Discrete Math for Computer Science

Peter Schaefer

Freshman - Kutztown

Contents

1 Logic

1.1 Propositions and Logical Operations

Proposition: a statement that is either true or false.

Some examples include "It is raining today" and "$3 \cdot 8 = 20$".

However, not all statements are propositions, such as "open the door"

Name	**Symbol**	**alternate name**
NOT	$\neg$	negation
AND	$\wedge$	conjunction
OR	$\vee$	disjunction
XOR	$\oplus$	exclusive or

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \oplus q$
T	T	F	T	T	F
T	F	F	F	T	T
F	T	T	F	T	T
F	F	T	F	F	F

XOR is very useful for encryption and binary arithmetic.

1.2 Evaluating Compound Propositions

p : The weather is bad. $p \wedge q$: The weather is bad *and* the trip is cancelled

q : The trip is cancelled. $p \vee q$: The weather is bad *or* the trip is cancelled

r : The trip is delayed. $p \wedge (q \oplus r)$: The weather is bad *and* either the trip is cancelled *or* delayed

Order of Evaluation $\neg$, then $\wedge$, then $\vee$, but parenthesis always help for clarity.

Example Truth Table:

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

1.3 Conditional Statements

$p \to q$ where p is the hypothesis and q is the conclusion

Format	Terminology
$p \to q$	given
$\neg q \to \neg p$	contrapositive
$q \to p$	converse
$\neg p \to \neg q$	inverse

given	$p \to q$	$\equiv$	$\neg q \to \neg p$	contrapositive
inverse	$\neg p \to \neg q$	$\equiv$	$q \to p$	converse

p	q	$p \to q$
T	T	T
T	F	F
F	T	T
F	F	T

p is a sufficient condition for q

q is a necessary condition for p

Phrase	Logic
q if p	$p \to q$
q only if p	$q \to p$
q if and only if p	$p \iff q$

Order of Operations: $p \wedge q \to r \equiv (p \wedge q) \to r$

1.4 Logical Equivalence

Tautology: a proposition that is always true **Contradiction**: a proposition that is always false

Logically equivalent: same truth value regardless of the truth values of their individual propositions

DeMorgan's Laws:
$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$
$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

Verbally,
It is not true that the patient has migraines *or* high blood pressure ≡
≡ The patient does not have migraines *and* does not have high blood pressure

It is not true that the patient has migraines *and* high blood pressure ≡
≡ The patient does not have migraines *or* does not have high blood pressure

1.5 Laws of Propositional Logic

You can use **substitution** on logically equivalent propositions.

Law Name	∨ or	∧ and
Idempotent	$p \vee p \equiv p$	$p \wedge p \equiv p$
Associative	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Commutative	$p \vee q \equiv q \vee p$	$p \wedge q \equiv q \wedge p$
Distributive	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Identity	$p \vee \mathrm{F} \equiv p$	$p \wedge \mathrm{T} \equiv p$
Domination	$p \vee \mathrm{T} \equiv \mathrm{T}$	$p \wedge \mathrm{F} \equiv \mathrm{F}$
Double Negation	$\neg\neg p \equiv p$	
Complement	$p \vee \neg p \equiv \mathrm{T}$	$p \wedge \neg p \equiv \mathrm{F}$
DeMorgan	$\neg(p \vee q) \equiv \neg p \wedge \neg q$	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
Absorption	$p \vee (p \wedge q) \equiv p$	$p \wedge (p \vee q) \equiv p$
Conditional	$p \to q \equiv \neg p \vee q$	$p \iff q \equiv (p \to q) \wedge (q \to p)$

1.6 Predicates and Quantifiers

Predicate: a logical statement where truth value is a function of a variable.

P(x): x is an even number. P(5): false P(2): true

Domain: the set of all possible values for a variable in a predicate.

Ex. $\mathbb{Z}^+$ is the set of all positive integers.
*If domain is not clear from context, it should be given as part of the definition of the predicate.

Quantifier: converts a predicate to a proposition.

Quantifier	Symbol	Meaning
Universal	$\forall$	"for all"
Existential	$\exists$	"there exists"

$\exists\, x(x + 1 < x)$ is false.

Counter Example: universally quantified statement where an element in the domain for which the predicate is false. Useful to prove a $\forall$ statement false.

1.7 Quantified Statements

Consider the two following two predicates:

$$P(x) : x \text{ is prime, } x \in \mathbb{Z}^+$$
$$O(x) : x \text{ is odd}$$

Proposition made of predicates: $\exists\, x(\mathrm{P}(x) \wedge \neg \mathrm{O}(x))$
Verbally: there exists a positive integer that is prime but is not odd.

Free Variable: a variable that is free to be any value in the domain.
Bound Variable: a variable that is bound to a quantifier.

P(x): x came to the party
S(x): x was sick

$\mathrm{P}(x) \stackrel{?}{\equiv} \neg \mathrm{S}(x)$
$\mathrm{P}(x) \not\equiv \neg \mathrm{S}(x)$

	P(x)	S(x)	$\neg$S(x)
Joe	T	F	T
Theo	F	T	F
Gert	T	F	T
Sam	F	F	T

1.8 DeMorgan's law for Quantified Statements

Consider the predicate: $\mathrm{F}(x)$: "x can fly", where x is a bird. According to the DeMorgan Identity for Quantified Statements,

$$\neg\forall\ x\mathrm{F}(x) \equiv \exists\ x\neg\mathrm{F}(x)$$
"not every bird can fly $\equiv$ "there exists a bird that cannot fly

Example using DeMorgan Identities:

$$\begin{aligned}\neg\exists\ x(\mathrm{P}(x) \to \neg\mathrm{Q}(x)) &\equiv \forall\ x\neg(\mathrm{P}(x) \to \neg\mathrm{Q}(x)) \\ &\equiv \forall\ x(\neg\neg\mathrm{P}(x) \wedge \neg\neg\mathrm{Q}(x)) \\ &\equiv \forall\ x(\mathrm{P}(x) \wedge \mathrm{Q}(x))\end{aligned}$$

1.9 Nested Quantifiers

A logical expression with more than one quantifier that binds different variables in the same predicate is said to have **Nested Quantifiers**.

Logic	Variable Boundedness
$\forall\ x\exists\ y\ \mathrm{P}(x,y)$	x, y bound
$\forall\ x\ \mathrm{P}(x,y)$	x bound, y free
$\exists\ x\exists\ y\ \mathrm{T}(x,y,z)$	x, y bound, z free

Logic	Meaning
$\forall\ x\forall\ y\ \mathrm{M}(x,y)$	"everyone sent an email to everyone"
$\forall\ x\exists\ y\ \mathrm{M}(x,y)$	"everyone sent an email to someone"
$\exists\ x\forall\ y\ \mathrm{M}(x,y)$	"someone sent an email to everyone"
$\exists\ x\exists\ y\ \mathrm{M}(x,y)$	"someone sent an email to someone"

There is a two-player game analogy for how quantifiers work:

Player	Action	Goal
Existential Player $\exists$	selects value for existentially-bound variables	tries to make expression <u>true</u>
Universal Player $\forall$	selects value for universally-bound variables	tries to make expression <u>false</u>

Consider the predicate $\mathrm{L}(x,y)$: "x likes y".

$\exists\ x\forall\ y\mathrm{L}(x,y)$ means "there is a student who likes everyone in the school".
$\neg\exists\ x\forall\ y\mathrm{L}(x,y)$ means "there is no student who likes everyone in the school".

After applying DeMorgan's Laws,

$\forall\ x\exists\ y\neg\mathrm{L}(x,y)$ means "there is no student who likes everyone in the school".

1.10 More Nested Quantifiers

$\mathrm{M}(x,y)$: "x sent an email to y". Consider $\forall\ x\forall\ y\ \mathrm{M}\ (x,y)$. It means that "email sent an email to everyone including themselves". Using $(x \neq y \to\ \mathrm{M}(x,y))$ can fix this quirk.

$\forall\ x\forall\ y(x \neq y \to\ \mathrm{M}(x,y))$ means "everyone sent an email to everyone else

Expressing Uniqueness in Quantified Statements

Consider $\mathrm{L}(x)$: x was late to the meeting. If someone was late to the meeting, how could you express that that someone was the only person late to the meeting? You want to express that there is someone where everyone else was not late, which can be done with

$$\exists\ x(\mathrm{L}(x) \wedge \forall\ y(x \neq y \to \neg\mathrm{L}(y)))$$

Moving Quantifiers in Logical Statements

Consider M(x, y): "x is married to y" and A(x): "x is an adult". One way of expressing "For every person x, if x is an adult, then there is a person y to whom x is married to" is by this statement:

$$\forall\, x(\ \mathrm{A}(x) \to \exists\ \ \mathrm{M}(x, y))$$

Since y does not appear in A(x), "$\exists\, y$" can be moved so that it appears just after the "$\forall$ ", resulting with

$$\forall\, x \exists\, y(\ \mathrm{A}(x) \to\ \ \mathrm{M}(x, y))$$

When doing this, keep in mind that $\forall\, x \exists\ y \not\equiv \exists\ y \forall\ x$:

$\forall\, x \exists\, y(\ \mathrm{A}(x) \to\ \ \mathrm{M}(x, y))$ means
for every x, if x is an adult, there exists y who is married to x.
$\exists\, y \forall\, x(\ \mathrm{A}(x) \to\ \ \mathrm{M}(x, y))$ means
There exists a y, such that every x who is an adult is also married to y

1.11 Logical Reasoning

Argument: a sequence of propositions, called hypothesis, followed by a final proposition, called the conclusion.

An argument is **valid** if the conclusion is true whenever the hypothesis are all true, otherwise the argument is **invalid**.

An argument is denoted as: $\begin{array}{l} p_1 \\ p_2 \\ \vdots \\ \underline{p_n} \\ \therefore c \end{array}$ where $\begin{array}{l} p_1, p_2, \ldots, p_n \text{ are hypothesis} \\ c \text{ is the conclusion} \end{array}$

The argument is valid whenever the proposition $(p_1 \wedge p_1 \wedge \cdots \wedge p_n) \to c$ is a tautology. Additionally, because of the commutative law, hypothesis can be reordered without changing the argument.

$$\begin{array}{l} p \\ \underline{p \to q} \\ \therefore q \end{array} \equiv \begin{array}{l} p \to q \\ \underline{p} \\ \therefore q \end{array}$$

The Form of an Argument

$$\begin{array}{l} \text{It is raining today.} \\ \underline{\text{If it is raining today, then I will not ride my bike to school.}} \\ \therefore \text{I will not ride my bike to school.} \end{array} \qquad \begin{array}{l} p \\ \underline{p \to q} \\ \therefore q \end{array}$$

The argument is valid because its form, $\begin{array}{l} p \\ \underline{p \to q} \\ \therefore q \end{array}$ is an valid argument.

$$\begin{array}{l} \text{5 is not an even number.} \\ \underline{\text{If 5 is an even number, then 7 is an even number.}} \\ \therefore \text{7 is not an even number.} \end{array} \qquad \begin{array}{l} p \\ \underline{p \to q} \\ \therefore q \end{array}$$

The argument is invalid because its form, $\begin{array}{l} \neg p \\ \underline{p \to q} \\ \therefore \neg q \end{array}$ is an invalid argument.

1.12 Rules of Inference with Propositions

Using truth tables to establish the validity of an argument can become tedious, especially if an argument uses a large number of variables.

p $p \to q$ $\therefore q$	Modus Ponens	p q $\therefore p \wedge q$	Conjunction
$\neg q$ $p \to q$ $\therefore \neg p$	Modus Tollens	$p \to q$ $q \to r$ $\therefore p \to r$	Hypothetical Syllogism
p $\therefore p \vee q$	Addition	$p \vee q$ $\neg p$ $\therefore q$	Disjunctive Syllogism
$p \wedge q$ $\therefore p$	Simplification	$p \to q$ $q \to r$ $\therefore q \vee r$	Resolution

Example expressed in English:

If it is raining or windy or both, the game will be cancelled. The game will not be cancelled $\therefore$ It is not windy.	$(r \vee w) \to c$ $\neg c$ $\therefore \neg w$

Steps to Solve:

$(r \vee w) \to c$	Hypothesis	(1)
$\neg c$	Hypothesis	(2)
$\neg(r \vee w)$	Modus Tollens: 1, 2	(3)
$\neg r \wedge \neg w$	DeMorgan's Law: 3	(4)
$\neg w \wedge \neg r$	Commutative Law: 4	(5)
$\neg w$	Simplification: 5	(6)

1.13 Rules of Inference with Quantifiers

In order to apply the rules of quantified expressions, such as $\forall\, x \neg(\, \mathrm{P}(x) \wedge\, \mathrm{Q}(x))$, we need to remove the quantifier by plugging in a value from the domain to replace the variable x.

For example:

Every employee who received a large bonus works hard. Linda is an employee at the company. Linda received a large bonus. $\therefore$ Some employee works hard.	$\forall\, x(\mathrm{B}(x) \to\, \mathrm{H}(x))$ Linda $\in x$ B(Linda) $\therefore \exists\, x\ \mathrm{H}(x)$

Arbitrary Element: has no special properties other than those shared by all elements of the domain.

Particular Element: may have special properties that are not shared by all the elements of the domain. For example, if the domain is the set of all integers, $\mathbb{Z}$, a particular element is 3, because it is odd, which is not true for all integers.

Rules of Inference for Quantified Statements

c is an element $\forall\, x$ P (x) $\therefore$ P(c)	Universal Instantiation	$\exists\, x$ P(x) $\therefore\, c$ is particular $\wedge$ P (c)	Existential Instantiation*
c is arbitrary P (c) $\therefore \forall$ P(x)	Universal Generalization	c is an element P(c) $\therefore \exists\, x$ P(x)	Existential Generalization

*Each use of Existential Instantiation must define a new element with its own symbol or name.

Example of using the Laws of Inference for Quantified Statements

Consider the following argument:

$\forall\, x(\mathrm{P}(x) \vee \mathrm{Q}(x))$
3 is an integer
$\neg\, \mathrm{P}(3)$
$\therefore\ \mathrm{Q}(3)$

Steps to Solve:

$\forall\, x(\mathrm{P}(x) \vee \mathrm{Q}(x))$	Hypothesis	(1)
3 is an integer	Hypothesis	(2)
$(\mathrm{P}(3) \vee \mathrm{Q}(3))$	Universal Instantiation: 1, 2	(3)
$\neg\, \mathrm{P}(3)$	Hypothesis	(4)
$\mathrm{Q}(3)$	Disjunctive Syllogism: 3, 4	(5)

Showing an Argument with Quantified Statements is Invalid

Consider the following argument:

$\exists\, x\mathrm{P}(x)$
$\exists\, x\mathrm{Q}(x)$
$\therefore \exists\, x(\mathrm{P}(x) \wedge \mathrm{Q}(x))$

Using a supposed domain $\{c, d\}$, with truth values of

	P	Q
c	T	F
d	F	T

, the argument is invalid.

2 Proofs

2.1 Mathematical Definitions

- An integer x is *even* if there is an integer k such that $x = 2k$
- An integer x is *odd* if there is an integer k such that $x = 2k + 1$

Parity

The parity of a number is whether the number is odd or even.

- If 2 numbers are both even or both odd, they have the *same parity.*
- If 1 number is even and 1 number is odd, they have the *opposite parity.*

Rational

A number r is rational if there exists x and y such that

$$r = \frac{x}{y},\ y \neq 0$$

where the choice of x and y are not necessarily unique.

Divides

An integer x **divides** an integer y if and only if $x \neq 0$ and $y = kx$, for some integer k. x divides y is denoted as $x \mid y$. If x does not divide y, it is denoted as $x \nmid y$. If $x \mid y$, then y is said to be a multiple of x, and x is a **factor** or *divisor* of y.

Primality

An integer n is **prime** if and only if $n > 1$ and the only positive integers that divide n and 1 and n.

Composite

An integer n is **composite** if and only if $n > 1$ and there is an integer m such that $1 < m < n$ and $m \mid n$.

Properties of greater than and less than

If x and c are real numbers, then exactly 1 of the following is true:

$$x < c \quad x = c \quad x > c \qquad \begin{array}{c} \neg(x < c) \Leftrightarrow x \geq c \\ \neg(x > c) \Leftrightarrow x \leq c \\ \hline x > c \implies x \geq c \\ x < c \implies x \leq c \end{array}$$

2.2 Introduction to Proofs

A **theorem** is a statement that can be proven to be true.

A **proof** consists of a series of steps, each of which follows logically from assumptions, or from previously proven statements, whose final step should result in the statement of the **theorem** being proven.

An **axiom** is a statement assumed to be true.

Example Theorem

Every positive integer is than or equal to its square.

Proof. let x be an integer, where $x > 0$.
Since x is an integer and $x > 0$, then $x \geq 1$.
Since $x > 0$, we can multiple both sides of the inequality by x to get

$$(x \cdot x \geq 1 \cdot x) = (x^2 \geq x)$$

□

Proof by Exhaustion

if $n \in \{-1, 0, 1\}$, *then* $n^2 = |n|$

Proof.

$$\begin{aligned} n &= -1 & (-1)^2 &= 1 = |-1| \\ n &= 0 & (0)^2 &= 0 = |-1| \\ n &= 1 & (1)^2 &= 1 = |1| \end{aligned}$$

□

Counter Example

An assignment of values to variables that shows that a universal statement is false.

2.3 Writing Proofs: Best Practices

Allowed assumptions in proofs

- the rules of algebra
- the set of integers is closed under addition, multiplication, and subtraction
- every integer is either even or odd
- if x is an integer, there is no integer between x and $x + 1$
- the relative order of any two real numbers, $x, y \in \mathbb{R}$
- the square of any real number is greater than or equal to 0

Best practices when writing proofs

- indicate when the proof starts and ends
- write proofs in complete sentences
- give the reader a road-map of what has been shown, what is assumed, and where the proof is going
- introduce each variable when the variable is used for the first time
- a block of equations should be introduced with English text and each step that does not follow from algebra should be justified

Common mistakes in proofs

- generalizing from examples
- skipping steps
- circular reasoning
- assuming facts that have not yet been proven

2.4 Writing Direct Proofs

In a **direct proof** of a conditional statement, the hypothesis p is assumed to be true and the conclusion c is proven as a direct result of the assumption.

After the assumptions are stated, a direct proof proceeds by proving the conclusion is true.

For example,

The square of every odd integer is also odd.

$\downarrow$

Let n be an integer that is odd. We will show that n^2 is also odd.

Direct Proof format

Assume hypothesis
$\vdots$
Derive conclusion

2.5 Proof by Contrapositive

A **proof by contrapositive** proves a conditional statement of the form $p \implies c$ by showing that the contrapositive $\neg c \implies \neg p$ is true. In other words, $\neg c$ is assumed to be true and $\neg p$ is proven as a result of $\neg c$.

For example,

The square of every odd integer is also odd.

$\downarrow$

Let n^2 be an integer that is *not* odd. We will show that n is also *not* odd.

Contrapositive Proof format

Assume ¬conclusion
$\vdots$
Show ¬hypothesis

2.6 Proof by Contradiction

A **proof by contradiction** starts by assuming that the theorem is false and then shows that some logical inconsistency arises as a result of the assumption. A proof by contradiction is sometimes called an **indirect proof**. A proof by contradiction shows the only option is for a theorem to be true to avoid logical errors.

For example,

The square of every odd integer is also odd.

$\downarrow$

Assume there is an even square of an odd integer. We will show there is a logical inconsistency.

Contradiction Proof format

Assume ¬theorem
⋮
Show *logical inconsistency*

2.7 Proof by Cases

A **proof by cases** of a universal statement breaks the domain into different classes and gives a different proof for each class. The proof for each class is called a **case**. **Every** value in the domain *must* be included in at least one class.

For example,

The square of every odd integer is also odd.
↓
Consider case n, where *condition*. We will show theorem is true for this case.
Consider case $n + 1$, where *condition*. We will show theorem is true for this case.

Cases Proof format

Assume hypothesis, and partition domain
⋮
Show *for each* case, the conclusion is true.

Without loss of generality

Sometimes the proof for two different cases are so similar, that it is repetitive to include both cases. When this happens, the two cases *can be merged into one case*. The term **without loss of generality, WLOG, or w.l.o.g.** is used in mathematical proofs to narrow the scope of the proof to one special case in situations when the proof can be easily adapted to apply the general case.

3 Sets

3.1 Sets and Subsets

A **set** is a collection of objects. Objects in a set are called **elements**. Order does not matter, and there are no duplicates.

Roster notation:

$$A = \{2, 4, 6, 10\}$$
$$B = \{4, 6, 10, 2\}$$
$$A = B$$

To show membership, use the $\in$ symbol. For example, $2 \in A$, while $7 \notin A$. The empty set, which contains nothing, typically uses the $\emptyset$ symbol, or $\{\}$. Sets can be finite, or infinite. **Cardinality** of a set is the number of elements in a set. For example, the cardinality of A is 4.

$$|A| = 4$$

Cardinality can be infinite. Consider the set of all the integers, $\mathbb{Z}$. $|\mathbb{Z}| = \infty$

$\mathbb{N}$: set of natural numbers
$= \{0, 1, 2, 3, \ldots\}$

$\mathbb{Z}$: set of integers
$= \{\ldots, -2, -1, 0, 1, 2, \ldots\}$

$\mathbb{B}$: set of rational numbers
$= \{x | x = \frac{a}{b} \text{ where } a, b \in \mathbb{Z}, b \neq 0\}$

$\mathbb{R}$: set of real numbers
$= \{x | x \text{ has a decimal representation}\}$

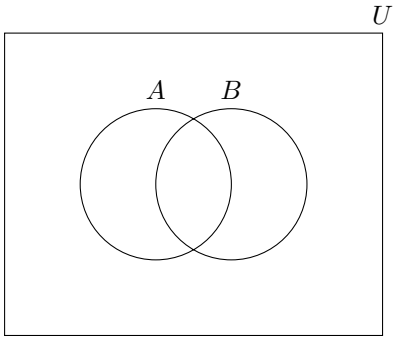
The subset operator is $\subseteq$

$$A \subseteq B \text{ if } \forall\, x(x \in A \implies x \in B)$$
$$A \subseteq A \text{ is true for any set}$$
$$\emptyset \subseteq A \text{ is true for any set}$$

Sometimes it is easier to define a set by defining properties that all the elements have. That is easy to do in **set builder notation**.

$$A = \{x \in S : A(x)\}, \text{ where } S \text{ is another set}$$
$$C = \{x \in \mathbb{Z} : 0 < x < 100 \text{ and } x \text{ is prime}\}.$$
$$D = \{x \in \mathbb{R} : |x| < 1\}$$

The **Universal Set**, usually called 'U', is a set that contains all elements mentioned in a particular context. For example, a discussion about certain types of real numbers, it would be understood that any element in the discussion is a real number. Sets are often represented pictorially with **Venn Diagrams**.



If $A \subseteq B$ and there is an element of B that is not an element of A, meaning $A \neq B$, then A is a **proper subset** of B, denoted as $A \subset B$. An important fact is that $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{B} \subset \mathbb{R}$

3.2 Sets of sets

Elements of sets can be sets themselves, consider $A = \{\{1,2\}, \emptyset, \{1,2,3\}, \{1\}\}$. The cardinality of A is 4, $|A| = 4$. Additionally, $\{1,2\} \in A$, but $1 \notin A$.

The **Powerset** of A, denoted as $P(A)$ is the set of all subsets of A. For example,

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

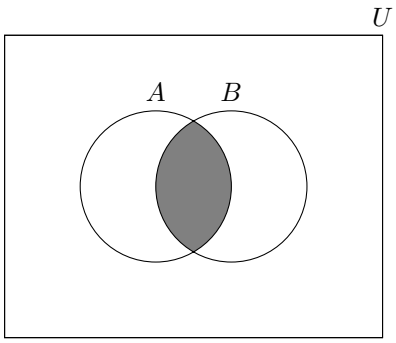
Cardinality of a Powerset

Let A be a finite set of cardinality n. Then the cardinality of the powerset of A is 2^n.

$$|A| = n$$
$$|P(A)| = 2^n$$

3.3 Union and Intersection

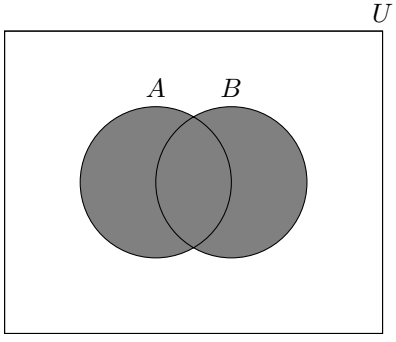
Intersection set operation: $\cap$. A intersected with B is defined to be the set containing elements which are in both A and B. That is, $A \cap B = \{x : x \in A \wedge x \in B\}$.



