb{P}(S)| = 2^k$.

This can now be combinatorically understood: every power set is a k-bit string representation for the absence of each of k elements.
Now, let us discuss if the power set of $\mathbb{N}$ can be countable, and from there, reveal a thing or two about infinity.

Theorem 11.5.1. Uncountability of $\wp(\mathbb{N})$

For the sake of contradiction, let us assume there is a bijection $f : \mathbb{N} \to \wp(\mathbb{N})$ defined exactly like Cantor's Diagonalization Argument, where each row of a table describes a subset via a bit-string, and each column stands for whether an element of $\mathbb{N}$ is present or not.
Let us now present a modified version of the diagonal number (again, leftup-rightdown) that is the flipped version of diagonal (which means 0s and 1s are flipped). That digit then must not exist in the enumeration, therefore proving the bijection malfunctional, and moreover,

$$|\wp(\mathbb{N})| > |\mathbb{N}|$$

The cardinality of $\mathbb{N}$ is denoted in the symbol $\aleph_0$ (aleph-null), and using that symbol, $|\wp(\mathbb{N})| = 2^{\aleph_0}$.
Such cardinality of $\wp(\mathbb{N})$ is the same as the cardinality of $\mathbb{R}$, which is known as $\mathbf{c}$.
Even larger infinite cardinalities are denoted as $\aleph_k$ where $k > 0$, and they can be defined using set theory. They, interestingly, obey the rules of arithmetics, and in turn provide assistance towards famous mathematical hypotheses.

Chapter 12

Computability

12.1 The Liar's Paradox

Propositions are, once again, either true or false. For example, the proposition:

> Let A be the set of players in an Among Us game, then there exists one player in A such that the player is an impostor.

But here is a problematic proposition:

> An impostor says: "All impostors are liars."

Why do we have so many among us jokes? And also, how exactly is this statement probelmatic?
Well, if the statement is true, then all impostors are liars. However, since this sentence is said by an impostor, it is false, and all impostors are in fact not liars. But if impostors are not liars, then this statement is true.
The logical loop goes on due to the self-referencing nature of this sentence.

Suppose we view another case of this.

> In the village with just one barber, some men shave themselves or go to that barber.
> The barber proclaims: I shall shave all and only those men who do not shave themselves.
> Does the barber shave himself?

Does the barber shave himself?
If a barber's statement is true, then he shaves only those who do not shave themselves. He would shave himself if he does not shave himself. If he does not shave himself, however, he does shave himself.
Such is the wonder of liar's paradox: you cannot get any effective results from interpreting these statements.

12.2 Self Replicating Programs

Let us view self-reference as a computational concept: can we use self-reference to design a program that outputs itself?
Let us consider how we can do this if we write the program in psuedocode, where thus we get:

```
print the following sentence twice, the second time in quotes:
    "print the following sentence twice, the second time in quotes:"
```

If we execute this instruction above, then we will get the output be exactly what we executed. This self-replicating program is known as a quine (which you might have encountered in COMPSCI 61A, if lucky).
To construct such a quine in general, we can use:

Theorem 12.2.1. Recursion Theorem

Given any program $P(x,y)$, we can always convert it into another program $Q(x)$ such that $Q(x) = P(x,Q)$.
Q behaves exactly as P would if its second input is the decsription of program Q.
Therefore, Q has access to its own decsription, and is essentially a self-aware version of P.

12.3 The Halting Problem

Now we see self-reference working in computation (albeit still not in human logic), let's then come to ask: is there anything a computer cannot do?
Yes. In 1936, Alan Turing managed to show that there is no program that cannot answer for itself: "does your execution lead to an infinite loop?"
The proof has two ingredients: self-reference and how inseparable programs and data are to each other. Before inspecting the proof, let's inspect the problem more closely.
Let us write a program that behaves as follows:

$$\texttt{TestHalt(P,x)} = \begin{cases} \text{“}yes\text{”}, & if\ program\ P\ halts\ on\ input\ x \\ \text{“}no\text{”}, & if\ program\ P\ loops\ on\ input\ x \end{cases}$$

And now, let us access the proof:

Theorem 12.3.1. The Uncomputability of Halting Problem

Theorem: The Halting Problem is uncomputable.

Proof: For the sake of contradiction, let us construct the following program:

```
Turing (P)
    if TestHalt(P, P) is ``yes'' then loop forever
    else halt
```

Let us then look at the behavior of `Turing(Turing)`, which either halts or not.
If it halts, then it must be that `TestHalt(Turing, Turing)` returned "no". However, that means this program must have looped.
If it does not halt, then it loops on forever, due to it should've halted on the provided input. In both cases, we arrive at a contradiction, so the program `TestHalt` must've been wrong and impossible to be entirely right!

I know that you are curious about what proof technique we used for this. Well, it is once again a diagonalization argument, as every computer program is just finite-length strings over alphanumeric characters. Therefore, we can count the set of all programs.

