COMPSCI 70 Self-Study Lecture Notes

by 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**. This allows us to abbreviate a set of objects. The format is as follows:

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:

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

For example, the list of all rational numbers can be written as:

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

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". This concept corresponds to 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:

- $A \backslash \emptyset = A$
- $\emptyset \backslash A = \emptyset$

And the last arithmetic is the "multiplication of sets":

Definition 0.1.7. Cartesian Products

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

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

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: $\nexists$

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

In this section, we discuss proposition, a fundamental mathematical object for logical statements in the mathematical language. Particularly, we discuss its definition, fundamental properties, and essential operations to create propositions out of proposition(s).

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.

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

We may join propositions together to form more complex propositions. There are approximately four methods of using preexistent propositions to make new propositions, 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, $P \vee \not P$ always holds true, because P is either true or false (not true).

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

Figure 1.1.2. Truth Table for Implication

A truth table for implications is as follows:

P	Q	$P \implies Q$	$\not P \vee Q$
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. Meanwhile, since $P \implies Q$ and $\not P \vee Q$ 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. However, sometimes we 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. Meanwhile, there can be implications that are based on A yet doesn't necessarily hold true when A does.
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 is known as "proof by contrapositive", which may be introduced in later assignments and notes.

1.2 Quantifiers

In this section, we discuss advanced uses and interpretations of quantifiers, as well as exploring the general syntax of propositions that involve 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. Then, syntactically, the quantifier serves to involve this range of quantification for our proposition.
Interestingly, in a finite universe, there is also another way of conceptualizing the quantifiers we know:

Think Brandon! Think! 1.2.1: Is there another way to think of quantifiers? :hmm:

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

In this section, we discuss the variety of usages for negation, as well as the interpretation of complicated propositions involving negations and quantifiers.

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). 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 more interesting about propositions. 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.

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.
Without further ado, let us start.

2.1 Introduction to Proofs

This section provides an introduction to the definition and usage of proof, as well as some preliminary knowledge and tools for constructing mathematical proofs. We will introduce the necessary symbols used in mathematical language for proofs as well.

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 example, 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.
Having introduced the form and foundation of proof, let us also observe the basic notations we speak in mathematical language to present a proof with.

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 ($b = aq$)
$a := b$	Definition: number a is defined to have value b

2.2 Proof Technique

This section discusses the various cursed techniques for proofs, each accompanied with some example and line of thought.
Since proofs are not necessarily algorithmic processes *yet*, the demonstrations of proofs attached per subsections could help organize relevant logistics for practical uses and tips of each 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 style of attack from a Direct Proof user is to prove the legitimacy of $Q(x)$ via assuming $P(x)$.
To be more specific, 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. Well, 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 use of the Direct Proof Technique, let us consider the following prompt:

Think Brandon! Think! 2.2.1: Show me an 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:

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

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.
Let us demonstrate now:

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:

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

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

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

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:

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

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!

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: $p_1 < p_2 < \cdots < p_k$.
Let us then have a number:

$$q := p_1 p_2 \dots p_k + 1$$

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.
However, 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.
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!

In the above proof, we were able to find a contradiction from the negation of the prompt. This process disproves the **negation** of the prompt, or in other words, proves such proposition false.
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:

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

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:

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

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

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:

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

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.
On a broader horizon, every proof has some techniques to prove by, and some are to be more efficient than others. 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.

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

In this section, we introduce mathematical induction as a tool of proof, and discuss the usage of mathematical induction via examples.

3.1.1 Definition of Mathematical Induction

Previously, we introduced "techniques", or 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. 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? Each mathematical induction is composed of three fundamental aspects: **Base case**, **Induction Hypothesis**, and **Inductive Step**:

- **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.

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 Powerful

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.

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 instututions, 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:

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

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:

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

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:

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.

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

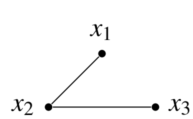
The mathematical theory of properties and applications of Graphs.

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:

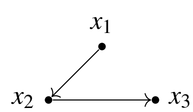
```
\begin{tikzpicture}
    [point/.style = {circle, fill, inner sep = 1pt}]
    \node[point] (1) [label=above:$x_1$] {};
    \node[point] (2) [below left of=1, label=left:$x_2$] {};
    \node[point] (3) [below right of=1, label=right:$x_3$] {};
    \draw (1) -- (2);
    \draw (2) -- (3);
\end{tikzpicture}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

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?

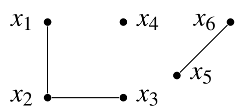
Definition 5.1.6. Connectivity

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:

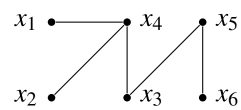
Figure 5.1.4. Connected vs Disconnected

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.

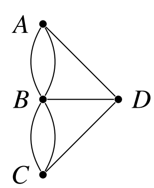
5.2 Koenigsberg's Seven Bridges and Eulerian Tour

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.
And don't worry. We will talk more about trees in the next section.

5.3.2 Planar Graphs

What is a planar graph?

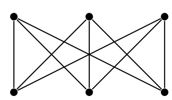
Definition 5.3.1. Planarity

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 $E = L \times R$.

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.

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

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.

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

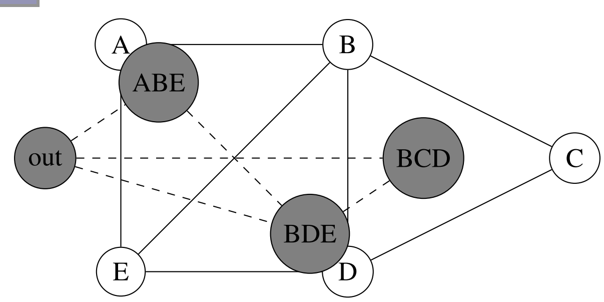
The Greeks knew everything, from the section above to the next notion:

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.
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.

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.

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 <u>at least</u> $2^k \geq |S|$ as desired.
(The $|S_0| - |S_1|$ was derived from the cardinality of complement from $\{x : 0x \in S_0\} - \{x : 1x \in S_1\}$).

Chapter 6

Modular Arithmetics

Chapter Description.

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, $\pmod{}$, 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 $0 \leq r \leq m-1 \land r \in Z$.
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:

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

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

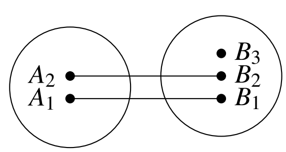
Take this as a side track!
A function is, in its nature, the relationship of sets: it is a relationship of elements from set A to set B. Mathematically expressed,

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

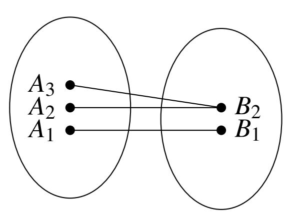
Definition 6.3.1. Injective, Surjective, Bijection

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).

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 $xy \equiv 1 \pmod{m}$, $\underline{y \text{ is the multiplicative inverse of } x \text{ modulo } m}$.

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.

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 $r = x - qy = x \pmod{y}$.
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.

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

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: $8x \equiv 9 \mod 15$

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