Intersection $\cap$ can also apply to infinite sets:

$$A = \{x \in \mathbb{Z} : x \text{ is an integer multiple of } 2\}$$
$$B = \{x \in \mathbb{Z} : x \text{ is an integer multiple of } 3\}$$
$$A \cap B = \{x \in \mathbb{Z} : x \text{ is an integer multiple of } 6\}$$

Union set operation $\cup$. A union with B is defined to be the set containing elements which are in A or B. That is, $A \cup B = \{x : x \in A \vee x \in B\}$.



A special notation, similar to $\sum$ or $\prod$ notation, allows for compound representation of the intersections or unions of a long sequence of sets.

$$\bigcap_{i=1}^{n} A_i = A_1 \cap A_2 \cap A_3 \cap \cdots \cap A_n = \{x : x \in A, \text{ for } \underline{\text{all}}\ 1 \leq i \leq n\}$$
$$\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup A_3 \cup \cdots \cup A_n = \{x : x \in A, \text{ for } \underline{\text{some}}\ 1 \leq i \leq n\}$$

Consider A_j = a word with j letters, with U = is the Oxford English Dictionary.

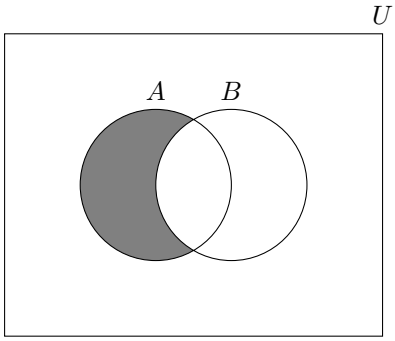
$\bigcup_{j=1}^{10} A_j =$ the set of all words with 10 letters or fewer in the OED

$\bigcap_{j=1}^{45} A_j = \emptyset$

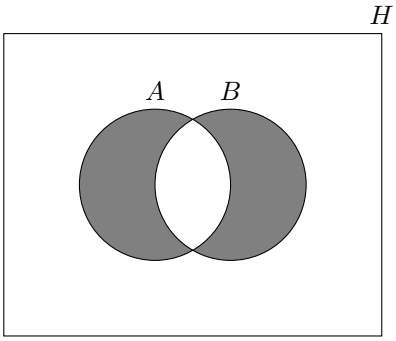
$\bigcup_{j=1}^{45} A_j =$ the set of all words in the OED.

3.4 More set operations

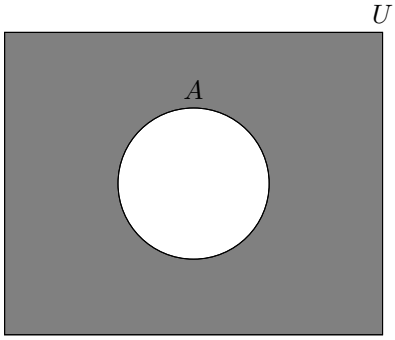
Difference set operation $-$. A difference with B is defined to be the set containing elements which are in A but <u>not</u> B. That is, $A - B = \{x : x \in A \wedge x \notin B\}$. A set difference is <u>not</u> strictly commutative, often $A - B \neq B - A$.



Symmetric Difference set operation $\triangle$. A symmetric difference with B is defined to be the set containing elements which are in A or B, but not A and B. That is, $A \triangle B = \{x : x \in A \oplus x \in B\}$.



Complement set operation . complement A is defined to be the set containing elements in U which are not in A. That is, $\overline{A} = \{x : x \in U \wedge x \notin A\}$.



Summary of Set Operations

Operation	Notation	Set Builder
Intersection	$A \cap B$	$\{x : x \in A \wedge x \in B\}$
Union	$A \cup B$	$\{x : x \in A \vee x \in B\}$
Difference	$A - B$	$\{x : x \in A \wedge x \notin B\}$
Symmetric Difference	$A \triangle B$	$\{x : x \in A \oplus x \in B\}$
Complement	$\overline{A}$	$\{x : x \in U \wedge x \notin A\}$

3.5 Set identities

The laws of propositional logic can be used to derive corresponding set identities. A **set identity** is an equation involving sets that is true, regardless of the contents of the sets used in the expression.

Law Name	$\cup$ Union	$\cap$ Intersection
Idempotent	$A \cup A = A$	$A \cap A = A$
Associative	$(A \cup B) \cup C = A \cup (B \cup C)$	$(A \cap B) \cap C = A \cap (B \cap C)$
Commutative	$A \cup B = B \cup A$	$A \cap B = B \cap A$
Distributive	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Identity	$A \cup \emptyset = A$	$A \cap U = A$
Domination	$A \cup U = U$	$A \cap \emptyset = \emptyset$
Double Complement	$\overline{\overline{A}} = A$	
Complement	$A \cup \overline{A} = \mathrm{T}$	$A \cap \overline{A} = \mathrm{F}$
DeMorgan	$\overline{A \cup B} = \overline{A} \cap \overline{B}$	$\overline{A \cap B} = \overline{A} \cup \overline{B}$
Absorption	$A \cup (A \cap B) = A$	$A \cap (A \cup B) = A$

3.6 Cartesian products

An **ordered pair** of items is written (x, y), where the first entry is x and the second entry is y. The use of () instead of {} indicates that order matters.
Cartesian Product of A and B, $A \times B = \{(a, b) : a \in A \wedge b \in B\}$

$$A = \{1, 2\} \qquad A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\}$$
$$B = \{a, b, c\} \qquad B \times A = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$$

An ordered list of 3 items is called an **ordered triple**, denoted as (x, y, z). For a size of ≥ 4, use the term **n-tuple**. For example, (u, w, x, y, z).

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) : a_i \in A \text{ for all } i \text{ such that} 1 \leq i \leq n\}$$

Another Example

$$A = \{a, b\} \qquad (a, 1, y, \beta) \in A \times B \times C \times D$$
$$B = \{1, 2\} \qquad (b, 1, x, \alpha) \in A \times B \times C \times D$$
$$C = \{x, y\} \qquad (1, b, x, \beta) \notin A \times B \times C \times D$$
$$D = \{\alpha, \beta\} \qquad \text{order matters}$$

$A \times A = A^2$, and in general,

$$A^k = \underbrace{A \times A \times \cdots \times A}_{k-times}$$

The **Cardinality of Cartesian Products**:

$$|A^n| = |A|^n$$
$$|A_1 \times A_2 \times \cdots| = |A_1| \cdot |A_2| \cdots$$

Strings

A sequence of characters is called a **string**. The set of characters used in a set of string is called the **alphabet** for the set of strings. The **length** of a string is the number of characters in the string. For example, the length of $'xxyxyx'$ is 6. The **empty string** is a string whose length is 0, and is usually denoted by λ. It is useful for A^0, for some alphabet A. $\{0,1\}^0 = \{\lambda\}$. If s and t are two strings, then the **concatenation** of s and t is the string obtained by putting s and t together.

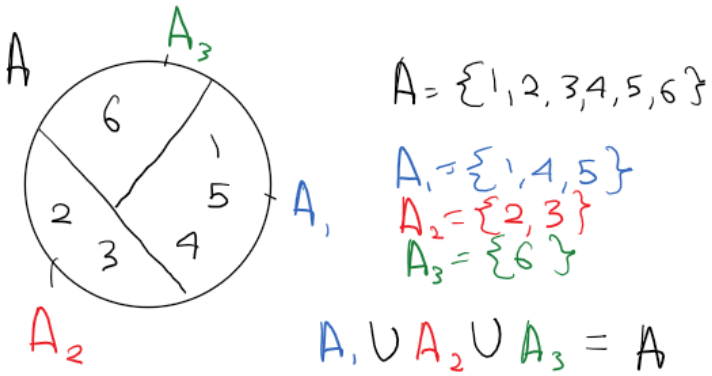
$$s = 010 \qquad st = 01011$$
$$t = 11 \qquad t0 = 110$$

Strings are used to specify passwords for computers or online accounts. Security systems vary with respect to the alphabet of characters allowed or required in a valid password. Strings also play an important rules in discrete mathematics as a mathematical tool to help count cardinality of sets.

3.7 Partitions

Two sets, A and B, are said to be **disjoint** if their intersection is empty ($A \cap B = \emptyset$). A sequence of sets, $A_1, A_2, A_3, \ldots, A_n$, is **pairwise disjoint** if every pair of distinct sets in the sequence is disjoint. A **partition** of a non-empty set A is a collection of non-empty subsets such that each element of A is in exactly one of the subsets. $A_1, A_2, A_3, \ldots, A_n$ is a partition for a nonempty set A if:

- For all i, $A_i \subseteq A$
- For all i, $A_i \neq \emptyset$
- $A_1, A_2, \ldots, A_n$ are pairwise disjoint
- $A = \bigcup_{i=1}^{n} A_i$, for some $n \in \mathbb{Z}^+$



4 Functions

4.1 Definition of functions

A **function** maps elements of one set X to elements of another set Y. A function from X to Y can be viewed as a subset of $X \times Y : (x, y) \in f$ if f maps x to y. The notation for a function is:

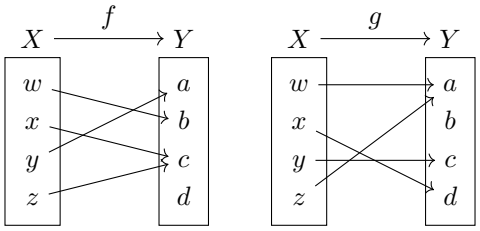
$$f : X \to Y, \text{ where } X \text{ is the } \textbf{domain} \text{ and } Y \text{ is the } \textbf{co-domain}.$$

*if f maps an element of the domain to zero elements or more than one element of the target, then f is not *well-defined*

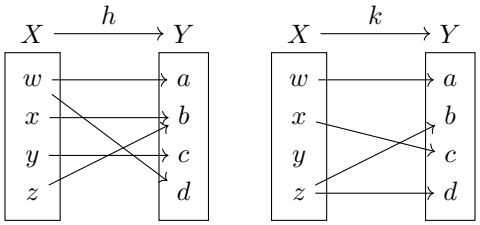
Arrow Diagram:

$$X = \{w, x, y, z\}$$
$$Y = \{a, b, c, d\}$$

Well-defined functions:



Not well-defined functions:



For function $f : X \to Y$, an element y is in the **range** of f iff there is an $x \in X$ such that $(x, y) \in f$.

$$\text{Range of } f = \{y : (x, y) \in f, \text{ for some } x \in X\}$$

Two functions, f and g, are **equal** if f and g have the same domain and target and $f(x) = g(x)$ for every x in the domain.

$$\forall\, x : f(x) = g(x) \implies f = g$$

4.2 Floor and Ceiling functions

The **Floor** function, $\lfloor x \rfloor$

$$\text{floor} : \mathbb{R} \to \mathbb{Z}, \text{ where floor}(x) = \text{the largest integer } y \text{ such that } y \leq x.$$

Notation: $\text{floor}(x) = \lfloor x \rfloor$

The **Ceiling** function, $\lceil x \rceil$

$$\text{ceiling} : \mathbb{R} \to \mathbb{Z}, \text{ where ceiling}(x) = \text{the smallest integer } y \text{ such that } y \geq x.$$

Notation: $\text{ceiling}(x) = \lceil x \rceil$

Examples of floor and ceiling:

$$\lceil 4.32 \rceil = 5 \qquad \lfloor 4.32 \rfloor = 4$$
$$\lceil -4.32 \rceil = -4 \qquad \lfloor -4.32 \rfloor = -5$$
$$\lceil 4 \rceil = 4 \qquad \lfloor 4 \rfloor = 4$$
$$\lceil -4 \rceil = -4 \qquad \lfloor -4 \rfloor = -4$$

4.3 Properties of functions

A function $f : X \to Y$ is **one-to-one** or **injective** if $x_1 \neq x_2$ implies that $f(x_1) \neq f(x_2)$. f maps different elements in x to different elements in y.

A function $f : X \to Y$ is **onto** or **surjective** if the range of f is equal to the target Y. That is, $\forall\, y \exists\, x(y \in Y \wedge x \in X \wedge f(x) = y)$

A function $f : X \to Y$ is **bijective** if it is both **injective** and **surjective**. A **bijective** function is called a **bijection**, or a **one-to-one correspondence**.

When the domain and target are finite sets, it is possible to infer information about their relative sizes based on whether a function is one-to-one or onto.

$$f : D \to T \text{ is } \textbf{one-to-one} \implies |D| \leq |T|$$
$$f : D \to T \text{ is } \textbf{onto} \implies |D| \geq |T|$$
$$f : D \to T \text{ is } \textbf{bijective} \implies |D| = |T|$$

4.4 The inverse of a function

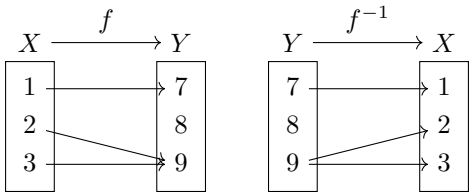
If a function $f : X \to Y$ is a *bijection*, then the **inverse** of f is obtained by exchanging the first and second entries in each pair in f.

$$\text{given } f : X \to Y$$
$$\text{inverse } f^{-1} : \{(y, x) : (x, y) \in f\}$$

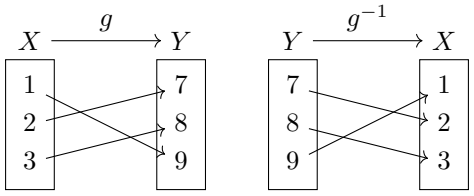
Reversing the cartesian pair does not always create a well-defined function. *Some functions do not have an inverse.*

Examples:

$$X = \{1, 2, 3\} \qquad f = \{(1, 7), (2, 9), (3, 9)\}$$
$$Y = \{7, 8, 9\} \qquad g = \{(1, 9), (2, 7), (3, 8)\}$$



f^{-1} is not well defined, therefore f does not have an inverse.



g^{-1} is well defined, therefore g does have an inverse.

4.5 Composition of functions

The process of applying a function to the result of another function is called **composition**.

$$\begin{aligned} f &: X \to Y \\ g &: Y \to Z \\ (g \circ f) &: X \to Z, \text{ such that } \forall\, x : x \in X, (g \circ f)(x) = g(f(x)) \end{aligned}$$

Remember that order matters, as often $(g \circ f)(x) \neq (f \circ g)(x)$. However, composition is associative:

$$(f \circ g \circ h)(x) = ((f \circ g) \circ h)(x) = (f \circ (g \circ h))(x) = f(g(h(x)))$$

Identity Function

The **Identity Function** maps a set onto itself and maps every element to itself. It is notated as $I_A : A \to A$, where A is the set it maps. There are a number of identities about the Identity Function.

Let $f : A \to B$ be a bijection. Then,

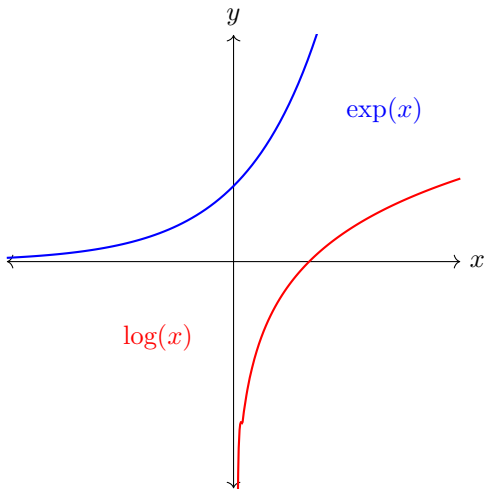
$$f \circ f^{-1} = I_B \text{ and } f^{-1} \circ f = I_A$$

4.6 Logarithms and exponents

The **Exponential** function, $\exp_b : \mathbb{R} \to \mathbb{R}^+, \exp_b(x) = b^x$. b is the base of the exponent and x is the exponent.

Properties of exponents:

$$\begin{aligned} b^x b^y &= b^{x+y} \\ (b^x)^y &= b^{xy} \\ \frac{b^x}{b^y} &= b^{x-y} \\ (bc)^x &= b^x c^x \end{aligned} \qquad \begin{aligned} &b \in \mathbb{R}^+ \\ &x \in \mathbb{R} \end{aligned} \qquad \begin{aligned} &c \in \mathbb{R}^+ \\ &y \in \mathbb{R} \end{aligned}$$



The **Logarithms** function, $\log_b : \mathbb{R} \to \mathbb{R}^+, \log_b(y) = x$. b is the base of the logarithm and x is the exponent.

Properties of exponents:

$$\log_b(xy) = \log_b x + \log_b y \qquad b \in \mathbb{R}^+$$
$$\log_b(\frac{x}{y}) = \log_b x - \log_b y \qquad c \in \mathbb{R}^+$$
$$\log_b(x^y) = y\log_b x \qquad x \in \mathbb{R}$$
$$\log_c x = \frac{\log_b x}{\log_b c} \qquad y \in \mathbb{R}$$

5 Boolean Algebra

5.1 An introduction to Boolean Algebra

Boolean Algebra is a set of rules/operations for working with variables whose values are either 0 or 1. It corresponds highly to propositional logic.

Boolean Multiplication, denoted by $\cdot$

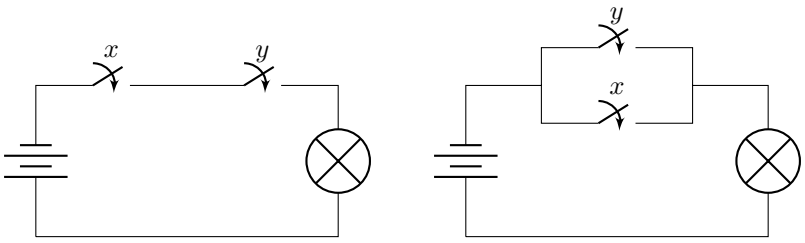
Boolean $\cdot$	Logic $\wedge$
$0 \cdot 0 = 0$	$\mathrm{F} \wedge \mathrm{F} = \mathrm{F}$
$0 \cdot 1 = 0$	$\mathrm{F} \wedge \mathrm{T} = \mathrm{F}$
$1 \cdot 0 = 0$	$\mathrm{T} \wedge \mathrm{F} = \mathrm{F}$
$1 \cdot 1 = 1$	$\mathrm{T} \wedge \mathrm{T} = \mathrm{T}$

Boolean Addition, denoted by $+$

Boolean $\cdot$	Logic $\vee$
$0 + 0 = 0$	$\mathrm{F} \vee \mathrm{F} = \mathrm{F}$
$0 + 1 = 1$	$\mathrm{F} \vee \mathrm{T} = \mathrm{T}$
$1 + 0 = 1$	$\mathrm{T} \vee \mathrm{F} = \mathrm{T}$
$1 + 1 = 1$	$\mathrm{T} \vee \mathrm{T} = \mathrm{T}$

Boolean Complement, denoted by $^-$

Boolean $^-$	Logic $\neg$
$\bar{0} = 1$	$\neg\mathrm{F} = \mathrm{T}$
$\bar{1} = 0$	$\neg\mathrm{T} = \mathrm{F}$



Shannon Circuit (AND $\cdot$) Switching Circuit (OR $+$)

Variables that can have a value of either 1 or 0 are called **Boolean Variables**. Boolean expressions are made of boolean variables. There are also common shorthand ways of notating operations.

$$x \cdot y + 1 \cdot \bar{z} = xy + \bar{z}$$
$$x + z + \overline{0 + y} = x + z \cdot \bar{y}$$

Law Name	+ OR	· AND
Idempotent	$x + x = x$	$x \cdot x = x$
Associative	$(x + y) + z = x + (y + z)$	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$
Commutative	$x + y = y + x$	$x \cdot y = y \cdot x$
Distributive	$x + (y \cdot z) = (x + y) \cdot (x + z)$	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
Identity	$x + 0 = x$	$x \cdot 1 = x$
Domination	$x + 1 = 1$	$x \cdot 0 = 0$
Double Complement	$\overline{\overline{x}} = x$	
Complement	$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$
DeMorgan	$\overline{x + y} = \overline{x} \cdot \overline{y}$	$\overline{x \cdot y} = \overline{x} + \overline{y}$
Absorption	$x + (x \cdot y) = x$	$x \cdot (x + y) = x$

5.2 Boolean functions

A **boolean function** is a function which maps $B^k \to B$, where $B = \{0, 1\}$. For example, consider $f : B^3 \to B$

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

x	y	z	$f(x,y,z)$	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\bar{x}yz$
1	0	0	1	$x\bar{y}\bar{z}$
1	0	1	1	$x\bar{y}z$
1	1	0	0	
1	1	1	1	xyz

This function is equivalent to $f(x, y, z) = x\overline{y} + yz$. This can be determined using the boolean table and a strategy of combining the cases.

$$\begin{aligned} f(x,y,z) &= \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z + xyz \\ &= y(\bar{x}z + xz) + \bar{y}(x\bar{z} + xz) \\ &= y(z \cdot 1) + \bar{y}(x \cdot 1) \\ &= yz + \bar{y}x = \bar{y}x + yz \\ &= x\bar{y} + yz \end{aligned}$$

A **literal** is a boolean variable or the complement of a boolean variable, for example x or $\bar{x}$. In a boolean function whose input variables are $v_1, v_2, \ldots, v_k$, a *mini-term* is a product of literals $u_1, u_2, \ldots, u_k$, such that u_j is either v_j or $\overline{v_j}$

5.3 Disjunctive and conjunctive normal form

A boolean expression that is the sum of literals is said to be in *disjunctive normal form*, **DNF**. It has the following form:

$$c_1 + c_2 + \cdots + c_m, \text{ where } c_j \text{ for } j \in \{1, \ldots, m\} \text{ is a product of literals.}$$

For example, $\bar{x}y\bar{z} + xy + w + y\bar{z}w$. The complement only applies to a single variable and no addition within a term.

A boolean expression that is the product of sums of literals is said to be in *conjunctive normal form*, **CNF**. It has the following form:

$$d_1 \cdot d_2 \cdots d_m, \text{ where } d_j \text{ for } j \in \{1, \ldots, m\} \text{ is a sum of literals.}$$

Each d_j is called a clause, and complements are only applied to a single variable. Additionally, there is no multiplication within variables. An example is $(\bar{x} + y + z)(x + \bar{y})(w)(y + \bar{z} + w)$.

5.4 Functional completeness

A set of operators is functionally complete if any boolean function can be expressed using only operations from the set. Two expressions can be added using only multiplication and complement.

$$x + y = \overline{\bar{x}\bar{y}} \text{ DeMorgan's Law}$$

DeMorgan's Law can be extended to more than two boolean variables:

$$x + y + w + z = \overline{\bar{x}\bar{y}\bar{w}\bar{z}}$$

The same can be said about the addition variant of the law:

$$xy = \overline{\bar{x} + \bar{y}}$$
$$xywz = \overline{\bar{x} + \bar{y} + \bar{w} + \bar{z}}$$

The **NAND** operation, $\uparrow$, and the **NOR** operation, $\downarrow$.

x	y	$x \uparrow y$	$x \downarrow y$
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

{NAND} is functionally complete, it can create the complement, from which all other possible gates can be created.

x	$x \uparrow x$
0	1
1	0

From the complement, AND can be created, and {AND, COMPLEMENT} has already been proven to be functionally complete.

$$xy = \overline{x \uparrow y} = (x \uparrow y) \uparrow (x \uparrow y)$$

5.5 Boolean satisfiability

The **Boolean Satisfiability problem**, called the *SAT* for short, takes the boolean expression as an input and asks whether it is possible to set the values of the variables so that the expression evaluates to 1.

If $\exists\, x \exists\, y \ldots B(x, y, \ldots)$, then the expression is **satisfiable**
If $\forall\, x \forall\, y \ldots \neg B(x, y, \ldots)$, then the expression is **unsatisfiable**

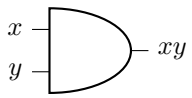
Expressions in DNF form are very easy to determine satisfiability. If there is <u>any</u> term which does *not* contain a variable and its complement, it is satisfiable. For example,

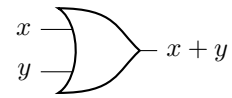
$$x\bar{y}z\bar{x} + \overbrace{\bar{w}xy\bar{z}}^{\text{no self-complements}} + \bar{w}xw\bar{x} + xy\bar{z}z$$

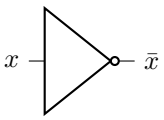
The above equation *is* satisfiable because there is a term which does not contain a self-complement.