Theorem 12.3.2. Continued, The Uncomputability of Halting Problem

Proof:
Let us then construct a table of program enumeration vs program enumeration (P_i). In this case, for the $(i,j)^{th}$ entry in the table is H if P_i halts on P_j (halts when P_j is an input), and L if otherwise (which means it loops).
Suppose the program $P_n = Turing$ exists, but it cannot be enumerated due to a proof by case:

If the (n,n) entry is H, then P_n halts on P_n and `Turing` will thus loop forever.
If the (n,n) entry is L, then `Turing` halts on P_n instead, but contradicts the original assumption.

Since `Turing` is a simple modification of `TestHalt`, if `Turing` does not get enumerated, neither can `TestHalt`.

Therefore, the Halting problem is unsolvable.

12.3.1 Easy Halting Problem

Brandon. Who in the right mind would feed a program back to the original program?
To that answer, I respond *"computer scientists"*, which you are one, for the record of this fact.
But, computer scientsts in turn proposed a new, more practical problem according to your concerns, complaints, insights:

Theorem 12.3.3. The Uncomputability of Halting Problem

Theorem: There does not exist a program that can decide whether any given program P will halt on 0.

Proof: Let there be a program `TestEasyHalt` that resolves this problem (for the sake of contradiction):

$$\texttt{TestHalt(P,x)} = \begin{cases} \text{“}yes\text{”}, & if\ program P halts on input 0 \\ \text{“}no\text{”}, & if\ program P loops on input 0 \end{cases}$$

Then we can use this as a subroutine to build the following algorithm:

```
Halt(P, x)
    construct P' such that when inputted 0, it returns P(x)
    return TestEasyHalt(P')
```

Well, $P'(0)$ would only halt iff $P(x)$ does, so if such a program `TestEasyHalt` exists, then `Halt` effectively solves the Halting Problem, which is impossible to solve!
Therefore, there cannot be such a program `Halt`, and thus cannot be a program `TestEasyHalt`.

This proof utilizes a technique called "reduction", where we reduced a problem into another problem in the hope that the smaller problem leads to the solution of the larger problem. You have done this more or less when coding, writing mathematics, or in general just coping about the thoughts of programming in your head.
But, this implies that the second problem is as difficult as the first, since you need the second problem's solution to resolve the first problem.

12.4 Godel's Incompleteness Theorem

In 1900, David Hilbert posed two questions about the foundation of logic in mathematics:

- Is arithmetic consistent?
- Is arithmetic complete?

In return, humanity now has an existential crisis. Let me elaborate.
To understand the problems he posted, we should remember that mathematics is a formal system built on axioms and rules of inferences, hence all the theorems we have relied on.
Axioms are the cornerstones of mathematics.

So, the first question above asks whether it is possible to prove a proposition P and its negation $\neg P$ (which we attempted to not do in the Law of Excluded Middle). The case in which this is proven will cause arithmetic to be inconsistent; otherwise, consistent.
But, if arithmetics is inconsistent, many currently wrong statements can later be proven right.
The second question asks whether every true statement in arithmetic can be mathematically proved.

Let's talk about history. There was indeed an unproven theorem (Fermat's Last Theorem) until around 1990s.
Meanwhile, as these questions were proposed in 1928 as "Entscheidungsproblem", which we hope the answer is 'yes' to, they were immediately proven "no" by Godel in 1930.
This proof exploited the deep connection between proofs and arithmetics, as well as proofs and computation. We will, in the following sections, discuss the sketch and essence of Godel's Proof.

12.4.1 Sketch of Godel's Proof

Suppose we have a formal system F and assume it is sufficiently expressive that we can use it to express all of arithmetic. Suppose we can write the following statement:

S(F) = "This statement is not provable in system F."

Two things can happen:

- Case 1: $S(F)$ is provable. Then the statement is true, but by inspecting the content of the statement itself, we attempt to see that this implies $S(F)$ is not provable. This makes F inconsistent.
- Case 2: $S(F)$ is not provable, which means $S(F)$ is true, and F is therefore already incomplete.

We would then need to construct such a statement $S(F)$, which is the reduction of the larger problem for proving mathematics wrong. Once again, that is as difficult as the original larger problem.

12.4.2 Proof via Halting Problem

Suppose arithmetic is both consistent and complete, we will now use this statement to perform a proof by contradiction.
Let the proposition $S_{P,x}$ depict "Program P halts on input x", and we can thus phrase it as an arithmetic:

$$\exists z(\text{z encodes a valid halting exception sequence of P on input x})$$

Now let us assume that arithmetic is both consistent and complete. This means the above statement, which represents the halting problem, can be proven either true or false due to the completeness of arithmetics.
However, this then in turn implies the halting problem is prove-able via searching through every single possible proof enumerable (which is countably many). The problem, however, is that the Halting Problem is uncomputable, not allowing such a program for searching proofs to exist.
Therefore, the initial assumption is false.
Henceforth, arithmetics is inconsistent and incomplete. The mathematics you have learned along the way is potentially all wrong.