5.6 Gates and circuits

Circuits are built from electrical devices called **gates**.

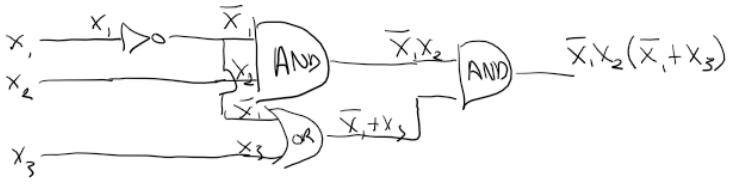
AND: 

OR: 

NOT/INVERTER: 

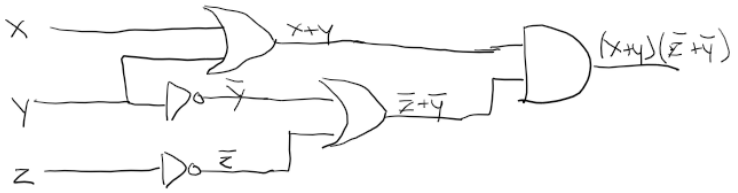
The boolean function $f(x_1, x_2) : (f(x_1, x_2) \cdot x_1) + x_2$. Yes, circuits can contain recursion.

Boolean expressions can also be constructed by following the logic or a circuit.



$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 (\bar{x}_1 + x_3)$$

An example of constructing a circuit from a boolean expression, $f(x, y, z) : (x + y)(\bar{z} + \bar{y})$



Designing Circuits

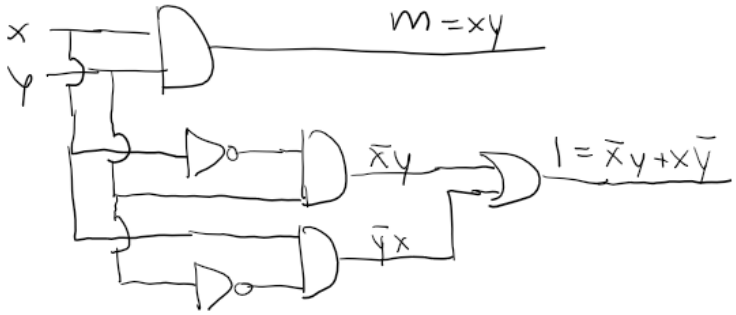
1. Build an input/output table with the desired output for every combination of input
2. Construct a boolean expression that computes the same function as the function specified in the input/output table
3. Construct a digital circuit that realizes the boolean expression

I/O for sum of two bits, x, y

x	y	m	l
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

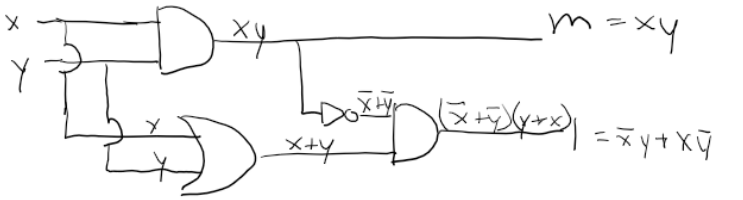
$m = xy$ most significant bit

$l = x\bar{y} + \bar{x}y$ least significant bit

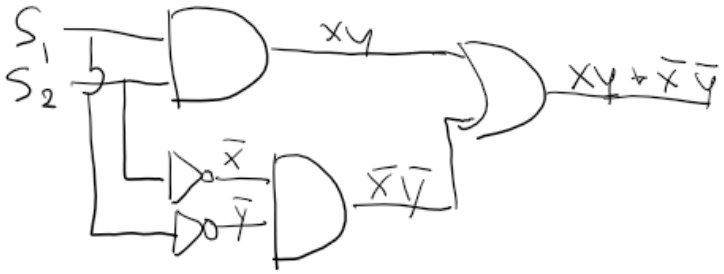


*this method does not necessarily give the most efficient circuit.

Circuits with fewer gates cost less to manufacture. Here is a simplified circuit to add two bits:



Here is another example with boolean logic for light switches:



6 Relation and Digraphs

6.1 Introduction to binary relations

A **Binary Relation** between two sets A and B is a subset R of $A \times B$. It is binary because it is between two sets.

$$\text{for } a \in A \wedge b \in B, (a,b) \in R \text{ is denoted as } a\mathrm{R}b$$

For example, consider the relation C between $\mathbb{R}$ and $\mathbb{Z}$:

$$x\mathrm{C}y \text{ if } |x-y| \leq 1, \text{ where } x \in \mathbb{R} \text{ and } y \in \mathbb{Z}$$

If A and B are finite, then relation R between A and B can be represented by a set of ordered pairs.

Matrix Representation

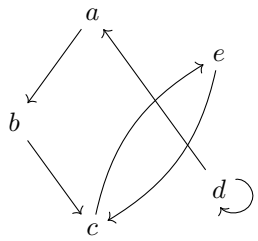
$$P = \{\text{Sue}, \text{Harry}, \text{Sam}\}$$
$$\text{File} = \{\text{File A}, \text{File B}, \text{File C}, \text{File D}\}$$

$$\begin{array}{c} \\ \text{Sue} \\ \text{Harry} \\ \text{Sam} \end{array} \begin{array}{c} \begin{array}{cccc} \text{File A} & \text{File B} & \text{File C} & \text{File D} \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$\text{An element is } \begin{array}{l} 1 \text{ if } p\mathrm{R}f \text{ is true} \\ 0 \text{ if } p\mathrm{R}f \text{ is false} \end{array}$$

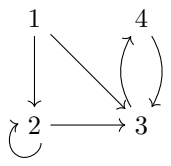
Arrow Diagram

$$A = \{a, b, c, d, e\}$$
$$R \subseteq A \times A$$
$$R = \{(a,b), (b,c), (e,c), (c,e), (d,a), (d,d)\}$$



Arrow Diagram vs. Matrix Representation

$$A = \{1, 2, 3, 4\}$$
$$R = \{(1,2), (1,3), (2,2), (2,3), (3,4), (4,3)\}$$



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

6.2 Properties of binary relations

A binary relation of R on set A is **Reflective** if for *every* $x \in A$, $x\mathrm{R}x$. For Arrow Diagrams, this means the graph contains self-loops:

For Matrix Representation, this means that the top left to bottom right diagonal are all 1's:

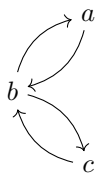
$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1 & - & - & - \\ - & 1 & - & - \\ - & - & 1 & - \\ - & - & - & 1 \end{pmatrix} \end{array}$$

A binary relation of R on set A is **Anti-reflective** if for *every* $x \in A$, $x\mathrm{R}x$ is *not* true. For Arrow Diagrams, this means the graph does not contain self-loops:

For Matrix Representation, this means that the top left to bottom right diagonal are all 0's:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & - & - & - \\ - & 0 & - & - \\ - & - & 0 & - \\ - & - & - & 0 \end{pmatrix} \end{array}$$

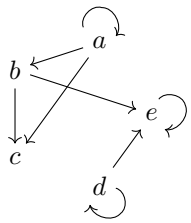
A binary relation of R on set A is **Symmetric** if and only if for *every* pair $x \in A$, $y \in Y$, either *both* $x\mathrm{R}y$ and $y\mathrm{R}x$, or *both* not $x\mathrm{R}y$ or not $y\mathrm{R}x$ is true. For Arrow Diagrams, this means that every arrow has an arrow going the other way:



For Matrix Representation, this means that the matrix is symmetric along the top left to bottom right diagonal:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} - & u & v & x \\ u & - & w & y \\ v & w & - & z \\ x & y & z & - \end{pmatrix} \end{array} \text{ where } \begin{array}{c} u \in \{0,1\} \\ \vdots \\ z \in \{0,1\} \end{array}$$

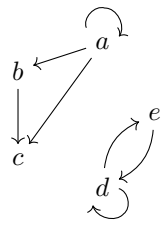
A binary relation of R on set A is **Anti-symmetric** if and only if for *every* pair $x \in A$, $y \in Y$, $x\mathrm{R}y$ xor $y\mathrm{R}x$. For Arrow Diagrams, this means that each arrow does not have an arrow going the other way:



For Matrix Representation, this means that the matrix is anti-symmetric along the top left to bottom right diagonal:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} - & \bar{u} & \bar{v} & \bar{x} \\ u & - & \bar{w} & \bar{y} \\ v & w & - & \bar{z} \\ x & y & z & - \end{pmatrix} \end{array} \text{ where } \begin{array}{c} u \in \{0,1\} \\ \vdots \\ z \in \{0,1\} \end{array}$$

A binary relation of R on set A is **Transitive** if for *every* three elements $x, y, z \in A$, if $x\mathrm{R}y$ and $y\mathrm{R}z$, then $x\mathrm{R}z$. Logically, $(x\mathrm{R}y \wedge y\mathrm{R}z) \implies x\mathrm{R}z$. For Arrow Diagrams, this means the graph follows a hierarchy or kind of flow:



For Matrix Representation, it is much more difficult to determine transitivity, but here is an example:

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array}\begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

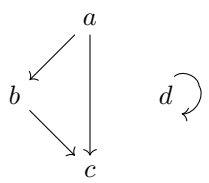
6.3 Directed graphs, paths, and cycles

A directed graph, or **Diagraph**, consists of a pair (V, E), where V is the set of vertices and E is the set of *directed edges*. It is a subset of $V \times V$.

- indegree: # of edges pointing towards a vertex, $\text{indegree}(u) = |\{v : (v, u) \in E\}|$
- outdegree: # of edges pointing away from a vertex, $\text{outdegree}(u) = |\{u : (v, u) \in E\}|$

A digraph is organized into a cartesian pair of the set of vertices and edge pairs:

$$\begin{aligned} \text{Graph } G &= (V, E) \\ V &= \{a, b, c, d\} \\ E &= \{(a, b), (b, c)(a, c), (d, d)\} \end{aligned}$$

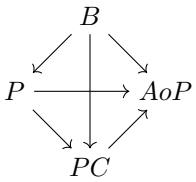


$\text{indegree}(c) = 2$	$\text{outdegree}(a) = 2$	a is the tail of edge (a, b)
$\text{indegree}(d) = 1$	$\text{outdegree}(d) = 1$	b is the head of edge (a, b)

A digraph is mathematically the same as a relation. Here is an example of the internet as a graph:

Graph $G = (V, E)$
$V =$ set of all URLs
$E =$ set of all hyperlinks from one URL to another URL

$B =$ Blog
$P =$ Pediatrician website
$PC =$ Pharmaceutical Company
$AoP =$ Academy of Pediatrics



Walks in Directed Graphs

A *walk* is a sequence of vertices and edges. For example, a walk from v_0 to v_l is notated as:

$$\langle v_0, (v_0, v_1), v_1, (v_1, v_2), \ldots, (v_{l-1}, v_l), v_l \rangle$$

where each edge in the sequence appears after its tail and before its head. A walk can also be a set of vertices:

$$\langle v_0, v_1, \ldots, v_l \rangle$$

provided that the edges between the vertices *actually exist*. The **length** of a walk is the number of edges traversed.

- **Open walk**: first and last vertices are *not* the same
- **Closed walk**: first and last vertices *are* the same.

Trails, Circuits, Paths, Cycles

- **Trail**: *open* walk in which no edge occurs more than once.
- **Circuit**: *closed* walk in which no edge occurs more than once. A circuit is a closed trail.
- **Path**: *trail* where no vertex occurs more than once.
- **Cycle**: *circuit* where no vertex occurs more than once, *except* for the first and last, which are the same.

Here are some examples:

$\langle a, c, d, a \rangle$ is a **cycle**: only the first and last vertices are repeated.
$\langle a, c, a, d, a \rangle$ is a **circuit**: vertices are repeated, but not edges
$\langle a, b, c, b, d \rangle$ is a **trail**: open, vertices are repeated, but not edges

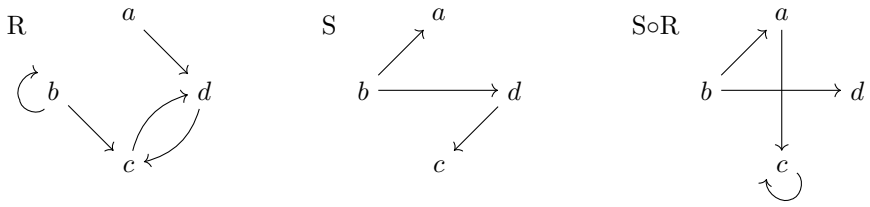
6.4 Composition of relations

- one-to-one correspondence between digraphs and binary relations
- arrow diagram for a binary relation *is* a directed graph

The **composition** of relations R and S on set A is denoted as S∘R. Logically, this is what is means:

$$(a, c) \in S \circ R \iff \exists\, b : (b \in A \wedge (a, b) \in R \wedge (b, c) \in S)$$

Composition is applied *right to left*, much like composition of functions, or matrix transformations. Therefore, S∘R means R is applied first, then S.



6.5 Graph powers and the transitive closure

A relation can be composed with itself. For example, consider relation P, which expresses parent-child relationship.

xPy means x is the parent of y.
xP∘Pz means x is the grandparent of z

A relation composed with itself also represents walks of different lengths.

P∘P represents all walks of length 2.
P∘P∘P represents all walks of length 3.

The Graph Power Theorem

: Let G be a directed graph. Let u and v be any two vertices in G. There is an edge from u to v in G^k if and only if there is a walk of length k from u to v in G.

$$R^1 = R$$
$$R^k = R \circ R^{k-1} \text{ for all } k \geq 2$$

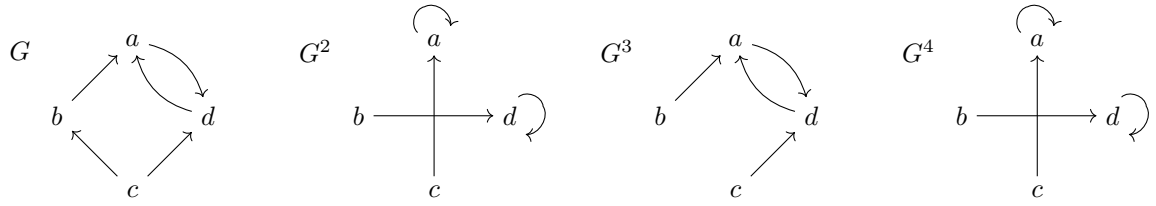
Transitive Closure

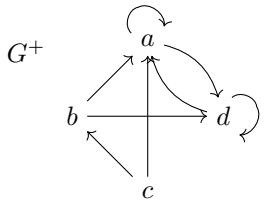
$$G^+ = G^1 \cup G^2 \cup G^3 \cup \cdots$$

if G is not infinite, only up to the number of vertices are required for a complete graph of G^+

$$G^+ = G^1 \cup G^2 \cup G^3 \cup \cdots \cup G^{|V|}$$

Here is an example of a series of graph powers:



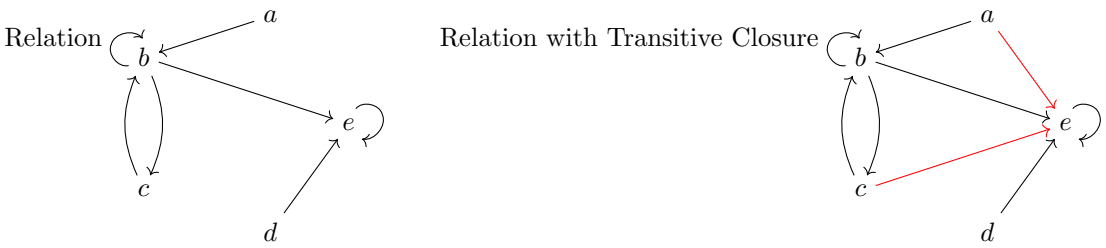


Finding the Transitive Closure of a Relation R on set A

Repeat until no pair is added to R.

If there are 3 elements $x, y, z \in A$ such that $x\mathrm{R}y$ and $y\mathrm{R}z$ but not $x\mathrm{R}z$, add $x\mathrm{R}z$ to R.

For example:



Edges added to find transitive closure are shown in red.

6.6 Matrix multiplication and graph powers

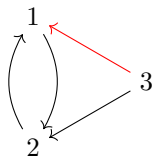
An $n \times m$ **matrix** over set S is an array of elements from S with n rows and m columns. Each element in a matrix is called an *entry*. A **square matrix** has the same number of rows and columns. Here are a number of example matrixes

$$\begin{bmatrix} 1 & 3 \\ 3 & -5 \\ -2 & -2 \end{bmatrix} \qquad \begin{bmatrix} 1.1 & 3.0 & -5.4 \\ -2.2 & -2.1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

3×2 matrix over $\mathbb{Z}$ $\qquad$ 2×3 matrix over $\mathbb{Z}$ $\qquad$ 2×2 matrix over $\{0, 1\}$

A directed graph G can be represented by a Matrix.

n vertices $\to$ $n \times n$ matrix over the set {0,1}, called an **adjacency matrix**



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

A **boolean matrix** is a matrix over the set {0,1}, and boolean addition and multiplication are used. The **dot product** of a matrix A and B is defined only if # of columns in A = # of rows in B

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$\begin{array}{r} 1 \\ \times\ 1 \\ \hline 1 \end{array} + \begin{array}{r} 0 \\ \times 1 \\ \hline 0 \end{array} + \begin{array}{r} 1 \\ \times\ 1 \\ \hline 1 \end{array} = 1 = (A \times B)_{2,3}$$

Matrix Product

- denoted as AB or $A \cdot B$
- uses a series of dot products to compute

There are a number of properties of matrix multiplication:

~~Commutative~~	$AB \neq BA$
Associative	$(AB)C = A(BC)$
Distributive	$A(B+C) = AB + AC$
	$(B+C)A = BA + CA$
Multiplicative	$IA = A$ and $AI = A$
	$OA = A$ and $AO = O$
Dimension	$(m \times n) \cdot (n \times k) = (m \times k)$

k^{th} power of a matrix:

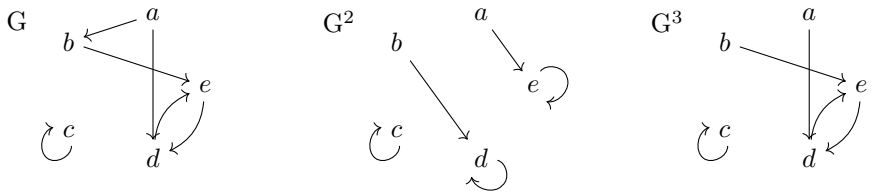
$$A^k = \underbrace{A \cdot A \cdots A}_{k \text{ times}}$$

If G is a digraph, G^k represents all walks of length k in G. There is an edge from vertex v to vertex w in G^k if and only if there is a walk of length *exactly* k from v to w in G. Matrix multiplication provides a systematic way of computing G^k.

1. Take *adjacency matrix* A for graph G
2. Compute A^k by repeated *matrix multiplication*
3. Matrix A^k is the *adjacency matrix* for graph G^k.

Relationship between powers of a graph and the powers of its adjacency matrix

Let G be a directed graph with n vertices and let A be the adjacency matrix for G. Then for $k \geq 1$, A^k is the adjacency matrix of G^k, where boolean addition and multiplication are used to compute A^k.



Example:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad A^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad A^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Matrix Sum

- denoted as $A + B$
- well-defined if same # of row and # of columns

$$(A+B)_{i,j} = A_{i,j} + B_{i,j} \text{ for all } i \text{ and } j \text{ in range of } A \text{ or } B$$

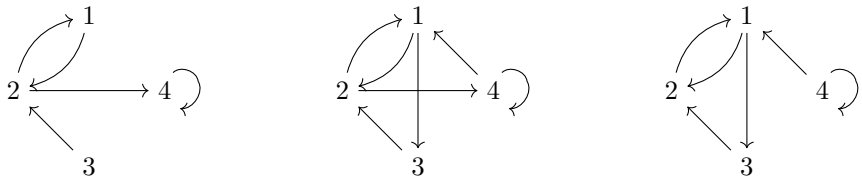
For example,

$$\begin{array}{ccccc} & \begin{matrix} 0 & & \end{matrix} & + & \begin{matrix} & 1 & \end{matrix} & = & \begin{matrix} & 1 & \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & & + & \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} & = & \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \\ A & & + & B & = & A+B \end{array}$$

Addition and Graph Union

Let G and H be two directed graphs with the same vertex set. Let A be adjacency matrix for G and B the adjacency matrix for H.

Then the adjacency matrix for $G \cup H = A + B$, where boolean addition is used on the entries of matrices A and B.



$$\begin{array}{ccccc} & & \cup & & = \\ G & & \cup & H & = & G \cup H \\ A & & + & B & = & A+B \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & & + & \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} & = & \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

For digraph G and adjacency matrix A for G:

$$G^+ = G^1 \cup G^2 \cup G^3 \cup \cdots \cup G^n$$
$$A^+ = A^1 + A^2 + A^3 + \cdots + A^n$$

Condition for well-defined matrix multiplication

$A_{m\times n} \times B_{s\times t}$ is only defined when $n = s$, and $A \times B$ has dimensions $m \times t$. For example:

$$A_{3\times 3}\begin{bmatrix} 0 & 2 & 3 \\ 1 & 0 & 1 \\ 2 & 0 & 1 \end{bmatrix} \times B_{3\times 1}\begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = (A \times B)_{3\times 1}\begin{bmatrix} 5 \\ 4 \\ 0 \end{bmatrix}$$

6.7 Partial orders

A relation on set A is a **partial order** if it is:

- reflexive
- transitive

- anti-symmetric

Notation: $a \preceq b$ used to express $a\mathrm{R}b$

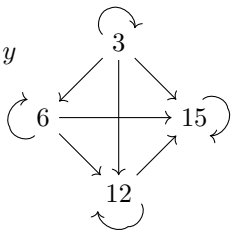
Domain along with a partial order defined on it is denoted $(A, \preceq)$ and is called a **partial ordered set** or **poset**.

$$\preceq \neq \leq \text{ (notice the curves)}$$

Two elements of a poset are said to be *comparable* if $x \preceq y$ or $x \succeq y$. Otherwise they are said to be *incomparable*. A partial order is a *total order* if every two elements in the domain are *comparable*.

Here is an example of a partial order:

$x\mathrm{R}y$ if $x \leq y$



- An element x is **minimal** element if there is no such $y \neq x$ such that $y \preceq x$
- An element x is **maximal** element if there is no such $y \neq x$ such that $x \preceq y$

Hasse Diagram

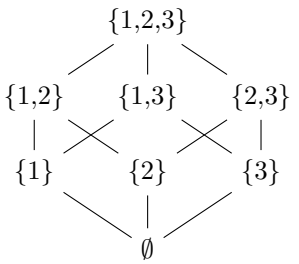
- useful way to depict a partial order on a finite set
- each element is represented by a point
- shows relationships by placing elements that are greater than others toward the top of the diagram.

Rules for placement and for connecting segments

- if $x \preceq y$, then make x appear lower in the diagram than y
- if $x \preceq y$, and there is no such z that $x \preceq z \preceq y$, then draw a segment connecting x and y

Examples: Hasse Diagrams for a partial order on the power set of $\{1, 2, 3\}$, and $\{1, 2, 3, 4, 5, 6\}$.

Hasse Diagram 1

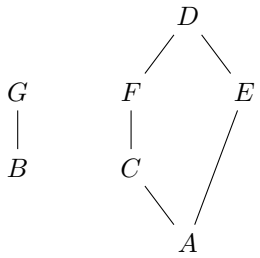


Hasse Diagram 2

The first example uses a rule of $A \preceq B \leftrightarrow A \subseteq B$, while the second example uses a rule of $x \preceq y \leftrightarrow x \leq y$.

In general, if two elements are incomparable, then they are not connected at all by a path of line segments or the only paths between x and y require a change in direction from up to down or down to up. Consider this partial order on the set $\{A, B, C, D, E, F, G\}$:

Hasse Diagram 3



In this example, $B \preceq G$, and $A \preceq D$, but B and F are not comparable.

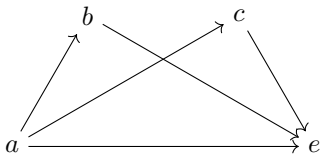
6.8 Strict orders and directed acyclic graphs

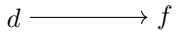
- Partial Order acts $\preceq$ on a domain
- Strict Order acts $\prec$ on a domain

A relation R is a **Strict Order** if R is *transitive, anti-symmetric,* and *anti-reflexive.*

- two elements are comparable if $x \prec y$ or $x \succ y$, and otherwise incomparable
- strict order is a *total order* if every pair of elements is comparable
- element x is **minimal** if no y exists such that $y \prec x$
- element x is **maximal** if no y exists such that $x \prec y$

Here is an example of a strict order:



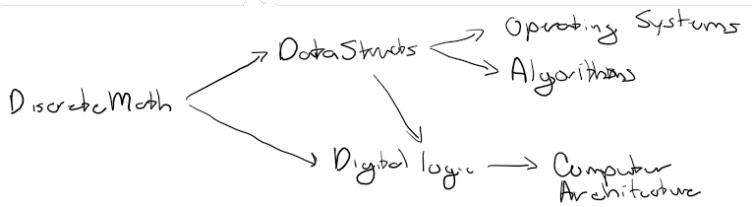


maximal: e and f
minimal: a and d

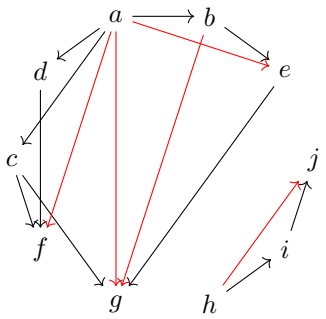
Strict orders are also anti-symmetric. Consider a relation R that is transitive and anti-reflexive. Then R is also anti-symmetric.

Directed Acyclic Graphs, DAGs

A directed acyclic graph is a directed graph that has no cycles. For example, consider this DAG:



Theorem: Directed Acyclic Graphs and Strict Orders Let G be a directed graph. G has no cycles if and only if G^+ is a strict order.

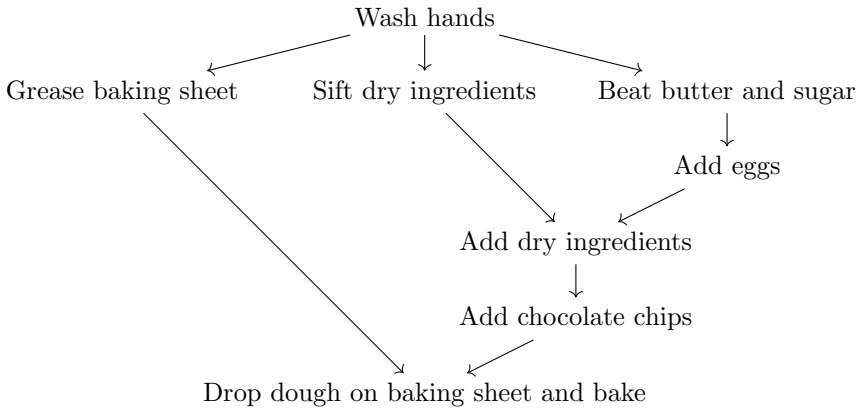


G is a DAG
Edges added by G^+ are shown in red
Minimal: a and h
Maximal: g, f, and j

Consider another example of precedence constraints for baking chocolate chip cookie:

- Wash hands
- Grease cooking sheet
- Sift together dry ingredients
- Beat together butter and sugar
- Add eggs to butter and sugar
- Add chocolate chips
- Drop spoonfuls of batter onto cookie sheet and bake

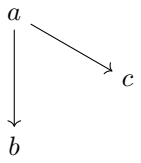
Converting this into a DAG, notice how incomparable tasks can be done simultaneously by different people:



Topological Sorts of DAGs

Consider a DAG which represents precedence constraints for a set of tasks. Need to find an ordering which does not violate any of the precedence constraints. A **topological** sort for a DAG is an ordering of vertices that is consistent with the edges in the graph.

- If there is an edge (u, v), then u must appear earlier than v in the topological sort.



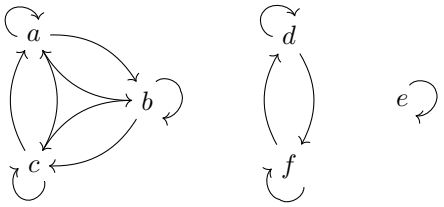
example topological sorts:
$\langle a, b, c \rangle$
$\langle a, c, b \rangle$

6.9 Equivalence relations

A relation R is an **equivalence relation** if:

- R is reflective
- R is transitive
- R is symmetric

For example:



- Reflective
- Symmetric
- Transitive

If A is the domain of a equivalence relation and $a \in A$, then $[a]$ is called an **equivalence class**, defined to be the set of all $x \in A$, such that $a \sim x$.

For example, consider domain $\mathbb{Z}^+$ and $x \sim y$ if x and y have the same remainder when divided by 3.

$$[0] \text{ is } \{x \in \mathbb{Z}^+ : x \mod 3 = 0\}$$
$$[1] \text{ is } \{x \in \mathbb{Z}^+ : x \mod 3 = 1\}$$
$$[2] \text{ is } \{x \in \mathbb{Z}^+ : x \mod 3 = 2\}$$

Equivalence classes are either:

- completely identical, or
- completely disjoint

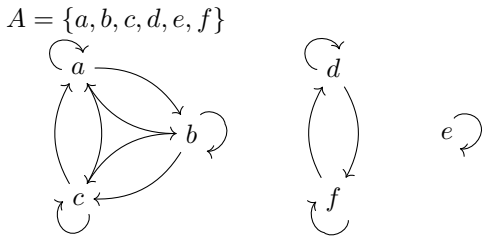
Theorem: Structure of Equivalence Relations Consider an equivalence relation on a set A. Let $x, y \in A$:

$$\text{If } x \sim y, \text{ then } [x] = [y]$$
$$\text{If } x \not\sim y, \text{ then } [x] \cap [y] = \emptyset$$

Theorem: Equivalence Relations define a Partition Consider and equivalence relation over a set A. The set of all distinct equivalence classes defines a partition of set A.

- The term "distinct" means that if there are two equivalent classes $[a] = [b]$, then set $[a]$ is the only included one.

For example:



6.10 N-ary relations and relational databases

A binary relation can be generalized to more than two sets. A relation on n sets is called an **N-nary Relation**. For example:

$(w, x, y, z) \in \mathbb{R}^4$ such that $wx = yz$	
$(3, 12, 4, 9)$ would be in the relation because	$3 \cdot 12 = 4 \cdot 9$
$(3, 8, 5, 6)$ would not be in the relation because	$3 \cdot 8 \neq 5 \cdot 6$

Databases

A database is a large collection of data records that is searched and manipulated by a computer. The *regional database model* stores data records as relations.

The type of data stored in each entry of the n-tuple is called an attribute.

A query to a database is a request for a particular set of data.

A key is an attribute or set of attributes that uniquely identifies each n-tuple in the databases.

For example, Airlines use the combination of flight number, date, and departure time to uniquely identify a flight.

Selection

The **selection** operation chooses n-tuples from a relational database that satisfies particular conditions on their attributes. For example:

# boxes	Order Date	Complete?	Client City
8	6/19/2013	NO	Irvine
12	6/20/2013	YES	Huntington Beach
15	6/20/2013	YES	Huntington Beach
21	6/20/2013	NO	Irvine
3	6/21/2013	NO	Costa Mesa

Search[Complete=NO]

# boxes	Order Date	Complete?	Client City
8	6/19/2013	NO	Irvine
21	6/20/2013	NO	Irvine
3	6/21/2013	NO	Costa Mesa

Search[Complete=NO $\wedge$ Data $<$ 6/21/2013]

# boxes	Order Date	Complete?	Client City
8	6/19/2013	NO	Irvine
21	6/20/2013	NO	Irvine

Projection

The **projection** operation takes a subset of the attributes and deletes all other attributes in each of the n-tuples. For example:

# boxes	Order Date	Complete?	Client City
8	6/19/2013	NO	Irvine
12	6/20/2013	YES	Huntington Beach
15	6/20/2013	YES	Huntington Beach
21	6/20/2013	NO	Irvine
3	6/21/2013	NO	Costa Mesa

Project[Order Date, Client City]

Order Date	Client City
6/19/2013	Irvine
6/20/2013	Huntington Beach
6/20/2013	Irvine
6/21/2013	Costa Mesa

Select[Complete=NO], Project[City]

Client City
Irvine
Costa Mesa

7 Computation

7.1 An introduction to algorithms

An algorithm is a step by step method for solving a problem. It usually includes:

- name
- brief description
- description of input
- description of output
- sequence of steps to follow

Algorithms are often described in **pseudocode**

Assignment operator

```
x := y
```

Return statement

```
Return(value)
```

If-else statement

```
If(x = 5), y := 7
```

```
If(condition)
  Step 1
  Step 2
  ...
  Step n
End-if
```

```
If(condition)
  Step(s)
Else
  Step(s)
End-if
```

For-loop

```
For i = s to t    <- first value is s, then s+1, until t is reached
  Step(s)
End-for
```

While-loop

```
While(condition)
  Step(s)
End-while
```

Nested Loops

```
Input: sequence a_1, ..., a_n; n

count := 0
For i = 1 to n-1
  For j = i+1 to n
    If(a_i = a_j) count := count+1
  End-for
End-for

Return(count)
```

7.2 Asymptotic growth of functions

Consider $f : \mathbb{Z}^+ \to \mathbb{R}^{\geq}$, where $\mathbb{R}^{\geq}$ denotes the set of non-negative real numbers. **Asymptotic growth** of a function f is a measure of how fast the object $f(n)$ grows as the input n grows. Classification of functions $\mathcal{O}, \Omega$, and Θ provide a way to concisely characterize the growth of a function.

$$f = \mathcal{O}(g) \quad "f \text{ is Oh of } g"$$

Constant factors

$$7n^3 \to 7 \text{ is constant factor}$$
$$5n^2 \to 5 \text{ is constant factor}$$
$$3 \to 3 \text{ is constant factor}$$

$\mathcal{O}$ notation

Let f and g be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$. Then $f = \mathcal{O}(g)$ if there is are positive real numbers c and n_0 such that for any $n \in \mathbb{Z}^+$ such that $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

Constants c and n_0 are said to be a *witness* to the fact $f = \mathcal{O}(g)$

Ω notation

Let f and g be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$. Then $f = \Omega(g)$ if there is are positive real numbers c and n_0 such that for any $n \in \mathbb{Z}^+$ such that $n \geq n_0$,

$$f(n) \geq c \cdot g(n)$$

$f = \Omega(g)$ is read "f is Omega of g"

Theorem: Relationship of $\mathcal{O}$-notation and Ω-notation

Let f and g be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$. Then $f = \Omega(g) \iff g = \mathcal{O}(f)$

Θ notation

Let f and g be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$.

$$f = \Theta(g) \text{ if} : f = \mathcal{O}(g) \wedge f = \Omega(g)$$

- $f = \Theta(g)$ is read "f is Theta of g"
- if $f = \Theta(g)$, then f is said to be the *order of* g.

Theorem: Asymptotic Growth of Polynomials

Let $p(n)$ be a degree-k polynomial of the form

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n + a_0 \text{ where } a_k > 0$$

Then $p(n)$ is $\Theta(n^k)$

Asymptotic Growth of Logarithm Functions with Different Bases

Let a and b be two real numbers greater than 1. Then

$$\log_a n = \Theta(\log_b n)$$

This is because of the fact that

$$\log_a n = \log_a b \cdot \log_b n, \text{ for } a, b > 1$$

when a function is said to be the $\mathcal{O}$ or Ω of a logarithm function, the base is often omitted because it is understood that as long as the base is greater than 1, the value of the base does not matter.

Growth rate of common functions

Constant Functions

- function that does not depend on n at all
- any constant function is $\Theta(1)$

Linear

- $\Theta(n)$

Function	Name
$\Theta(1)$	Constant
$\Theta(\log \log n)$	Log Log
$\Theta(\log n)$	Logarithmic
$\Theta(n)$	Linear
$\Theta(n \log n)$	$n \log n$
$\Theta(n^2)$	Quadratic
$\Theta(n^3)$	Cubic
$\Theta(n^m)$ for $m \in \mathbb{Z}^+$	Power
$\Theta(c^n)$ for $c > 1$	Exponential
$\Theta(n!)$	Factorial

Rules about Asymptotic Growth

Let f, g, and h be functions from $\mathbb{Z}^+$ to $\mathbb{R}^{\geq}$.

- if $f = \mathcal{O}(h)$ and $g = \mathcal{O}(h)$, then $f + g = \mathcal{O}(h)$
- if $f = \Omega(h)$ and $g = \Omega(h)$, then $f + g = \Omega(h)$
- if $f = \mathcal{O}(g)$, $c \cdot f = \mathcal{O}(g)$, $c \in \mathbb{R}^{\geq}$
- if $f = \Omega$, $c \cdot f = \Omega(g)$, $c \in \mathbb{R}^{\geq}$
- if $f = \mathcal{O}(g)$ and $g = \mathcal{O}(h)$, then $f = \mathcal{O}(h)$
- if $f = \Omega(g)$ and $g = \Omega(h)$, then $f = \Omega(h)$

7.3 Analysis of algorithms

Resources an algorithm requires to run

- time, called *time complexity*
- space, called *space complexity*
- Together called **computational complexity**

```
ComputeSum
Input: a_1, a_2, ..., a_n  (n is length of sequence)
Output: the sum of the numbers in the sequence
sum := 0                    | 1 assignment operation
For i = 1 to n              | loop iterated n times
  sum := sum + a_i          |   for loop test and increments (2 operations)
End-for                     |   1 addition and 1 assignment (2 operations)
Return(sum)                 | 1 op for return statement
```

$$\begin{aligned} f(n) &= 1 + n[2+2] + 1 \\ &= 1 + 4n + 1 \\ &= 4n + 2 \\ &= \mathcal{O}(n) \end{aligned}$$

Growth rates for different input sizes

$f(n)$	$n=10$	$n=50$	$n=100$	$n=1000$	$\cdots$
$\log_2 n$	$3.3\mu s$	$5.6\mu s$	$6.6\mu s$	$10\mu s$	$\cdots$
n	$10\mu s$	$50\mu s$	$100\mu s$	$1000\mu s$	$\cdots$
$n \log_2 n$	$.03ms$	$.28ms$	$.66ms$	$10ms$	$\cdots$
n^2	$.1ms$	$2.5ms$	$10ms$	$1s$	$\cdots$
n^3	$1ms$	$.125s$	$1s$	$16.7min$	$\cdots$
2^n	$1ms$	$35.7yrs$	$4 \times 10^{16} yrs$	$3.4 \times 10^{287} yrs$	$\cdots$

Worst-case analysis

Worst-case analysis evaluates the time complexity on the input which takes the longest time.

- upper bound: use $\mathcal{O}$-notation

  upper bound must apply for every input of size n
- lower bound: use Ω-notation

  lower bound need only apply for one possible input of n

Average-case analysis takes an average running time of algorithm on random inputs.

```
For(----)
  operations        -> linear (n)
End-for

For(----)
  For(----)
    operations      -> quadratic (n^2)
  End-for
End-for
```

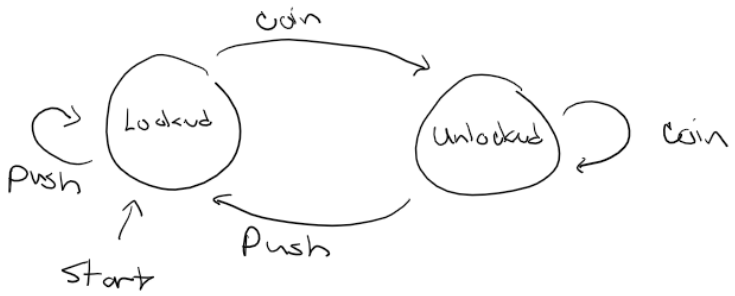
```
For(----)
  For(----)
    For(----)
      operations  -> cubic (n^3)
    End-for
  End-for
End-for
```

and so on. An algorithm runs in polynomial time if its time complexity is $\mathcal{O}^k$ for some fixed constant k. An algorithm is considered "efficient" if it runs in polynomial time. For example,

$$\mathcal{O}(n^5) \text{ is "efficient"}$$
$$\mathcal{O}(n^{\log n}) \text{ is not "efficient"}$$

7.4 Finite state machines

A **finite state machine** consists of a finite set of states, with transitions between states triggered by different input actions. A finite state machine is sometimes called *finite state automation*.



states: Q ={locked, unlocked}

The reaction of a finite state machine to the input received is denoted by a **transitive function**, often denoted by the symbol 'δ'

$$\delta([\text{state}], [\text{action}]) = [\text{state}]$$

In the case of the coin machine,

$$\delta(\text{Locked}, \text{Coin}) = \text{Unlocked}$$

State transition table:

- rows represent current state
- columns represent possible inputs
- each entry for a particular row and column indicate the new state resulting from that state/input combination

For example, the state transition table for the coin machine is

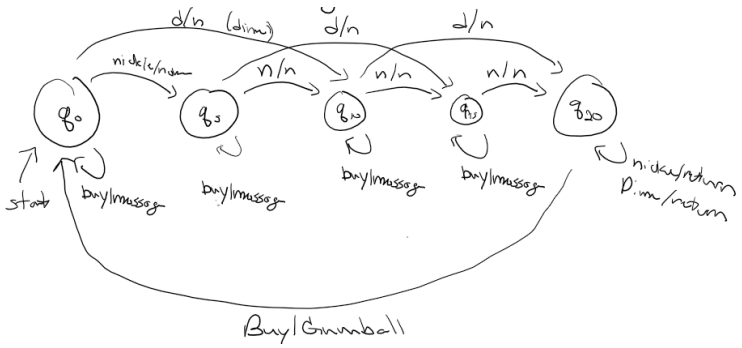
	Coin	Push
Locked	unlocked	locked
Unlocked	unlocked	locked

Components of a Finite State Machine

Notation	Description
Q	finite set of states
$q_0 \in Q$	q_0 is the start state
I	finite set of actions
$\delta : Q \times I \to Q$	transition function

FSM with Output

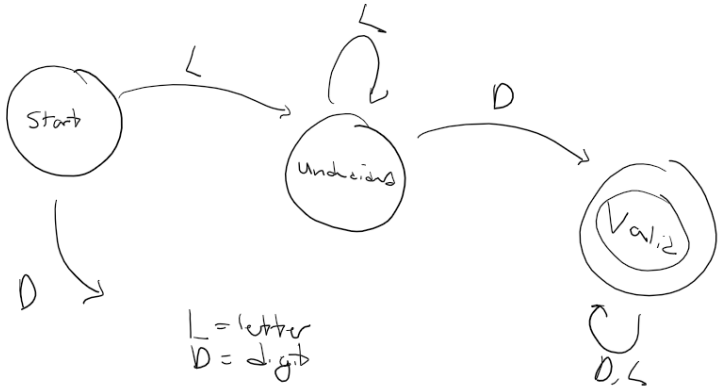
$$Q = \{q_0, q_5, q_{10}, q_{15}, q_{20}\}$$
$$I = \{\text{NICKLE}, \text{DIME}, \text{BUY}\}$$
$$O = \{\text{Gumball}, \text{Return}, \text{Message}, \text{None}\}$$



An accepted state is a state that is okay to end in.

$$A \subseteq Q, \text{ Accepted states are a subset of the total states}$$

Example, recognizing valid password



A valid password must begin with a letter and contain at least one digit.

7.5 Turing machines

FSMs are unable to solve even simple computational tasks such as determining whether a binary string has more 0's than 1's.

Church-Turing conjecture

Any problem that can be solved efficiently on any computing device can be solved efficiently by a Turing Machine.

Definition of a Turing Machine

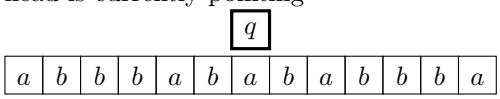
- memory is a 1-dimensional tape.

a	b	b	b	a	b	a	b	a	b	b	b	a

  example tape for $\{a, b, *\}$

- blank symbol (represented by a $*$ symbol)

- a configuration consists of the contents of the tape, the current state, and the tape cell to which the head is currently pointing



- action is determined by a transition function δ

Input to Turing Machine is the Input Alphabet, denoted by Σ, which much be a subset of the tape alphabet Γ

$$\Sigma \subset \Gamma$$

Components of a Turing Machine

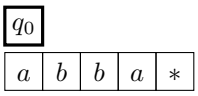
Notation	Description
Q	finite set of states
Γ	finite set of tape symbols
$\Sigma \subset \Gamma$	A subset of the tape symbols are input symbols
$q_0 \in Q$	q_0 is the start state
$q_{acc} \in Q$	q_{acc} is the accept state
$q_{rej} \in Q$	q_{rej} is the reject state
$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$	Transition Function

Additional Rules

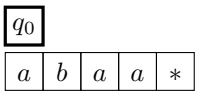
- if Turing machine reaches the accept state from a particular input x, the Turing machine **accepts** x
- if Turing machine reaches the reject state from a particular input x, the Turing machine **rejects** x
- if Turing machine *accepts* or *rejects* x, then the Turing machine **Halts** on x

Turing machine that accepts strings with 2 b's

	a	b	$*$
q_0	(q_0, a, R)	(q_1, b, R)	$(q_{rej}, *, L)$
q_1	(q_1, a, R)	(q_{acc}, b, R)	$(q_{rej}, *, L)$



↑ Halts and accepts

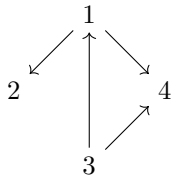


↑ Halts and rejects

Transition function for Turing machine that recognizes powers of 2:

-	a	x	$*$
q_0	$(q_{first}, *, R)$	$(q_{rej}, *, R)$	(q_{rej}, x, R)
q_{first}	(q_{even}, x, R)	(q_{first}, x, R)	$(q_{first}, *, R)$
q_{even}	(q_{odd}, a, R)	(q_{even}, x, R)	$(q_{first}, *, R)$
q_{odd}	(q_{even}, x, R)	(q_{odd}, x, R)	$(q_{first}, *, R)$
q_{ret}	(q_{ret}, a, R)	(q_{rej}, x, R)	$(q_{first}, *, R)$

7.6 Decision problems and languages



Symbol set	() , ; 0 1 2 3 4 5 6 7 8 9
Graph encoding	4; (1, 2)(1, 4)(3, 1)(3, 4)

Turing Machine can only accept or reject on input. This limits the class of problems answerable by a turing machine to **yes** or **no** problems.

- **Decision Problem**: given a boolean expression, is there an assignment to the boolean expression that causes the expression to evaluate to 1?
- **Search Problem**: given a boolean expression, find an assignment to the boolean expression that causes the expression to evaluate to 1 if one exists, or output that no such assignment exists.

If Σ is a finite alphabet, then a subset of Σ^* is called a *language* over Σ.

Language computed by a Turing Machine

Let Σ denote a finite alphabet and let L be a language over Σ. A turing machine M **computes language L**, or **decides language L** if for every $x \in \Sigma$, if $x \in L$, then M rejects x in a finite number of steps.

- **Time Complexity** is measured by how many steps taken by a Turing machine on a particular input.
- **Space Complexity** is measured by the number of tape cells that the turing machine uses in the course of it execution on a particular input.

A language is *incomputable* if there is no turing machine that computes the language.

8 Induction and Recursion

8.1 Sequences

A **sequence** is a special type of function in which the domain is the set of consecutive integers.

When a function is specified as a sequence, using subscripts to denote input is more common, so g_k is used instead of $g(k)$

A value g_k is called a **term**, and k is the *index* of g_k

For example:

$$g_1 = 3.67 \qquad g_2 = 2.88$$
$$g_3 = 3.25 \qquad g_4 = 3.75$$

$$g(k) = 3.67, 2.88, 3.25, 3.75$$

An entire sequence is denoted by $\{gk\}$, whereas g_k is used to denote a single term in the sequence.

A sequence commonly starts with 0 or 1, but it could be *any* integer.

Finite sequence

A sequence with a finite domain is a **finite sequence**. In a finite sequence, there is an *initial index* m and a *final index* n.

Infinite sequence

A sequence with an infinite domain is a **infinite sequence**. In an infinite sequence, there is an *initial index* m and the sequence is defined for indices $k \geq m$:

$$a_m, a_{m+1}, a_{m+2}, a_{m+3}, \ldots$$

A sequence can be specified by an **explicit formula**, such as $d_k = 2^k$ for $k \geq 1$.

$$\{d_k\} = 2, 4, 8, 16, \ldots$$

Increasing and Decreasing Sequences

- a sequence is *increasing* if for every two consecutive indices, k and $k+1$, $a_k < a_{k+1}$
- a sequence is *non-decreasing* if for every two consecutive indices, k and $k+1$, $a_k \leq a_{k+1}$

For example,

$2 < 4 < 5 < 6$ increasing *and* non-decreasing

$2 \leq 4 \leq 5 \leq 6$ non-decreasing *but* not increasing

The same relationship can be said for ***decreasing*** *and* ***non-increasing****.*

Geometric Sequences

A **geometric sequence** is a sequence of real numbers where each term is found by taking the previous term and multiplying it by a fixed number called the **common ratio**.

For example, with an *initial term*: 4, and *common ratio*: $\frac{1}{2}$,

$$4, 2, \frac{1}{2}, \frac{1}{4}, \ldots$$

Arithmetic Sequence

An **arithmetic sequence** is a sequence of real numbers where each term after the initial term is found by taking the previous term and adding a fixed number called the **common difference**.

For example, with an *initial value*: 2, and *common difference*: 3,

$$2, 5, 8, 11, \ldots$$

8.2 Recurrence relations

A rule that defines a term a_n as a function of previous terms in the sequence is called a **recurrence relation**

For example,

$$a_0 = a \text{ initial value}$$
$$a_n = d + a_{n-1}$$

Fibonacci Sequence:

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2} \text{ for } n \geq 2$$

A **dynamical system** is a system that changes over time. The state of the system at any point is determined by a set of well-defined rules that depend on the past states of the system.

8.3 Summations

Summation Notation is used to express the sum of terms in a numerical sequence.

$$\sum_{i=s}^{t} a_i = a_s + a_{s+1} + \ldots + a_t$$

i is the *index*
s is the *lower limit*
t is the *upper limit*

Any variable name could be used for index, but i, j, k and ℓ are most common.

$$\sum_{j=1}^{4} j^3 : \text{ summation form}$$
$$1^3 + 2^3 + 3^3 + 4^3 : \text{ expanded form}$$

$$\sum_{k=m}^{n} a_k = \sum_{k=m}^{n-1} a_k + a_n, \text{ for } n > m$$

$$\sum_{j=1}^{n} (j+2)^3 = \sum_{k=1}^{n} (k+2)^3 = \sum_{i=3}^{n+2} i^3$$

Closed Form

A **closed form** for a mathematical sum expresses the value of the sum without summation notation. For example,

$$\sum_{k=0}^{n-1}(a+kd) = an + \frac{d(n-1)n}{2}$$

Arithmetic Sequence Closed Form:

$$\sum_{k=0}^{n-1}(a+kd) = an + \frac{d(n-1)n}{2}$$

Geometric Sequence Closed Form:

$$\sum_{k=0}^{n-1}(a \cdot r^k) = \frac{a(r^n-1)}{r-1}$$

8.4 Mathematical induction

Two Components of an inductive proof:

- Base Case:

  establishes that the theorem is true for the first value in the sequence

- Inductive Step:

  establishes that if the theorem is true for k, then the theorem holds for $k+1$

  For a $k \in \mathbb{Z}^+$, $s(k) \implies s(k+1) \iff [s(1) \implies s(2) \implies s(3) \implies s(4) \implies \cdots]$

The supposition that $s(k)$ is true is called the inductive hypothesis.

8.5 More inductive proofs

Explicit formula for a sequence defined by a recurrence relation

$$\{g_n\}: \ g_0 = 1, \ g_n = 3g_{n-1} + 2n \text{ then for any } n \geq 0,$$
$$g_n = \frac{5}{2} \cdot 3^n - n - \frac{3}{2}$$

Proof. by induction on n.
Base Case: $n = 0$

$$g_o = \frac{5}{2} \cdot 3^0 - 0 - \frac{3}{2} = 1$$
$$g_0 = 1 \ \checkmark \quad \text{(by initial condition)}$$

Inductive Step: assume for any integer $k \geq 0$, assume that $g_k = \frac{5}{2} \cdot 3^k - k - \frac{3}{2}$ is true.
For $k+1$:

$$\begin{aligned} g_{k+1} &= 3g_k + 2(k+1) \quad \text{by definition} \\ &= 3(\frac{5}{2}3^k - k - \frac{3}{2}) + 2(k+1) \quad \text{by induction hypothesis} \\ &= \frac{5}{2}3^{k+1} - 3k - \frac{9}{2} + 2k + 2 \\ &= \frac{5}{2}3^{k+1} - k - \frac{5}{2} \\ &= \frac{5}{2}3^{k+1} - (k+1) - \frac{5}{2} \quad \text{by algebra} \end{aligned}$$

$\therefore g_n = \frac{5}{2} \cdot 3^n - n - \frac{3}{2}$, for all $n \in \mathbb{Z}^{\geq}$ □

8.6 Strong induction and well-ordering

The **principle of strong induction** assumes the fact to be proven holds for all values less than or equal to k and proves the fact holds for $k+1$

Inductive Step for Weak Induction

$$\begin{array}{lllll} & \text{For all } k \geq 1,\ S(k) & \implies & S(k+1) & \\ k=1 & S(1) & \implies & S(2) & \\ k=2 & & S(2) & \implies & S(3) \\ k=3 & & & S(3) & \implies \quad S(4) \\ \vdots & & & & \ddots \end{array}$$

Inductive Step for Strong Induction

$$\begin{array}{llllll} \text{For all } k \geq 1, & S(0) \wedge S(1) \wedge \ldots \wedge S(k) & \implies & S(k+1) & & \\ k=1 & S(1) & \implies & S(2) & & \\ k=2 & S(1) & \wedge & S(2) & \implies & S(3) \\ k=3 & S(1) & \wedge & S(2) & \wedge & S(3) \implies \quad S(4) \\ \vdots & & & & & \ddots \end{array}$$

for $n \geq 0,\ f_n \leq 2^n$

Proof. By Strong Induction of n

Base Case:

$$\begin{aligned} n = 0 &: \ f_0 = 0 \leq 2^0 \ \checkmark \\ n = 1 &: \ f_1 = 1 \leq 2^1 \ \checkmark \end{aligned}$$

Inductive Step:

For $k \geq 1$, suppose that for any j in the range from 0 through k, $f_k \leq 2^j$. We will prove that $f_{k+1} \leq 2^{k+1}$.

Since $k \geq 1$, then $k-1 \geq 0$. Therefore both k and $k-1$ fall in the range from 0 through k, and by the induction hypothesis $f_{k-1} \leq 2^{k-1}$ and $f_k \leq 2^k$

$$\begin{aligned} f_{k+1} &= f_k + f_{k-1} && \text{by definition} \\ &\leq 2^k + 2^{k-1} && \text{by inductive hypothesis} \\ &\leq 2^k + 2^{k-1} + 2^{k-1} && \text{since } 2^{k-1} \geq 0 \\ &\leq 2^k + 2 \cdot 2^{k-1} \\ &\leq 2^k + 2^k \\ &\leq 2 \cdot 2^k \\ f_{k+1} &\leq 2^{k+1} && \text{by algebra} \end{aligned}$$

$\therefore f_{k+1} \leq 2^{k+1}$ □

8.7 Loop invariants

Program Verification

- formally proving that programs perform correctly
- behavior is correct if:

  a pre-condition is true before the program starts

  a post-condition is true before the program ends

Loop Invariants for While Loops

A loop invariant is an assertion that is true before each iteration of a loop.

8.8 Recursive definitions

In a recursive definition of a function, the value of the function is defined in term of the output of the function on smaller input values.

For examples, the factorial:

$$\begin{aligned} n! &= f(n) \quad \text{such that} \\ f(0) &= 1 \\ f(1) &= n \cdot f(n-1) for n \geq 1 \end{aligned}$$

8.9 Structural induction

Structural induction is a type of induction used to prove theorem about recursively defined sets that follow the structure of the recursive definition.

For example, balanced parenthesis

$$\text{right}[\ ))()\ ] = 3 \qquad \text{left}[\ )))\ ] = 0$$
$$\text{left}[\ ((()))\ ] = 3$$

Recursive definition for the set of properly nested parenthesis:

Basis	$() \in P$
Recursive Rules	If $u, v \in P$, then
1	$(u) \in P$
2	$uv \in P$

Theorem: If string $x \in P$, then left[x] = right [x]

Proof. by structural induction

Base Case: $() \in P$

$$\text{left}[\ ()\ ] = \text{right}[\ ()\ ] = 1$$

Inductive Step: If $x \in P$, then x was constructed by apply a sequence of recursive rules starting with the string () given in the basis.

We consider two case, depending on the last recursive rule that was applies to construct x.

Case 1 Rules 1 was the last applied to construct x. Then $x = (u)$, where $u \in P$. We assume that left[u] = right [u].

$$\begin{aligned} \text{left}[\ x\ ] &= \text{left}[\ (u)\ ] && \text{because } x = (u) \\ &= 1 + \text{left}[\ u\ ] && (u) \text{ has one more (than } u \\ &= 1 + \text{right}[\ u\ ] && \text{by the inductive hypothesis} \\ &= \text{right}[\ (u)\ ] && (u) \text{ has one more) than } u \\ &= \text{right}[\ x\ ] && \text{because } x = (u) \end{aligned}$$

Case 2 Rule 2 was the last rule applied to construct x. Then $x = uv$, where $u, v \in P$. We assume that left[u] = right [u] and left[v] = right [v].

$$\begin{aligned} \text{left}[\ x\ ] &= \text{left}[\ uv\ ] && \text{because } x = uv \\ &= \text{left}[\ u\ ] + \text{left}[\ v\ ] \\ &= \text{right}[\ u\ ] + \text{right}[\ v\ ] && \text{by inductive hypothesis} \\ &= \text{right}[\ uv\ ] \\ &= \text{right}[\ x\ ] && \text{because } x = uv \end{aligned}$$

$\therefore \text{left}[\ x\ ] = \text{right}[\ x\ ]$ □

8.10 Recursive algorithms

A **recursive algorithm** is an algorithm that calls itself.

Ex. a recursive algorithm to compute the factorial formula:

```
Factorial(n)

Input: A non-negative integer n
Output: n!

If(n=0), Return(1)
r := Factorial(n-1) //the recursive call

Return(r*n)
```

Ex. recursive algorithm to compute the powerset of a set:

```
PowerSet(A)

Input: set A
Output: the powerset of A

If(A=[emptyset]), return {[emptyset]}

Select an element a in A
A' := A - {a}
P := PowerSet(A') //recursive call
P' := P
For each S in P'
  Add (S union {a}) to P
End-for

Return(P)
```

8.11 Induction and recursive algorithms

Nothing very useful in this section, only kept for counting reasons.

8.12 Analyzing the time complexity of recursive algorithms

The **time complexity** of an algorithm is a function $T(n)$, whose n is the input size

Ex. Determining recurrence relation for factorial(n)

```
Factorial(n)

If(n=0), Return(1)      // two operations, comparison and returning

r := Factorial(n-1)     // n-1 recursive calls, 1 operation for assignment

Return(r*n)             // 2 ops, multiplying and returning
```

Therefore, $T(0) = 2$, $T(n) = T(n-1) + 5$

8.13 Divide-and-conquer algorithms: Introduction and mergesort

A **divide-and-conquer** algorithm solves a problem by recursively breaking the original input into smaller sub-problems of roughly equal size

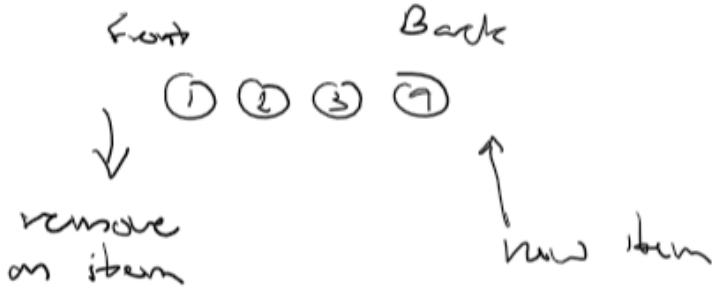
Ex.

```
Findmin(L)

If L has only one item x, return(x)

Break list L into two lists, A and B

a := Findmin(A)
b := Findmin(B)

If(a <= b) Return(a)
Else Return(B)
```

Queue

A **queue** maintains items in an ordered list

- new items are added to back of queue
- items are removed according to first-in, first-out



8.14 Divide-and-conquer algorithms: Binary Search

- A **search algorithm** finds a target in a list.
- **Binary Search** is an efficient algorithm to search for a target item in a *sorted list*.

Pseudo-code for recursive binary search:

```
RecBinSearch(low, high, A, x)

If(low = high AND a[low] = x), Return(low)
If(low = high AND a[low] != x), Return(-1)

mid := floor((low + high)/2)

If(x <= a[mid]), then high := mid
If(x > a[mid]), then low := mid+1

Return(RecBinSearch(low, high, A, x))
```

8.15 Solving linear homogeneous recurrence relations

Finding an explicit formula for a recursively defined sequence is called *solving a recurrence relation.*

Linear Homogeneous Recurrence Relation

A linear homogeneous recurrence relation of degree k has the following form:

$$f_n = c_1 f_{n-1} + c_2 f_{n-2} + \cdots + c_k f_{n-k},$$

where the c_j's are constants that do not depend on n, and $c_k \neq 0$.

Examples illustrating linear and non-linear recurrence relations

$b_n = b_{n-1} + (b_{n-2} \cdot b_{n-3})$	Non-linear
$c_n = 3n \cdot c_{n-1}$	Non-linear
$d_n = d_{n-1} + (d_{n-2})^2$	Non-linear
$f_n = 3f_{n-1} - 2f_{n-2} + f_{n-4} + n^2$	Linear, degree 4. Non-homogeneous
$g_n = g_{n-1} + g_{n-3} + 1$	Linear, degree 3. Non-homogeneous
$h_n = 2h_{n-1} - h_{n-2}$	Linear, degree 2. Homogeneous
$s_n = 2s_{n-1} = \sqrt{3}s_{n-5}$	Linear, degree 5. Homogeneous

The expression denoting the infinite set of solutions to a recurrence relation without initial values is called the *general solution* to the recurrence relation.

Solutions to linear homogeneous recurrence relations

If g_n and h_n satisfy a linear homogeneous relation then so does $f_n = s \cdot g_n + t \cdot h_n$ for any real numbers s and t.

Using Initial Values to determine to unique solution to a recurrence relation

Any $f_n = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^n + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^n$

Initial Values:

$$n = 0 \qquad f_0 = 0 = c_1 \left(\frac{1+\sqrt{5}}{2}\right)^0 + c_2 \left(\frac{1-\sqrt{5}}{2}\right)^0 = c_1 + c_2 = 0$$

$$n = 1 \qquad f_1 = 1 = c_1 \left(\frac{1+\sqrt{5}}{2}\right) + c_2 \left(\frac{1-\sqrt{5}}{2}\right) = 1$$

Then solving a two variable linear system of equations:

$$c_1 = \frac{1}{\sqrt{5}}, \qquad c_2 = -\frac{1}{\sqrt{5}}$$

which then yields the specific solution of

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2}\right)^n.$$

General Steps

1. Use the recurrence relation to find the characteristic equation which will be $p(x) = 0$, where $p(x)$ is a degree d polynomial.
2. Find all solutions to the characteristic equation. Assume $p(x)$ has distinct roots $x_1, x_2, \ldots, x_d$.
3. Every solution $f_n = (x_1)^n$ satisfies the recurrence relation. Therefore, any solution of the form $f_n = a_1(x_1)^n + \cdots + a_d(x_d)^n$ satisfies the recurrence relation.

4. Each initial value gives a value of f_n for a specific value for n. Plug the value for n and f_n into the above expression to get a linear equation with variables $a_1, a_2, \ldots, a_d$. There are d linear equations (from the d initial values) and d variables.

5. Solve for $a_1, a_2, \ldots, a_d$.

6. Plug values for $a_1, a_2, \ldots, a_d$ back in to the expression for f_n to get the closed form expression for f_n.

8.16 Solving linear non-homogeneous recurrence relations

A **non-homogeneous linear recurrence relation** is a linear recurrence relation with additional terms that are either a constant or a function of n. The recurrence relations below are all examples of homogeneous linear recurrence relations.

$$f_n = 3f_{n-1} + 10f_{n-2} + 2$$
$$f_n = 3f_{n-1} + 10f_{n-2} + 24n$$
$$f_n = 3f_{n-1} + 10f_{n-2} + 3n$$

The **associated homogeneous recurrence relation** is the recurrence relation with the additional non-homogeneous terms dropped. For example the associated homogeneous recurrence relation for all of the recurrence relations give above is $f_n = 3f_{n-1} + 10f_{n-2}$.

Particular and homogeneous solutions

The solution to a non-homogeneous linear recurrence relation is the sum of two parts: a *homogeneous solution* plus a *particular solution*. If the sequence is $\{f_n\}$, then the homogeneous solution is denoted by $f_n^{(h)}$ and the particular solution is denoted by $f_n^{(p)}$.

- The homogeneous solution $f_n^{(h)}$ is the general solution to the associated homogeneous recurrence relation.
- The method for finding the particular solution $f_n^{(p)}$ is to guess the form of the particular solution and then check the guess.

Summary of all the steps

1. Find the homogeneous part of the solution which is the general solution to the associated homogeneous recurrence relation.

2. Guess the correct form for the particular solution.

3. Verify the guess by solving for constants so that the guess satisfies the recurrence relation for all n.

4. Add the homogeneous and particular solutions to get the general solution.

5. For a degree d linear recurrence relation, there must be d initial values to specify the sequence. Each initial value gives a value for f_n for a specific value for n. Plug the values for n and f_n into the general solution to get a linear equation with variables $a_1, a_2, \ldots, a_d$. There are d linear equations (from the d initial values) and d variables.

6. Solve for $a_1, a_2, \ldots, a_d$.

7. Plug values for $a_1, a_2, \ldots, a_d$ back in to the general solution for f_n to get the final closed form expression for f_n.

8.17 Divide-and-conquer recurrence relations

Deriving solutions to divide-and-conquer recurrence relations

Many recurrence relations that describe the time complexity of divide-and-conquer algorithms have the form

$$T(n) = aT(n/b) + \Theta(n^d).$$

Expressing the formula as a geometric sum

The explicit formula for $T(n)$ can be generalized for a recurrence relation of the form $T(n) = aT(n/b) + n^d$ which has an explicit formula

$$T(n) = n^d \cdot \sum_{j=0}^{L} \left(\frac{a}{b^d}\right)^j \quad \text{where } L = \log_b n$$

The expression for n^d times the sum of terms in a geometric sequence of the form $1 + r + r^2 + \cdots + r^L$, where the ratio $r = a/b^d$. The formula for the sum of a geometric sequence is

$$1 + r + r^2 + \cdots + r^L = \sum_{j=0}^{L} r^j = \frac{r^{L+1} - 1}{r - 1}$$

The asymptotic growth rate of $T(n)$ depends on whether the ratio r is less than, greater than, or equal to 1.

The Master Theorem

if $T(n) = aT(n/b) + \Theta(n^d)$ for constants $a > 0,\ b > 1,$ and $d \geq 0$, then

- If $(a/b^d) < 1$, then $T(n) = \Theta(n^d)$
- If $(a/b^d) = 1$, then $T(n) = \Theta(n^d \log n)$
- If $(a/b^d) > 1$, then $T(n) = \Theta(n^{\log_b a})$

9 Integer Properties

9.1 The Division Algorithm

In **integer division**, the input and output values must always be integers. For example, when 9 is divided by 4, the answer is 2 with a remainder of 1, instead of 2.25.

Divides

Let x and y be two integers. Then x *divides* y, $x \mid y$, if and only if $x \neq 0$ and there is an integer k such that $y = kx$. If there is no such integer or if $x = 0$, then x does not divide y, $x \nmid y$. If $x \mid y$, then y is said to be a *multiple* of x, and x is a *factor* or *divisor* of y.

Theorem: Divisibility and linear combinations

Let $x, y,$ and z be integers. If $x \mid y$ and $x \mid z$, then $x \mid (sy + tz)$ for any integers s and t.

Proof. Since $x \mid y$, then $y = kx$ for some integer k. Similarly, since $x \mid z$, then $z = jx$ for some integer j. A linear combination of y and z can be expressed as:

$$sy + tz = s(kx) + t(jx) = (sk + tj)x.$$

For some integers s and t. Since $sy + tz$ is an integer multiple of x, then $x \mid (sy + tz)$. □

Quotients and remainders

If $x \nmid y$, then there is s non-zero remainder when x is divided into y. The **Division Algorithm**, states that the result of the division and the remainder are unique.

Theorem: The Division Algorithm

Let n be an integer and let d be a positive integer. Then, there are unique integers q and r, with $0 \leq r \leq d-1$, such that $n = qd + r$.

Integer division definitions

In the Division Algorithm, q is called the **quotient** and r is called the **remainder**. The operations **div** and **mod** produce the quotient and the remainder as a function of n and d.

$$q = n \operatorname{div} d$$
$$r = n \bmod d$$

Here is some examples of computing div and mod:

$$15 \bmod 6 = 3 \qquad -11 \bmod 4 = 1$$
$$15 - 2 \cdot 6 = 3 \qquad -11 - (-3) \cdot 4 = 1$$

$$15 \operatorname{div} 6 = 2 \qquad -11 \operatorname{div} 4 = -3$$
$$\frac{15 - 3}{2} = 2 \qquad \frac{-11 - 1}{4} = -3$$

9.2 Modular arithmetic

Given a finite set of integers, we can define addition and multiplication on the elements in the set such that after every operation, we apply a modular function equal to the cardinality of the set.

- **addition mod** m

  the operation defined by adding two numbers and applying $\bmod m$ to the result

- **multiplication mod** m

  the operation defined by multiplying two numbers and applying $\bmod m$ to the result

The set $\{0, 1, 2, \ldots, m-1\}$ along with addition and multiplication mod m defines a closed mathematical system with m elements called a **ring**. The ring $\{0, 1, 2, \ldots, m-1\}$ with addition and multiplication mod m is denoted by $\mathbb{Z}_m$.

Applications

A common way to organize data is to maintain an array called a **hash table** which is slightly larger than the number of data items to be stored. A bldhash function is used to map each data item to a location in the array. Modulus is used to keep the results from a hash function in the range of the hash table.

Computers use functions called bldpseudo-random number generators that produce numbers having many of the statistical properties of random numbers but are in fact deterministically generated. Modulus is used to keep these pseudo-random number generators in a certain range when used.

Congruence mod m

Let $m \in \mathbb{Z} > 1$. Let x and y be any two integers. Then bld$x$ is congruent to $y \bmod m$ if $x \bmod m = y \bmod m$. The fact that x is congruent to $y \bmod m$ is denoted

$$x \equiv y \pmod{m}.$$

Theorem: Alternate characterization of congruence mod m

Let $m \in \mathbb{Z} > 1$. Let x and y be any two integers. Then $x \equiv y \pmod{m}$ if and only if $m \mid (x - y)$.

Proof. First suppose that $x \equiv y \pmod{m}$. By definition $xtmodm = y \bmod m$. We define the variable r to be the value of $x \bmod m = y \bmod m$. Therefore, $x = r + km$ for some integer k and $y = r + jm$ for some integer j. Then

$$x - y = (r + km) - (r + jm) = (k - j)m.$$

Since $(k - j)$ is an integer, $m \mid (x - y)$.
Now suppose that $m \mid (x - y)$. Then $(x - y) = tm$ for some integer t. Let r be the value of $x \bmod m$. Then $x = r + km$ for some integer k. The integer y can be expressed as

$$y = x - (x - y) = (r + km) - tm = r + (k - t)m.$$

Since r is an integer in the range from 0 to $m - 1$, r is the unique remainder when y is divided by m. Therefore $r = y \bmod m = x \bmod m$, and by definition $x \equiv y \pmod{m}$. □

Precedence of the mod operation

$$6 + 2 \bmod y = 6 + (2 \bmod 7) = 8$$
$$6 \cdot 2 \bmod 7 = (6 \cdot 2) \bmod 7 = 5$$

However, in general it is best to just use parentheses in order to clarify which operations should be performed first.

Theorem: Computing arithmetic operations mod m

Let m be an integer larger than 1. Let x and y be any integers. Then

$$[(x \bmod m) + (y \bmod m)] \bmod m = [x + y] \bmod m$$
$$[(x \bmod m)(y \bmod m)] \bmod m = [x \cdot y] \bmod m$$

9.3 Prime factorizations

A number p is **prime** if it is an integer greater than 1 and its only factors are 1 and p. A positive integer is **composite** if it has a factor other than 1 or itself. Every integer greater than 1 is either prime of composite. Every positive integer greater than one can be expressed as a product of primes called its **prime factorization**. Moreover, the prime factorization is unique up to ordering of the factors.

Theorem: The Fundamental Theorem of Arithmetic

Every positive integer other than 1 can be expressed uniquely as a product of prime numbers where the primes factors are written in non-decreasing order.

Examples of prime factorizations in non-decreasing order

$$112 = 2^4 \cdot 7$$
$$612 = 2^2 \cdot 3^3 \cdot 17$$
$$243 = 3^5$$
$$17 = 17$$

Greater common divisors and least common multiples

- The **greatest common divisor (gcd)** of integers x and y that are not both zero is the largest integer that is a factor of both x and y.
- The **least common multiples (lcm)** of non-zero integers x and y is the smallest positive integer that is an integer multiple of both x and y.

Two numbers are **relatively prime** if their greatest common divisor is 1.

Theorem: GCD and LCM from prime factorizations

Let x and y be two positive integers with prime factorizations expressed using a common set of primes as:

$$x = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdots p_r^{\alpha_r}$$
$$y = p_1^{\beta_1} \cdot p_2^{\beta_2} \cdots p_r^{\beta_r}$$

The p_i's are all distinct prime numbers. The exponents α_i's and β_i's are non-negative integers. Then:

- $x \mid y$ if and only if $\alpha_i \leq \beta_i$ for all $1 \leq i \leq r$
- $\gcd(x, y) = p_1^{\min(\alpha_1, \beta_1)} \cdot p_2^{\min(\alpha_2, \beta_2)} \cdots p_r^{\min(\alpha_r, \beta_r)}$
- $\text{lcm}(x, y) = p_1^{\max(\alpha_1, \beta_1)} \cdot p_2^{\max(\alpha_2, \beta_2)} \cdots p_r^{\max(\alpha_r, \beta_r)}$

9.4 Factoring and primality testing

A **brute force algorithm** solves a problem by exhaustively searching all positive solutions without using an understanding of the mathematical structure in the problem to eliminate steps.

Theorem: Small Factors

If N is a composite number, then N has a factor greater than 1 and at most $\sqrt{N}$

Theorem: Infinite number of primes

There are an infinite number of primes.

Proof. Suppose that there are a finite number of primes. Since there are only a finite number, they can be listed:

$$p_1, p_2, \ldots, p_k$$

Take the product of all the primes and add 1. Call the resulting number N:

$$N = (p_1 \cdot p_2 \cdots p_k) + 1$$

The number N is larger than all of the primes numbers that were listed, so it must not be prime. Since N is a composite number, it is the product of at least two primes by the Fundamental Theorem of Arithmetic. There N is divisible by some prime p_j. Let

$$\frac{N}{p_j} = \frac{(p_1 \cdot p_2 \cdots p_k)}{p_j} + \frac{1}{p_j}$$

Note that p_j is one of the prime factors in $(p_1 \cdot p_2 \cdots p_k)$, so $(p_1 \cdot p_2 \cdots p_k)/p_j$ is an integer. However, $1/p_j$ is not an integer. Since N/p_j is the sum of two terms, one of which is an integer and the other of which is not an integer, then N/p_j is not an integer. This contradicts the fact that p_j evenly divides N. □

The Prime Number Theorem

Let $\pi(x)$ be the number of prime numbers in the range from 2 through x. Then

$$\lim_{x \to \infty} \frac{\pi(x)}{x/\ln x} = 1.$$

Another way to interpret the Prime Number Theorem is that if a random number is selected from the range 2 to x, then the likelihood that the selected number is prime is roughly $1/\ln x$.

9.5 Greatest common factor divisor and Euclid's algorithm

There is an efficient way to compute the gcd of two numbers without finding their prime factorizations. The algorithm presented in this subsection is in wide use today and is attributed to the Greek mathematician Euclid who lived around 300 B.C. The basis of the algorithm is the following theorem:

GCD Theorem

Let x and y be two positive integers. Then $\gcd(x, y) = \gcd(y \bmod x, x)$.

Euclid's Algorithm for finding the greatest common divisor

```
Input: Two positive integers, x and y.
Output: gcd(x, y)

If(y < x)
  Swap x and y.
r := y mod x

While(r != 0)
  y := x
```

```
   x := r
   r := y mod x
End-While

Return(x)
```

Sample execution of Euclid's algorithm for $\gcd(675, 210)$:

675	210	$675 \bmod 210$		
	210	45	$210 \bmod 45$	
		45	30	$45 \bmod 30$
			30	$30 \bmod 15$
				15 0

The last non-zero number was 15, so $\gcd(675, 210) = 15$.

Expressing $\gcd(x, y)$ as a linear combination of x and y

Let x and y be integers, then there are integers s and t such that

$$\gcd(x, y) = sx + ty.$$

The values for s and t in the theorem above can be found by a series of substitutions using the equation from each iteration. The algorithm used to find the coefficient, s and t, such that $\gcd(x, y) = sx + ty$, is called the **Extended Euclidean Algorithm**.

The Extended Euclidean Algorithm

		y	x	r
675	210	45	30	15
				$r = y \bmod x$
				$r = y - (y \operatorname{div} x) \cdot x$
				$15 = 45 - (45 \operatorname{div} 30) \cdot 30$
				$\mathbf{15 = 45 - 1 \cdot 30}$
			30	$= 210 - (210 \operatorname{div} 45) \cdot 45$
			30	$\mathbf{= 210 - 4 \cdot 45}$
		45	=	$675 - (675 \operatorname{div} 210) \cdot 210$
		45	=	$\mathbf{675 - 3 \cdot 210}$

We can use the bolded equations to solve for $15 = c \cdot 210 + d \cdot 675$.

$$\begin{aligned} 15 &= 45 - 30 \\ &= 45 - (210 - 4 \cdot 45) \\ &= 5 \cdot 45 - 210 \\ &= 5 \cdot (675 - 3 \cdot 210) - 210 \\ &= 5 \cdot 675 - 16 \cdot 210 \end{aligned}$$

Now we have the full answer and expansion for $\gcd(675, 210)$.

$$\gcd(675, 210) = 15 = 5 \cdot 675 - 16 \cdot 210.$$

The Multiplicative Inverse mod n

A **multiplicative inverse mod** n, or just **inverse mod** n, of an integer x, is an integer $s \in \{1, 2, \ldots, n-1\}$ such that $sx \bmod n = 1$.

For example, 3 is an inverse of 7 mod 10 because $3 \cdot 7 \bmod 10 = 1$. The number 7 is an inverse of 5 mod 17 because $7 \cdot 5 \bmod 17 = 1$. It is possible for a number to be its own multiplicative inverse mod n. For example, 7 is the inverse of $7 \bmod 8$ because $7 \cdot 7 \bmod 8 = 1$.

Not every number has an inverse mod n. For example, 4 does not have an inverse mod 6. The condition is that x has an inverse mod n if and only if x and n are relatively prime.

The Extended Euclidean Algorithm can be used to find the multiplicative inverse of $x \bmod n$ when is exists.

- If $\gcd(x, n) \neq 1$, then x does not have a multiplicative inverse mod n.
- If x and n are relatively prime, then the Extended Euclidean Algorithm finds integers s and t such that $1 = sx + tn$.
- $sx - 1 = -tn$. Therefore, $(sx \bmod n) = (1 \bmod n)$. If $A - B$ is a multiple of n then $(A \bmod n) = (B \bmod n)$.
- $(s \bmod n)$ is the unique multiplicative inverse of x in $\{0, 1, \ldots, n-1\}$.

For example, suppose that Euclid's Algorithm returns

$$\gcd(31, 43) = 1 = 13 \cdot 34 - 18 \cdot 31$$

The coefficient of 31 is -18. Therefore, the multiplicative inverse of $31 \bmod 43$ is $(-18 \bmod 43) = 25$.

9.6 Number representation

A digit in binary is called a **bit**. In binary notation, each place value is a power of 2. Numbers represented in **base** b require b distinct symbols and each place value is a power of b.

Theorem: Number representation

For an integer $b > 1$. Every positive integer n can be expressed uniquely as

$$n = a_k \cdot b^k + a_{k-1} \cdot b^{k-1} + \cdots + a_1 \cdot b^1 + a_0 \cdot b^0,$$

where k is a non-negative integer, and each a_i is an integer in the range from 0 to $b-1$, and $a_k \neq 0$. The representation of n base b is called the **base b expansion of** n and is denoted by $(a_k a_{k-1} \ldots a_1 a_0)_b$.

Hexadecimal Numbers

In **hexadecimal** notation (or **hex** for short), numbers are represented in base 16. Typically, the set of symbol, in order of value, is

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}.$$

Additionally, here are the first 15 hexadecimal digits and correspond encodings in decimal and binary.

Hex	0	1	2	3	4	5	6	7
Decimal	0	1	2	3	4	5	6	7
Binary	0	1	10	11	100	101	110	111

Hex	8	9	A	B	C	D	E	F
Decimal	8	9	10	11	12	13	14	15
Binary	1000	1001	1010	1011	1100	1101	1110	1111

Since both hexadecimal and binary are powers of 2, there is an easy way to translate between the binary expansion and the hexadecimal expansion of a number. Groups of 4 binary digits can be directly translated into hexadecimal digits. Here is an example

1	1101	0101	1110	1000
1	D	5	E	8

$$1, 1101, 0101, 1110, 1000_2 = 1D5E8_{16}$$

Hexadecimal notation is particularly useful in computer science because each hexadecimal digit can be used to represent a 4-bit binary number. A byte, which consists of 8 bits, can be represented by a 2-digit hexadecimal number. Two hexadecimal digits is easier for a human to recognize and remember than 8 bits.

Converting decimal numbers to base b

$$[\text{Base } b \text{ expansion of } (n \operatorname{div} b)]\,[n \bmod b]$$

Here is an example of 1161_{10} converted to base 7:

		$\ulcorner$	1161
		$1161 \operatorname{div} 7$	$1161 \bmod 7$
	$\ulcorner$	165	6
	$165 \operatorname{div} 7$	$165 \bmod 7$	|
$\ulcorner$	23	4	|
$23 \operatorname{div} 7$	$23 \bmod 7$	|	|
3	2	|	|
3	2	4	6

The number of digits required to represent a number

$$n = (\underbrace{[b-1][b-1]\ldots[b-1]}_{k\text{digits}})_b = (1\underbrace{00\ldots0}_{k0's})_b - 1 = b^k - 1$$

The number of digits required to expressed the base b expansion of a positive integer n is $\lceil \log_b(n+1) \rceil$. For example, the number of digits required to express the base 2 expansion of 13 is:

$$\lceil \log_2(13+1) \rceil = \lceil \log_2(14) \rceil = \lceil 3.8 \rceil = 4.$$

9.7 Fast exponentiation

Computing a power of x by repeated multiplication by x

$$p := x \xrightarrow{p:=p*x} p = x^2 \xrightarrow{p:=p*x} p = x^3 \xrightarrow{p:=p*x} p = x^4 \cdots$$

Computing a power of x by repeated squaring

$$p := x \xrightarrow{p:=p*p} p = x^2 \xrightarrow{p:=p*p} p = x^4 \xrightarrow{p:=p*p} p = x^8 \cdots$$

In order to compute x^y, examine the binary expansion of y:

$$y = a_k \cdot 2^k + a_{k-1} \cdot 2^{k-1} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0$$

We can use this expanded form of y to represent x as a sum of perfect squares

$$x^y = x^{a_k \cdot 2^k + a_{k-1} \cdot 2^{k-1} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0} = x^{a_k \cdot 2^k} \cdot x^{a_{k-1} \cdot 2^{k-1}} \cdots x^{a_1 \cdot 2^1} \cdot x^{a_0 \cdot 2^0}$$

Each coefficient a_j is either 0 or 1. If $a_j = 0$ for some j, then

$$x^{a_j \cdot 2^j} = x^0 = 1$$

and the corresponding factor can be omitted from the product. Here is an example, computing 7^{11}.

$$\begin{aligned} 7^{11} &= 7^{[(1\cdot 2^3)+(0\cdot 2^2)+(1\cdot 2^1)+(1\cdot 2^0)]} \\ &= 7^{(1\cdot 8)} \cdot 7^{(0\cdot 4)} \cdot 7^{(1\cdot 2)} \cdot 7^{(1\cdot 7)} \\ &= 7^8 \cdot 1 \cdot 7^2 \cdot 7^1 \\ 7^1 1 &= 5,764,801 \cdot 49 \cdot 7 \\ &= 1,977,326,743 \end{aligned}$$

An iterative algorithm for fast exponentiation

```
Input: Positive integers x and y.
Output: x^y

p := 1  // p holds the partial result
s := x  // s holds the current x^{2^j}
r := y  // r is used to compute the binary expansion of y

while(r > 0)
  if(r mod 2 = 1)
    p := p * s
  s := s * s
  r := r div 2
End-While

Return(p)
```

An iterative algorithm for fast modular exponentiation

```
Input: Positive integers x, y and n.
Output: x^y mod n

p := 1  // p holds the partial result
s := x  // s holds the current x^{2^j}
r := y  // r is used to compute the binary expansion of y

while(r > 0)
  if(r mod 2 = 1)
    p := p * s mod n
  s := s * s mod n
  r := r div 2
End-While

Return(p)
```

9.8 Introduction to cryptography

Cryptography is the science of protecting and authenticating data and communication. Cryptography is ubiquitous in the electronic age in which sensitive information such as credit card numbers and passwords are sent over the internet on a daily basis.

A cryptosystem is a system by which a **sender** sends a message to a **receiver**. The sender **encrypts** the message so that if an eavesdropper learns the transmitted message, they will be unable to recover the original message. The unencrypted message is called **plaintext** and the encrypted message is called the **ciphertext**. The receiver must have a **secret key** that allows him to **decrypt** the ciphertext to obtain the original plaintext.

Sending an Encrypted Text Message via a Secret Key

1. Alice wants to send the message "MEET AT DAWN" to Bob. Alice converts the text message to the number 13050520270129270401231 4.

2. Alice encrypts the numerical message with her copy of the secret key.

3. Alice sends encrypted message to Bob. Eve cannot read the encrypted message without the secret key.
4. Bob decrypts the encrypted message using the secret key to get 130505202701292704012314 which he then converts to "MEET AT DAWN".

Encryption and Decryption Functions

Consider a simple cryptosystem in which the set of all possible plaintexts come from $\mathbb{Z}_N$ for some integer N. Alice and Bob share a secret number $k \in \mathbb{Z}_N$. The security of their encryption scheme rests on the assumption that no one besides them knows the number k. To encrypt a plaintext $m \in \mathbb{Z}_n$, Alice computes:

$$c = (m + k) \bmod N \qquad \text{(encryption)}$$

Alice sends the ciphertext c to Bob. When Bob receives the ciphertext c, he decrypts c as follows:

$$m = (c - k) \bmod N \qquad \text{(decryption)}$$

The simple encryption scheme presented here is an example of symmetric key cryptography. In a **symmetric key cryptosystem**, Alice and Bob must meet in advance to decide on the value of a shared secret key.

Simple encryption scheme requirements

- If $m \neq m'$ and $m,\ m' \in \mathbb{Z}_N$ then $(m + k) \bmod N \neq (m' + k) \bmod N$ (no two distinct plaintexts map to same ciphertext).
- If $m \in \mathbb{Z}_N$ then $(((m+k) \bmod N) - k) \bmod N = m$ (decryption scheme is inverse of encryption scheme).

However, this encryption method is not terribly secure, and is not used in real world conditions because better methods exist.

9.9 The RSA cryptosystem

In **public key cryptography**, Bob has an **encryption key** that he provides *publicly* so that anyone can use it to send him an encrypted message. Bob holds a matching **decryption key** that he keeps *privately* to decrypt messages. While anyone can use the public key to encrypt a message, the security of the scheme depends on the fact that it is difficult to decrypt the message without having the matching private decryption key.

1. Alice encrypts her message to Bob using the public key.
2. Alice sends the encrypted message to Bob. Eve cannot read the encrypted message.
3. Bob decrypts the message using his private key.

Preparation of public and private keys in RSA

1. Bob selects two large prime numbers, p and q.
2. Bob computes $N = pq$ and $\phi = (p - 1)(q - 1)$.
3. Bob finds an integer e such that $\gcd(e, \phi) = 1$.
4. Bob computes the multiplication inverse of $e \bmod \phi$: an integer d such that $(ed \bmod \phi) = 1$.
5. Public (encryption) key: N and e.
6. Private (decryption) key: d.

The RSA scheme requires that m, the message, is an integer in $\mathbb{Z}_N$ and is not a multiple of p or q. Since p and q are primes with hundreds of digits, it is extremely unlikely that m is a multiple of primes p or q. Alice encrypts her plaintext using e and N to produce ciphertext c as follows:

$$c = m^e \bmod N \qquad \text{(encryption)}$$

Alice transmits c to Bob. Bob decrypts the ciphertext using d to recover m from c:

$$m = c^d \bmod N \qquad \text{(decryption)}$$

Number theoretic fact to establish correctness of RSA

Let p and q be prime numbers and $pq = N$. Suppose that $m \in \mathbb{Z}_n$ and $\gcd(m, N) = 1$. Then $m^{(p-1)(q-1)} \bmod N = 1$.

10 Introduction to Counting

10.1 Sum and Product Rules

The two most basic rules of counting are the sum and product rule. The **product rule** provides a way to count sequences.

Theorem: The Product Rule

Let $A_1, A_2, \ldots, A_n$ be finite sets. Then,

$$|A_1 \times A_2 \times \cdots \times A_n| = |A_1| \cdot |A_2| \cdots |A_n|$$

Counting Strings

If Σ is a set of characters (called an **alphabet**) then Σ^n is the set of all string of length n whose characters come from the set Σ. The product rule can be applied directly to determine the number of strings of a given length over a finite alphabet:

$$|\Sigma^n| = |\underbrace{\Sigma \times \Sigma \times \cdots \Sigma}_{n \text{ times}}| = \underbrace{|\Sigma| \cdot |\Sigma| \cdots |\Sigma|}_{n \text{ times}} = |\Sigma|^n$$

Theorem: The Sum Rule

Consider n sets, $A_1, A_2, \ldots, A_n$. If the sets are pairwise disjoint, meaning that $A_i \cap A_j = \emptyset$ for $i \neq i$, then

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

10.2 The Bijection Rules

One way to approach difficult counting problems is to show that the cardinality of the set to be counted is equal to the cardinality of a set that is easy to count. The **bijection rule** says that if there is a bijection from one set to another then the two sets have the same cardinality.

A function f from a set S to a set T is called a bijection if and only if f has a well-defined *inverse*, f^{-1}.

The Bijection Rule

Let S and T be two finite sets. If there is a bijection from S to T, then $|S| = |T|$

The k-to-1 Rule

Let X and Y be finite sets. The function $f : X \to Y$ is a **k-to-1 correspondence** if for every $y \in Y$, there are exactly k difference $x \in x$ such that $f(x) = y$.

Suppose there is a k-to-1 correspondence from a finite set A to a finite set B. Then

$$|B| = \frac{|A|}{k}.$$

10.3 The generalized product rule

The **generalized product rule** says that in selecting an item from a set, if the number of choices at each step does not depend on previous choices made, then the number of items in the set is a product of the number of choices in each step.

Generalized Product Rule

Consider a set S of sequences of k items. Suppose there are:

- n_1 choices for the first item.
- For every possible choice for the first item, there are n_2 choice for the second item.
- For every possible choice for the first and second items, there are n_3 choices for the third item.
- $\vdots$
- For every possible choice for the first $k-1$ items, there are n_k choices for the k-th item.

Then $|S| = n_1 \cdot n_2 \cdots n_k$.

10.4 Counting permutations

An **r-permutation** is a sequence of r items with **no repetitions**, all taken from the same set. In a sequence, order matters, so (a, b, c) is different from (b, a, c).

The number of r-permutations from a set with n elements

Let r and n be positive integers with $r \leq n$. The number of r-permutations from a set with n elements is denoted by $P(n, r)$:

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

10.5 Counting subsets

A subset of size r is called an **r-subset**. An r-subset is sometimes referred to as an **r-combination**. In a subset, order does not matter, so $\{a, b, c\}$ is the same as $\{b, a, c\}$. The counting rules for sequences and subsets are commonly referred to as "*permutations* and *combinations*". The term "combination" is the context of counting is another word for "subset".

Counting Subsets: 'n choose r' notation

The number of ways to select an r-subset from a set of size n is:

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

$\binom{n}{r}$ is read as "n choose r". The notation $C(n, r)$ is sometimes used for $\binom{n}{r}$.

We can calculated an expression for $\binom{n}{n-r}$ by replacing r with $n-r$ in the expression for $\binom{n}{r}$.

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

This is an **identity** for r-subsets.

10.6 Subset and permutation examples

Two different cat selection problems: Subset vs. Permutations

Consider two closely related counting problems:

1. A family goes to the animal shelter to adopt 3 cats. The shelter has 20 different cats from which to select. How many ways are there for the family to their selection?

2. Three different families go to the animal shelter to adopt a cat. Each family will select one cat. How many ways are there for the families to make their selection? (Note that which family gets which cat matters).

In the first problem, the number is ways to make the selection is $\binom{20}{3}$ because the order in which the cats are selected is not important. The outcome is a 3-subset.

In the second problem, the specific cat selected by each family is important. Additionally, no cat can belong to two families. Thus, the answer is $P(20,3) = 20 \cdot 19 \cdot 18$. The outcome is a 3-permutation.

10.7 Counting by complement

Counting by complement is a technique for counting the number of elements in a set S that have a property by counting the total number of elements in S and subtracting the number of elements in S that do not have the property.

$$|P| = |S| - |\overline{P}|$$

Suppose we want to count the number of people in a room with red hair. We know that there are 20 people in the room and exactly 12 of them do not have red hair. Then we can deduce that the number of people in the room with red hair is 20 - 12 = 8.

10.8 Permutations with repetitions

A **permutation with repetition** is an ordering of a set of items in which some of the items may be identical to each other. To illustrate with a smaller example, there are $3! = 6$ permutations of the letters CAT, because the letters in CAT are all different. However, there are only 3 different ways to scramble the letters in DAD: ADD, DAD, DDA.

Formula for Counting Permutations with Repetition

The number of distinct sequences with $n_1 1's, n_2's, \ldots, n_k k's$, where $n = n_1 + n_2 + \cdots + n_k$ is

$$\frac{n!}{n_1! n_2! \cdots n_k!}$$

The formula for permutations with repetition is derived from repeated use of the formula for counting r-subsets:

$$\begin{aligned}
&\binom{n}{n_1}\binom{n-n_1}{n_2}\binom{n-n_1-n_2}{n_3}\cdots\binom{n-n_1-n_2-\cdots-n_{k-1}}{n_k} \\
=&\frac{n!}{n_1!(n-n_1)!}\cdot\frac{(n-n_1)!}{n_2!(n-n_1-n_2)!}\cdot\frac{(n-n_1-n_2)!}{n_3!(n-n_1-n_2-n_3)!}\cdots\frac{(n-n_1-n_2-\cdots-n_{k-1})!}{n_k!0!} \\
=&\frac{n!}{n_1! n_2! \cdots n_k!}
\end{aligned}$$

10.9 Counting multisets

A set is a collection of distinct items. A **multiset** is a collection that can have multiple instances of the same kind of item. When $\{1,2,2,3\}$ is viewed as a set, the repetitions don't matter and $\{1,2,2,3\} = \{1,2,3\}$. However, when $\{1,2,2,3\}$ is viewed as a multiset, then the fact there are two occurrences of 2 is important, and $\{1,2,2,3\} \neq \{1,2,3\}$. Two multisets are equal if they have the same number of each type of element. For multisets, the order of elements still does not matter.

Rules for encoding a selection of n objects from m varieties

Selections	Code words
n = number of items to select	n = number of 0's in code word
m = number of varieties	$m-1$ = number of 1's in code word
Number selected from the first variety	Number of 0's before the first 1
Number selected from the i-th variety, for $1 < i < m$	Number of 0's between the i-1st and i-th 1
Number selected from the last variety	Number of 0's after the last 1

If the mapping of selections to code words is a bijection, then by the bijection rule, the number of distinct code words is equal to the number of distinct selections. If the number of objects to select is n, and the number of varieties of object is m, each code word has n 0's and $m-1$ 1's, for a total of $n+m-1$ bits. The binary string of length $n+m-1$ with exactly $m-1$ 1's is

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

Theorem: Counting Multisets

The number of ways to select n objects from a set of m varieties is

$$\binom{n+m-1}{m-1},$$

if there is no limitation on the number of each variety available and objects of the same variety are indistinguishable.

A set of identical items are called **indistinguishable** because it is impossible to distinguish one of the item from another. A set of different or distinct items are called **distinguishable** because it is possible to distinguish one of the items from the others.

10.10 Assignment problems: Balls in bins

	No restrictions (any positive m and n)	**Max 1 ball per bin** (m must be at least n)	**Same # of balls per bin** (m must evenly divide n)
Indistinguishable	$\binom{n+m-1}{m-1}$	$\binom{m}{n}$	1
Distinguishable	m^n	$P(m,n)$	$\frac{n!}{((n/m)!)^m}$

10.11 Inclusion-exclusion principle

The **principle of inclusion-exclusion** is a technique for determining the cardinality of the union sets that uses the cardinality of each individual set as well as the cardinality of their intersections.

The inclusion-exclusion principle with two sets

Let A and B be two finite sets, then $|A \cup B| = |A| + |B| - |A \cap B|$

The inclusion-exclusion principle with three sets

Let $A, B,$ and C be three finite sets, then

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

The inclusion-exclusion principle with an arbitrary number of sets

Let $A_1, A_2, \ldots, A_n$ be a set of n finite sets.

$$\begin{aligned}|A_1 \cup A_2 \cup \cdots \cup A_n| = &\sum_{j=1}^{n} |A_j| \\ &- \sum_{1 \le j \le k \le n} |A_j \cap A_k| \\ &+ \sum_{1 \le j \le j \le l \le n} |A_j \cap A_k \cap A_l| \\ &\vdots \\ &+ (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n|\end{aligned}$$

The inclusion-exclusion principle and the sum rule

A collection of sets is **mutually disjoint** if the intersection of every pair of sets in the collection is empty. If we apply the principle of inclusion-exclusion to determine the union of a collection of mutually disjoint sets, then all the terms with the intersections are zero. Thus, for a collection of mutually disjoint sets, the cardinality of the union of the sets is just equal to the sum of the cardinality of each of the individual sets:

$$|A_1 \cup A_2 \cup \cdots \cup A_n| = |A_1| + |A_2| + \cdots + |A_n|.$$

The equation above is a restatement of the sum rule which only applies when the sets are mutually disjoint.

Determining the Cardinality of a Union by Complement

Counting by complement can be used to express the size of the union as:

$$|U| - |\overline{P_1 \cup P_2 \cup \cdots \cup P_n}| = |P_1 \cup P_3 \cup \cdots \cup P_n|$$

11 Advanced Counting

11.1 Generating permutations

There are situations in which it is necessary to generate, not just count, all permutations of a set or subsets of a given size.

Lexicographic Order

A well-defined order imposed on the n-tuples is useful to systematically generate all the elements in a set of n-tuples. Generating the n-tuples in the set from smallest to largest ensures that each n-tuple is generated exactly once.

Lexicographic order is a way or ordering n-tuples in which two n-tuples are compared according to the first entry where they differ. An example of such ordering is the word in a dictionary.

Generating Permutations

A **permutation** of the set $\{1, 2, \ldots, n\}$ is an ordered n-tuple in which each number in $\{1, 2, \ldots, n\}$ appears exactly once. For example, $(2, 5, 1, 4, 3)$ is a permutation of the set $\{1, 2, 3, 4, 5\}$.

Generating r-subsets of a set

Unlike sequences or n-tuples, the order in which the elements of a set or subset are written does not matter. Sets can be ordered lexicographically by first sorting the elements in increasing order and then comparing the two sets as if they were ordered sequences. For example, $\{2, 3, 11\} < \{2, 5, 6\}$, because the first element is the same in both sets but in the second element $3 < 5$.

11.2 Binomial coefficients and combinatorial identities

An **identity** is a theorem stating that two mathematical expressions are equal.

Theorem: A Simple Combinatorial Identity

For any non-negative integers n and k such that $k \leq n$:

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

A proof that makes use of counting principles is called a **combinatorial proof**. Combinatorial proofs usually involve defining a set S and counting the number of elements in S to get a mathematical expression for the number of items in the set. Every combinatorial proof of an identity uses a bijection implicitly as part of the argument.

Theorem: The Binomial Theorem

For any non-negative integer n and any real numbers a and b:

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

The coefficients $\binom{n}{k}$ are called binomial coefficients.

For the case $n = 5$, the Binomial Theorem says that

$$\begin{aligned}(a+b)^5 &= \binom{5}{0}a^5 + \binom{5}{1}a^4 b + \binom{5}{2}a^3 b^2 + \binom{5}{3}a^2 b^3 + \binom{5}{4}ab^4 + \binom{5}{5}b^5 \\ &= a^5 + 5a^4 b + 10a^3 b^2 + 10a^2 b^3 + 5ab^4 + b^5\end{aligned}$$

The Binomial Theorem can also be used to obtain combinatorial identities. For example, by plugging in $a = b = 1$, the Binomial Theorem yields the identity below.

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

Similarly, by letting $a = -1$ and $b = 1$, and requiring that n be positive, the left hand side of the Binomial Theorem becomes 0. The right hand side of the Binomial Theorem becomes:

$$0 = \sum_{k=0}^{n} (-1)^k \binom{n}{k}$$

In the expanded form,

$$\binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \binom{n}{3} + \cdots + (-1)^n \binom{n}{n} = 0$$

Pascal's Triangle

The 17^{th} century French mathematician, Blaise Pascal, developed a triangular chart that contains all the number of the form $\binom{n}{k}$. The chart, now known as Pascal's Triangle, can be used to derive the value of a particular $\binom{n}{k}$. The n^{th} row of Pascal's Triangle contains the $n+1$ binomial coefficients of the form $\binom{n}{k}$ as shown in the figure below.

$$\begin{array}{lccccccccc}
n=0 & & & & \binom{0}{0} & & & & & \\
n=1 & & & \binom{1}{0} & & \binom{1}{1} & & & & \\
n=2 & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & & & \\
n=3 & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} & & \\
n=4 & \binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4}
\end{array}$$

Theorem: Pascal's Identity

For any positive n and k such that $k < n$:

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

11.3 The pigeonhole principle

The pigeonhole principle is a mathematical tool used to establish that repetitions are guaranteed to occur in certain sets and sequences. The **pigeonhole principle** says that if $n+1$ pigeons are placed in n boxes, then there must be at least one box with more than one pigeon. The diagram below shows 10 pigeons, represented as P, in 9 boxes.

$$\begin{array}{ccc} P & P & P \\ P & P & P, P \\ P & P & P \end{array}$$

Theorem: The Pigeonhole Principle

If a function f has a domain of size at least $n+1$ and a target of size at most n, where n is a positive integer, then there are two elements in the domain that map to the same element in the target (i.e., the function is not one-to-one)

Theorem: The Generalized Pigeonhole Principle

Consider a function whose domain has at least n elements and whose target has k elements, for n and k positive integers. Then there is an element y in the target such that f maps at least $\lceil n/k \rceil$ elements in the domain to y.

Theorem: Converse of the Generalized Pigeonhole Principle

Suppose that a function f maps a set of n elements to a target with k elements, where n and k are positive integers. In order to guarantee that there is an element y in the target to which f maps to at least b elements from the domain, then n must be at least $k(b-1)+1$.

11.4 Generating functions

Generating functions are a powerful tool that can be used to solve a variety of problems related to counting and recurrence relations. A **generating function** is a way of representing a sequence of number as a n algebraic function in which each term in the sequence is a coefficient of an x^j term in the function. The advantage of representing sequences algebraically is that there are many techniques from algebra that can be used to manipulate functions which then leads to insight about the sequences they represent.

The sequence $f_0, f_1, f_2, f_3, \ldots$ is represented by the generating function

$$F(x) = f_0 + f_1 x + f_2 x^2 + f_3 x^3 + \cdots$$

The numbers in the sequence are the coefficients of the terms in $F(x)$. For the sequence $1, 1, 1, 1, \ldots$ is represented by the generating function

$$H(x) = 1 + x + x^2 + x^3 + \cdots$$

When $|x| < 1$ for $H(x)$ the sum is finite and has a closed form of

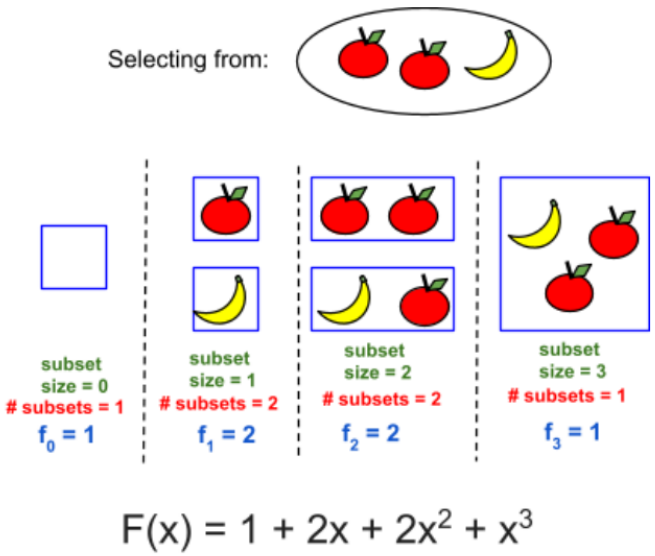
$$H(x) = \sum_{j=0}^{\infty} x^j = \frac{1}{1-x}$$

Additionally, for the partial sum up to x^k,

$$\sum_{j=0}^{k} x^j = \frac{1-x^k}{1-x}$$

Using Generating Functions for Counting

One of the primary uses for generating functions is to help solve counting problems. In general, f_j is the number of way to select a subset of j objects. Consider the situation in which there is a set of two apples and one banana. The apples are indistinguishable, so selecting one apple is the same as selecting the other.



If the set from which the subsets are selected is an infinite set, then the resulting sequence is also infinite.

Summary of Generating Functions

Description	Long Form	Short Form
Infinite supply of one kind of item	$1 + x + x^2 + x^3 + \cdots$	$\frac{1}{1-x}$
Selecting from a set of n identical items	$1 + x + x^2 + x^3 + \cdots + x^n$	$\frac{1-x^{n-1}}{1-x}$.
Infinite supple of identical items grouped in batches of k	$1 + x^k + x^{2k} + x^{3k} + \cdots$	$\frac{1}{1-x^k}$
Infinite supple of 2 different kinds of items	$1 + 2x + 3x^2 + 4x^3 + \cdots$	$\frac{1}{(1-x)^2}$

Products of Generating Functions

Generating functions become very useful when different sets of objects are combined together into larger sets.

For example, if we are selecting subsets of apples from a set of 3 apples, the generating function is $A(x) = 1 + x + x^2 + x^3$, and if we are selecting bananas from a set of 2 bananas, the generating function is $B(x) = 1 + x + x^2$. Now suppose we pool the three apples and two bananas into a set of five pieces of fruit and ask 'how many ways can one select a set of fruit from the collection of five pieces of fruit?'

The power of generating functions is illustrated in the fact that if we take the product of $A(x)$, the generating function for apples, and $B(x)$, the generating function for bananas, to get $A(x)B(x)$, then the resulting product is the generating function for the pooled set of five pieces.

$$A(x)B(x) = (1 + x + x^2 + x^3)(1 + x + x^2) = 1 + 2x + 3x^2 + 3x^3 + 2x^4 + x^5$$

Note that the rule of multiplying generating function only works when the two sets of objects being combined are *distinct*. The rule would not work for combining two sets of indistinguishable apples.

Now suppose that we add oranges to the set of selections. There are three oranges, but they come wrapped in a single pack, so one can select zero or three oranges, but not one or two. The generating function for oranges is $O(x) = 1 + x^3$. Again, we can take the product of the generating functions to solve, 'how many ways are there to select subsets of fruit of a particular size?'

$$A(x)B(x)O(x) = 1 + 2x + 3x^2 + 4x^3 + 4x^4 + 4x^5 + 3x^6 + 2x^7 + x^8$$

The coefficient of x^5 in the generating function for the whole set of fruit is 4, so there are 4 ways to select 5 pieces of fruit.

12 Discrete Probability

12.1 Probability of an event

One of the primary applications of counting is to calculate probabilities of random events.

An **experiment** is a procedure that results in one out of a number of possible **outcomes**. The set of all possible outcomes is called the **sample space** of the experiment. A subset of the sample space is called an **event**.

Discrete vs. Continuous Probability

Discrete probability is concerned with experiments in which the sample space is finite or a countably infinite set. A set is **countably infinite** if there is a one-to-one correspondence between the elements of the set and the integers. An infinite set that is not countably infinite is said to be **uncountably infinite**.

Probability Distributions

A **probability distribution** over the outcomes of an experiment with a countable sample space S if a function $p : S \to [0, 1]$ with the property that

$$\sum_{s \in S} p(s) = 1.$$

The probability of outcome s is $p(s)$. If $E \subseteq S$ is an event, then the **probability of event** E is

$$p(E) = \sum_{s \in E} p(s).$$

The Uniform Distribution

The probability distribution in which every outcome has the same probability is the **uniform distribution**. The uniform distribution reduces questions about probabilities to questions about counting because for every event E,

$$p(E) = \frac{|E|}{|S|}.$$

12.2 Unions and complements of events

Calculating Probabilities for Unions of Events

Two events are **mutually exclusive** if the two events are disjoint, meaning that the intersection of the two events is empty. If follows from the definition of the probability of an event that if E_1 and E_2 are mutually exclusive, then:

$$p(E_1 \cup E_2) = p(E_1) + p(E_2).$$

However, if two events are not mutually exclusive, the probability of the union of the events can be determined by a version of the Inclusion-Exclusion principle:

$$p(E_1 \cup E_2) = p(E_1) + p(E_2) - p(E_1 \cap E_2)$$

The statement holds for non-uniform as well as uniform distributions.

The Complement of an Event

The **complement** of an event E is $S - E$ and is denoted by $\overline{E}$. Since $\overline{E}$ and E are disjoint events, $p(\overline{E}) + p(E) = 1$. It follows then that

$$p(\overline{E}) = 1 - p(E).$$

12.3 Conditional probability and independence

If the event F happens, the new probability of E is the **conditional probability** of E given F, denoted by $p(E|F)$. The conditional probability of E given F is

$$p(E|F) = \frac{p(E \cup F)}{p(F)}.$$

If the distribution is uniform, then $p(E) = |E|/|S|$ and the conditional probability becomes:

$$p(E|F) = \frac{p(E \cup F)}{p(F)} = \frac{|E \cup F|/|S|}{|F|/|S|} = \frac{|E \cup F|}{|F|}.$$

The Complement of an Event and Conditional Probability

If E and F are both events in the same sample space S, then the probability of E and the probability of $\overline{E}$ still sum to 1, even when conditioned on the event F.

$$p(E|F) + p(\overline{E}|F) = 1$$

Independent Events

Let E and F be two events in the same sample space. The following three conditions are equivalent.

1. $p(E|F) = \frac{p(E \cap F)}{p(F)} = p(E)$
2. $p(E \cap F) = p(E) \cdot p(F)$
3. $p(F|E) = \frac{p(E \cap F)}{p(E)} = p(F)$

If one of the three conditions hold, then events E and F are independent.

Calculating the Probabilities of Two Independent Events

If X and Y are events in the same sample space, and X and Y are independent, then

$$p(X \cap Y) = p(X) \cdot p(Y).$$

Mutual Independence

Events $A_1, \ldots, A_n$ in sample space S are **mutually independent** if the probability of the intersection of any subset of the events is equal to the product of the probabilities of the events in the subset. In particular, if $A_1, \ldots, A_n$ are mutually independent, then

$$p(A_1 \cap A_2 \cap \cdots \cap A_n) = p(A_1) \cdot p(A_2) \cdots p(A_n).$$

12.4 Bayes' Theorem

Suppose that F and X are events from a common sample space and $p(F) \neq 0$ and $p(X) \neq 0$. Then

$$p(F|X) = \frac{p(X|F)p(F)}{p(X|F)p(F) + p(X|\overline{F})p(\overline{F})}.$$

This is known as Bayes' Theorem. In other words, Bayes' theorem tells us how to update our initial beliefs about a hypothesis (represented by $p(F)$) based on new evidence (represented by $p(X|F)$), taking into account the prior probability of the hypothesis (represented by $p(F)$) and the overall probability of observing the evidence (represented by $p(X)$).

12.5 Random variables

A **radnom variable** X is a function from the sample space S of an experiment to the real numbers. $X(S)$ denoted the range of the function X.

Random Variables and Probabilities

If X is a random variable defined on the sample space S of an experiment and $r \in \mathbb{R}$, then $X = r$ is an event. The event $X = r$ consists of all outcomes s in the sample space such that $X(s) = r$. $p(X = r)$ is the sum of the $p(s)$ for all s such that $X(s) = r$.

Distribution over a Random Variable

The **distribution** of a random variable is the set of all pairs $(r, p(X = r))$ such that $r \in X(S)$.

12.6 Expectation of random variables

The **expected value** of a random variable X is denoted $E[X]$ and is defined as

$$E[X] = \sum_{s \in S} X(s)p(s),$$

where $p(s)$ is the probability of outcome s. Alternatively, if X is a random variable defined over an experiment with a sample space S,

$$E[X] = \sum_{r \in X(S)} r \cdot p(X = r),$$

where $X(S)$ is the range of the function X.

12.7 Linearity of expectations

If X and Y are two random variables defined on the same sample space S, and $c \in \mathbb{R}$,

$$E[X + Y] = E[X] + E[Y], \quad \text{and}$$
$$E[cX] = xE[X].$$

Linearity of expectations can be shown by induction to apply to more than two variables. If $X_1, \ldots, X_n$ are n variables defined on the same sample space, then

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

12.8 Bernoulli trials and the binomial distribution

A **Bernoulli trial** is an experiment with two outcomes: **success** and **failure**. In a sequence of independent Bernoulli trials, called a **Bernoulli process**, the outcomes of the repeated experiments are assumed to be mutually independent and have the same probability of success and failure. Usually the probability of success is denoted by the variable p, and the probability of failure, $(1 - p)$, denoted by the variable q.

Bernoulli Trial and Probabilities

The probability of exactly k successes in a sequence of n independent Bernoulli trials, with probability of success p and probability of failure $q = 1 - p$ is

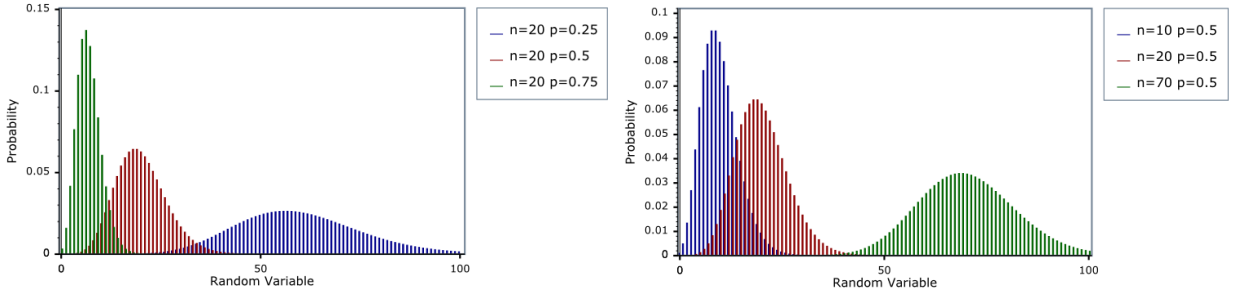
$$\binom{n}{k} p^k q^{n-k}.$$

The distribution over the random variable defined by the number of the successes in a sequence of independent Bernoulli trials is called the **binomial distribution**. The probability that the number of successes is k in a sequence of length n with probability of success p is denoted by $b(k; n, p)$. By the theorem above,

$$b(k; n, p) = \binom{n}{k} p^k.$$

The range of the random variable denoting the number of successes in a sequence of n Bernoulli trials is 0 through n. Since the values of $b(k;)$ are a probability distribution over the possible values for k, the probabilities should sum to 1 as k ranges from 0 through n:

$$\sum_{k=0}^{n} b(k; n, p) = \sum_{k=0}^{n} \binom{n}{k} p^k q^{n-k} = (p + q)^n = 1$$



The Expected Number of Successes

The expected number of successes for n Bernoulli trials with probability of success p is

$$E[K] = np,$$

where K is the random variable denoting the number of successes in n Bernoulli trials, and $E[K]$ is the expected number of successes.

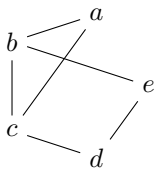
13 Graphs

13.1 Introduction to Graphs

Graphs are fundamental objects in discrete mathematics that model relationships between pairs of objects. Graphs arise in a wide array of disciplines but play an especially important role in computer science.

In an **undirected graph**, the edges are unordered pairs of vertices, which is useful for modeling relationships that are symmetric. A graph consists of a pair of sets (V, E), where V is a set of vertices and E is a set of edges. A graph is **finite** if the vertex set is finite. A single element of V is called a **vertex** and is usually represented pictorially by label, or a dot with a label. Each edge in E is a set of two vertices from V and is drawn as a line connecting the two vertices.

An undirected graph

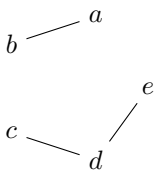


In the above graph, two edges cross each other, but there is no vertex at the crossing. The crossing is just a byproduct of how the graph is drawn on a two-dimensional surface. The graph above can be described by listing the vertex set and the edge set:

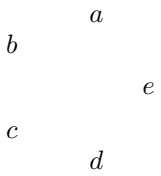
$$V = \{a, b, c, d, e\}$$
$$E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{b, e\}, \{c, d\}, \{d, e\}\}$$

A graph may appear to be disconnected into more than one piece but is still considered to be one graph. Here are two graphs, each with 5 vertices.

Graph A

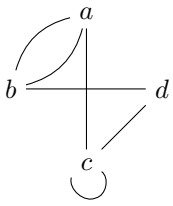


Graph B



Parallel edges are multiple edges between the same pair of vertices. In defining graphs with parallel edges, it *might* be important to have additional label besides the two endpoints to specify an edge in order to distinguish between different parallel edges. A graph can also have a **self-loop** which is an edge between a vertex, and itself.

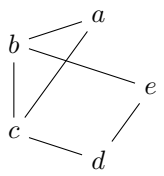
A graph with parallel edges and a self-loop



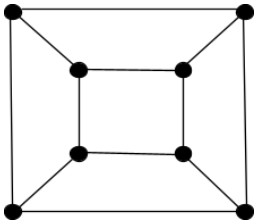
If a graph does not have parallel edges or self-loops, it is said to be a **simple graph**.

Graph terminology

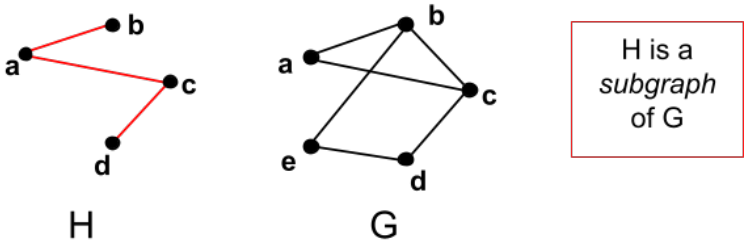
Undirected graph example



- If there is an edge between two vertices, they are said to be **adjacent**. In the graph above, d and e are adjacent, but d and b are not adjacent.
- Vertices b and e are the **endpoints** of edge $\{b, e\}$. The edge $\{b, e\}$ is **incident** to vertices $btande$.
- A vertex c is a **neighbor** of vertex b if and only if $\{b, c\}$ is an edge. In the graph above, the neighbors of b are the vertices $a, c,$ and e.
- In a simple graph, the **degree** of a vertex is the number of neighbors it has. In the graph above, the degree of b is 3 and the degree of vertex a is 2. The degree of vertex b is denoted by $\deg(b)$.
- The **total degree** of a graph is the sum of the degrees of all of the vertices. The total degree of the above graph is $2 + 3 + 3 + 2 + 2 = 12$.
- In a **regular graph**, all the vertices have the same degree. In a d**-regular graph**, all the vertices have degree d. The graph above is not regular because $\deg(a) \neq \deg(b)$. However, the graph below is 3-regular.



- A graph $H = (V_H, E_H)$ is a **subgraph** of a graph $G = (V_G, E_G)$ if $V_H \subseteq V_G$ and $E_H \subseteq E_G$. Note that any graph G is a subgraph of itself.



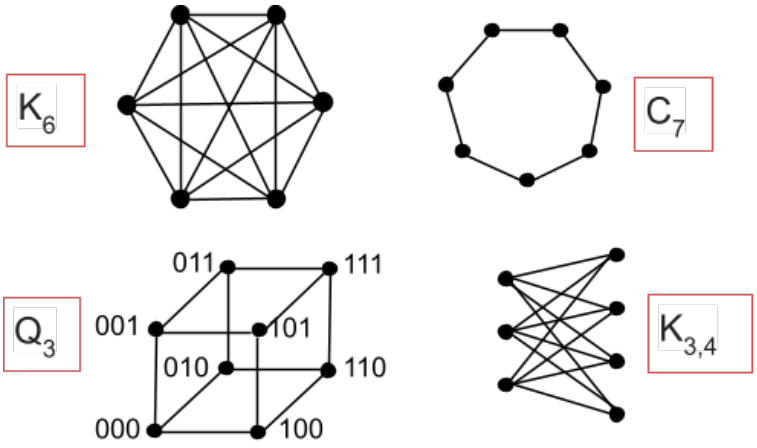
Theorem: Number of Edges and Total Degree

Let $G = (V, E)$ be an undirected graph, simple or not. Then, twice the number of edges is equal to the total degree:

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

Common Graphs

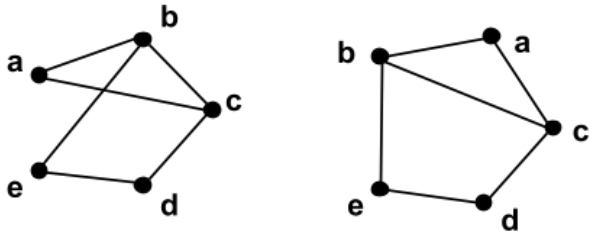
Some graphs with special structure are given their own name and notation because they come up so frequently in graph theory.



For the definition of all of these graphs, $n \in \mathbb{Z}^+$.

- K_n is called the **complete graph** on n vertices. K_n has an edge between every pair of vertices. K_n is sometimes called a **clique** of size n or an **n-clique**.
- C_n is called a cycle on n vertices. The edges connect the vertices in a ring. Note that C_n is well defined only for $n \geq 3$.
- The n-dimensional hypercube, denoted Q_n, has 2^n vertices. Each vertex is label with an n-bit string. Two vertices are connected by an edge if their corresponding labels differ only by one bit.
- $K_{n,m}$ has $n + m$ vertices, and $2nm$ edges. The vertices are divided into two sets: one with m vertices and one set with n vertices. There are no edges between vertices in the same set, but there is an edge between every vertex in one set and every vertex in the other set.

13.2 Graph representations



These two graphs look different, but that is only because they are drawn differently. The two graphs are actually the same graph because they have the same vertex and edge sets as shown below:

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

The way a graph is drawn is not part of the graph itself.

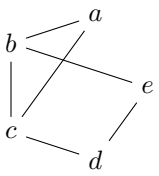
Adjacency List Representation for Graphs

In the **adjacency list representation** of a graph, each vertex has a list of all its neighbors. Note that since the graph is undirected if vertex a is in b's list of neighbors, then b must also be in a's list of neighbors.

If a graph is represented using adjacency lists, the time required to list the neighbors of a vertex v is proportional to $\deg(v)$, the number of vertices to be listed. In order to determine if $\{a, b\}$ is an edge, it is necessary to scan the list of a's neighbors or the list of b's neighbors. In the worst case, the time required is proportional to the larger of $\deg(a)$ or $\deg(b)$.

Matrix Representation for Graphs

The **matrix representation** for a graph with n vertices is an $n \times n$ matrix whose entries are all either 0 or 1, indicating whether or not each edge is present. The matrix representation of an undirected graph is symmetric.



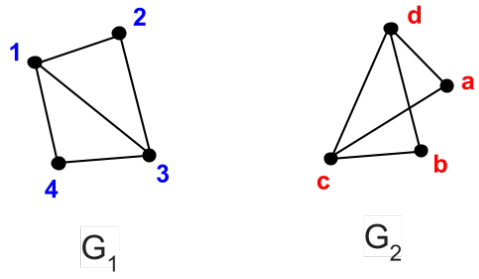
$$\Rightarrow \quad \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

13.3 Graph isomorphism

Two graphs are said to be **isomorphic** if there is a correspondence between the vertex sets of each graph such that there is an edge between two vertices of one graph if and only if there is an edge between the corresponding vertices of the second graph. Essentially, the graphs are not identical but the vertices can be relabeled so that they are identical.

Definition of Isomorphic Graphs

Let $G = (V, E)$ and $G' = (V', E')$. G and G' are isomorphic if there is a bijection $f : V \to V'$ such that for every pair of vertices $x, y \in V$, $\{x, y\} \in E$ if and only if $\{f(x), f(y)\} \in E'$. The function f is called an **isomorphism** from G to G'.



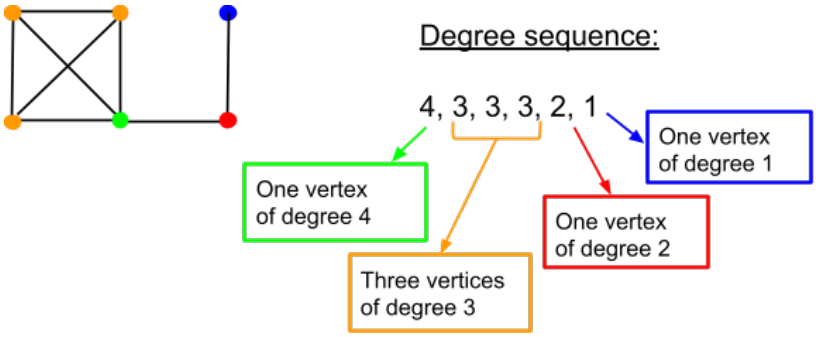
The following function is an isomorphism from the vertices of G_1 to G_2:

$$f(1) = d \qquad f(2) = a \qquad f(3) = c \qquad f(4) = b$$

Theorem: Vertex Degree Preserved under Isomorphism

Consider two graphs, G and G'. Let f be an isomorphism from G to G'. For each vertex v in G, the degree of vertex v in G is equal to the degree of vertex $f(v)$ in G'.

The **degree sequence** of a graph is a list of the degree of all of the vertices in non-increasing order.



Theorem: Degree Sequence Preserved under Isomorphism

The degree sequence of a graph is preserved under isomorphism.

13.4 Walks, trails, circuits, paths, and cycles

A **walk** from v_0 to v_l in an undirected graph G is a sequence of alternating vertices and edges that starts and ends with a vertex:

$$\langle v_0, \{v_0, v_1\}, v_1, \{v_1, v_2\}, \ldots, v_{\ell-1}, \{v_{\ell-1}, v_\ell\}, v_\ell \rangle$$

Since the edges in a walk are completely determined by the vertices, a walk can also be denoted by the sequence of vertices:

$$\langle v_0, v_1, \ldots, v_\ell \rangle.$$

However, keep in mind the sequence of vertices is a walk only if $\{v_{j-1}, v_j\} \in E$ for each $j = 1, 2, \ldots, \ell$. The **length of a walk** is ℓ, the number of edges in the walk. An **open walk** is a walk in which the first and last vertices are not the same. A **closed walk** is a walk in which the first and last vertices are the same.

- A **trail** is a walk in which no *edge* occurs more than once.
- A **path** is a walk in which no *vertex* occurs more than once.
- A **circuit** is a closed *trail*.
- A **cycle** is a *circuit* is length at least 1 in which no vertex occurs more than once, except the first and last vertices which are the same.

Here are some examples of closed walks:

- $\langle A, C, D, A \rangle$ is a *trail*, a *circuit*, and a *cycle*.
- $\langle A, C, B, A, D, E, A \rangle$ is a *trail* and a *circuit*.
- $\langle A, B, A \rangle$ is **not** a trail, circuit, or a cycle.

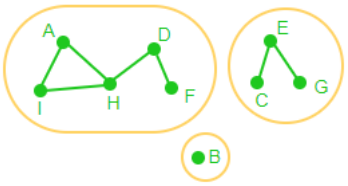
Since paths and cycles do not have any repeated edges, if a graph is simple, any cycle *must* have length at least 3. The sequence $\langle v \rangle$ is not a cycle because a cycle, by definition, must have length at least 1. The sequence $\langle v, v \rangle$ is only a walk if there is a self-loop at vertex v.

13.5 Graph connectivity

Two vertices, v and w, are **connected** if there is a path from v to w (and thus also a path from w to v). A vertex is always considered to be connected to itself. The property of being connected can be extended to sets of vertices and the entire graph:

- A set of vertices in a graph is said to be connected if every pair of vertices in the set is connected.
- A graph is said to be connected if every pair of vertices in the graph is connected, and is **disconnected** otherwise.

A **connected component** consists of a *maximal* set of vertices that are connected as well as all the edges between two vertices in the set. A vertex that is not connected with any other vertex is called an **isolated vertex** and is therefore a connected component with only one vertex.



The connected components are
{A, D, I, F, H}
{C, E, G}
{B}

Vertices in the same connected component are connected to each other

k-Connectivity

In some networks, it is important to be able to guarantee connectivity, even if one or more vertices or edges are removed from a graph. The definition of connectivity can be extended to encompass resilience to vertex or edge failures.

Definition of a K-vertex-connected Graph

An undirected graph G is **k-vertex-connected** if the graph contains at least $k+1$ vertices and remains connected after any $k-1$ vertices are removed from the graph. The **vertex connectivity** of a graph is the largest k such that the graph is k-vertex-connected. The vertex connectivity of a graph G is denoted $\kappa(G)$.

The vertex connectivity of a graph is the minimum number of vertices whose removal disconnects the graph into more than one connected component.

When the graph is a complete graph, there is no set of vertices whose removal disconnects the graph. For the special case of K_n, the vertex connectivity $\kappa(K_n)$ is just defined to be $n-1$.

Definition of a K-edge-connected Graph

An undirected graph G is **k-edge-connected** if removing any $k-1$ or fewer edges results in a connected graph. The **edge connectivity** of a graph is the largest k such that the graph is k-edge-connected. The edge connectivity of a graph G is denoted $\lambda(G)$.

The edge connectivity of a graph is the minimum number of edges whose removal disconnects the graph into more than one connected component.

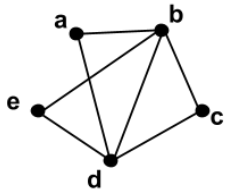
Theorem: Upper bound for Vertex and Edge Connectivity

Let G be an undirected graph. Denote the minimum degree of any vertex in G by $\delta(G)$. Then,

$$\kappa(G) \leq \delta(G) \quad \text{and}$$
$$\lambda(G) \leq \delta(G).$$

13.6 Euler circuits and trails

An **Euler circuit** is a circuit that contains every edge and every vertex. Note that a circuit, by definition, has no repeated edges, so an Euler circuit contains each edge exactly once.



Euler circuit: $\langle a, b, c, d, e, b, d, a \rangle$

Theorem: Characterization of Graphs that have an Euler Circuit

An undirected graph G has an Euler circuit if and only if G is connected and every vertex in G has even degree.

G has an Euler circuit $\leftrightarrow G$ is connected and every vertex in G has even degree.

Procedure to find a circuit in a Graph

```
Find a vertex w, that is not an isolated vertex.
Select any edge {w,x} incident to w.
(Since w is not isolated, there is always at least one such edge.)
Current trail T := <w,x>
last := x
While there is an edge {last, y} that has not been used in T:
  Add y to the end of T
  last := y
```

Procedure to find an Euler circuit in a Graph

Use the procedure above to find any circuit in G. Call the circuit C. The algorithm continues to iterate the following steps until all the edges in G are included in C:

1. Remove all edges in C from G. Remove any isolated vertices from G. Call the resulting graph G'.
2. Find a vertex w that is in G' and C.
3. Find a circuit in G' that begins and ends with w. Call the circuit C'.
4. Combine circuit C and C'. Suppose C starts and ends at vertex v. Create a new circuit that starts at v and follows the edges in C until w is reached. The new circuit then follows the edges in C' back to w and then follows the rest of the edges in C back to v. The new circuit is renamed C for the next iteration.

Euler trail

An **Euler trail** is an open trail that includes each edge. Note that a trail, by definition, has no repeated edges, so an Euler trail contains each edge exactly once. In an open trail, the first and last vertices are not equal.

Theorem: Characterizations of graphs that have an Euler trail

An undirected graph G has an Euler trail if and only if G is connected and has exactly two vertices with odd degree. The Euler trail begins and ends with the vertices of odd degree.

G has an Euler trail $\leftrightarrow G$ is connected and has exactly two vertices with odd degree.

13.7 Hamiltonian cycles and paths

A **Hamiltonian cycle** in an undirected graph is a cycle that includes every vertex in the graph. Note that a cycle, by definition, has no repeated vertices or edges, except for the vertex which is at the beginning and end of the cycle. Therefore, every vertex in the graph appears exactly once in a Hamiltonian cycle, except for the vertex which is at the beginning and end of the cycle. A **Hamiltonian path** in an undirected graph is a path that includes every vertex in the graph. Note that a path, by definition, has no repeated vertices or edges, so every vertex appears exactly once in a Hamiltonian path.

Note that a Hamiltonian cycle can be transformed into a Hamiltonian path by deleting the last vertex. Therefore if a graph has a Hamiltonian cycle, then the graph also has a Hamiltonian path.

Unlike Euler circuits and trails, there are no known conditions describing exactly which graphs have a Hamiltonian cycle or path. However, there are some cases in which a graph that does have a Hamiltonian cycle or path has a certain property.

- Any graph that has a vertex with degree 1 does not have a Hamiltonian cycle.
- For $n \geq 3$, K_n has a Hamiltonian cycle.

13.8 Planar graphs

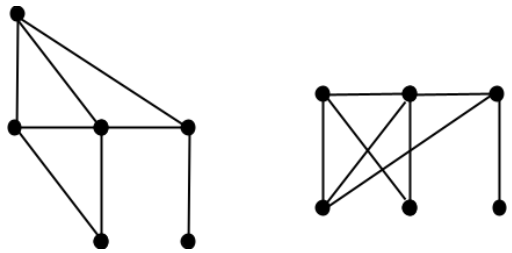
Placing graphs on two-dimensional surfaces to avoid crossings is a classic problem in graph theory. The problem also arises in the field of graph drawing in which the goal is to draw complex graphs in a way that helps people visualize structure and patterns.

An **embedding** for $G = (V, E)$ is an assignment of the vertices to points in the plane and an assignment of each edge to a continuous curve. The curve for each edge must start and edge at the two points corresponding to the endpoints of the edge. Essentially, an *embedding is a way of drawing a graph* on a plane, because mathematically, a graph is just a set of vertices and a set of edges.

An embedding is said to be a **planar embedding** if none of the edges cross. There is a crossing between two edges in an embedding if their curves intersect at a point that is not a common endpoint. An embedding of a graph can be planar or not planar, depending on whether it has a crossing. Planarity can also be defined as a property of the graph itself.

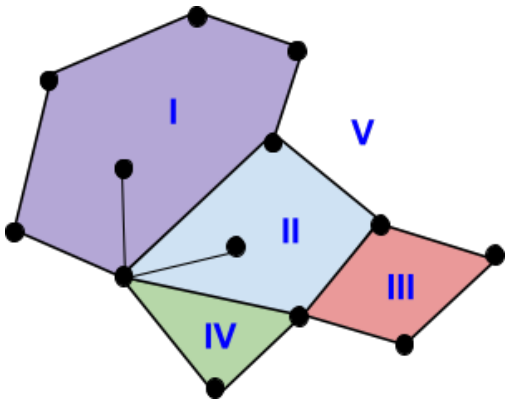
Definition of Planar Graphs

A graph G is a **planar graph** if the graph has a planar embedding.



- The embedding on the left *is planar.*
- The embedding on the right *is **not** planar.*

The **complement of an embedding** is the set of all points in the plane that are not part of a curve corresponding to an edge. A planar embedding carves up the plane into continuous regions. A **region** is a set of points in the complement of an embedding that forms a maximal continuous set, meaning that it is (continuous): possible to travel anywhere from any point in the region and (maximal): if any point were added to the region, it would no longer be continuous. In a planar embedding, there is always an infinite region called the **exterior region**, which is region V in the following graph.

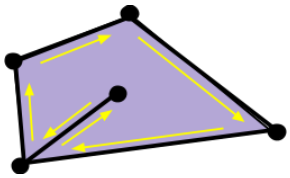


Theorem: Euler's Identity for Regions in an Embedding

Consider a planar embedding of a connected graph G. Let v be the number of vertices in G, e the number of edges, and r the number of regions in the embedding. Then,

$$v - e + r = 2$$

Degree of a Region



Degree of the purple region is 6:
6 edges traversed around the region

Consider a planar embedding of a graph G. Think of a tiny bug walking all the way around the region along the edges of the graph. The degree of a region is the number of times the bug traverses an edge until it gets back to its starting location. Note that if an edge sticks out into a region, the edge can be traversed twice by the bug and therefore contributes 2 towards the degree of the region.

Theorem: Number of Edges in a Planar Graph

Let G be a connected planar graph. Let v be the number of vertices in G and e the number of edges. If $v \geq 3$, then

$$e \leq 3v - 6$$

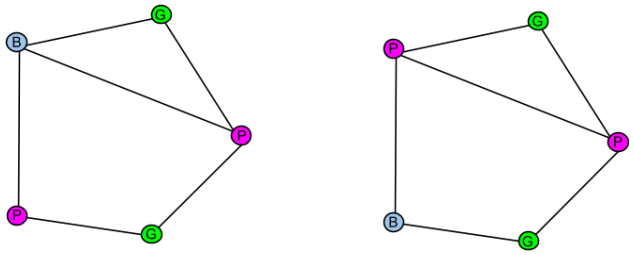
13.9 Graph coloring

Graph coloring is a classic problem in graph theory because it is useful in modeling resource constraints like scheduling.

Definition of Graph Coloring

Let $G = (V, E)$ be an undirected graph and C a finite set of colors. A **valid coloring** of G is a function $f : V \to C$ such that for every edge $\{x, y\} \in E, f(x) \neq f(y)$. If the size of the range of function f is k, then f is called a **k-coloring** of G.

The coloring on the left is a valid coloring because no two adjacent vertices are assigned the same color. The coloring on the left is a 3-coloring because 3 colors are used. The coloring on the right is not a valid coloring because there is an edge whose endpoints are both colored pink.



Definition of the Chromatic Number of a Graph

The **chromatic number** of a graph G, denoted as $X(G)$ is the smallest k such that there is a valid k-coloring of G. It is minimum number of colors that a graph requires to have a valid graph coloring.

In general, if K_r is a subgraph of G, then there is a subset of r vertices in G such that every pair of vertices in the subset is connected by an edge. In any valid coloring of G, each of the r vertices in the subset must be assigned a distinct color which implies that $X(G) \geq r$. The clique number of a graph G, denoted $\omega(G)$, is the largest r such that K_r is a subgraph of G.

Theorem: Relationship between the Clique Number of Chromatic Number

If G is an undirected graph, then

$$\omega(G) \leq X(g)$$

Greedy coloring

In general, it is difficult to determine the chromatic number of a graph G. However, there is an easy and natural method to color the vertices of a graph called the *greedy coloring algorithm*. The **greedy coloring algorithm** often leads to a color that uses a small number of colors, but there is no guarantee that it uses the *smallest* number of colors possible for a given graph.

The Greedy Coloring Algorithm

1. Number the set of possible colors. Assume that there is a very large supply of different colors, even though they might not all be used.
2. Order to vertices in any arbitrary order.
3. Consider each vertex v in order. Assign v a color that is different from the color of v's neighbors that have already been assigned that color. When selecting a color for v, use the lowest number color possible.

The term "greedy" is used to describe a general paradigm for solving many different kinds of problems. Greedy algorithms typically solve a problem one piece at a time.

The greedy algorithm provides a way to upper bound the chromatic number of a graph.

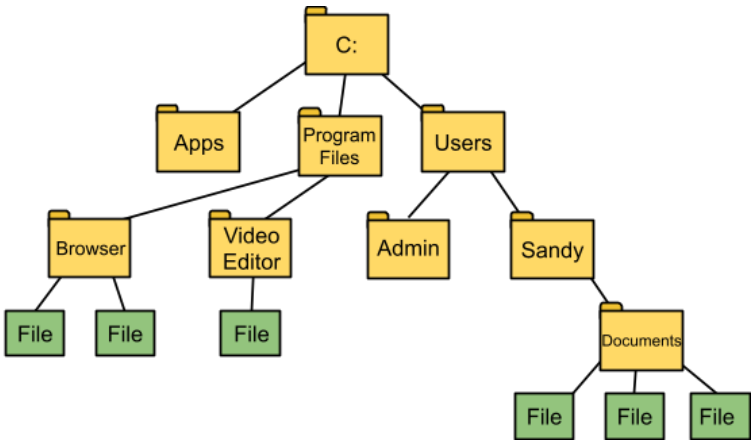
Theorem: Upper bound for $X(G)$

Let G be an undirected graph. Let $\Delta(G)$ be the maximum degree of any vertex in G. Then,

$$X(G) \leq \Delta(G) + 1$$

14 Trees

14.1 Introduction to trees

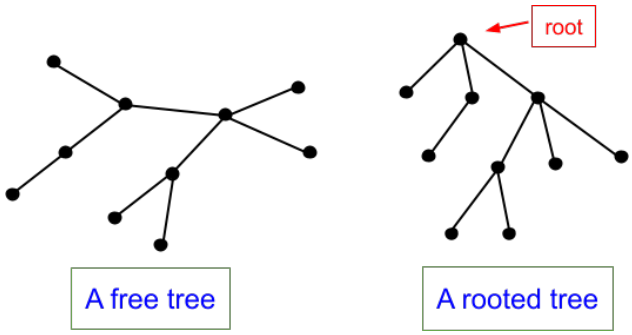


The file system can be seen as a graph in which each folder or file is a vertex. There is an edge between two folders if one folder is a subfolder of the other. There is an edge between a file and a folder if the file resides in that folder. In most computer operating systems, a file or folder can only reside in one location, which means that the underlying graph corresponds to a tree.

Definition of a Tree

A **tree** is an undirected graph that is connected and has no cycles.

Free Trees and Rooted Trees



The tree on the left is called a **free tree** because there is no particular organization of the vertices and edges. The tree on the right is called a **rooted tree**. The vertex at the top of the drawing is designated as the **root** of the tree. The remaining vertices are organized according to their distance from the root. The distance between two vertices in an undirected graph is the number of edges in the shortest path between the two vertices. The **level** of a vertex is its distance from the root. The **height** of a tree is the highest level of any vertex. The tree on the right has height 3.

Theorem: Unique Paths in Trees

Let T be a tree and let u and v be two vertices in T. There is *exactly one path* between u and v.

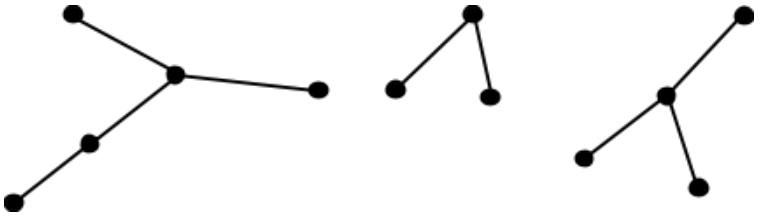
Terminology related to Rooted Trees

- Every vertex in a rooted tree T has a unique **parent**, except for the root which does not have a parent. The parent of vertex v is the first vertex encountered along the path from v to the root.

- Every vertex along the path from v to the root, except for the vertex v itself, is an **ancestor** of vertex v.
- If v is the parent of vertex u, then u is a **child** of vertex v.
- If u is an ancestor of v, then v is a **descendant** of u.
- A **leaf** is a vertex which has no children.
- Two vertices are **siblings** if they have the same parent.
- A **subtree** rooted as vertex v is the tree consisting of v and all of v's descendants.

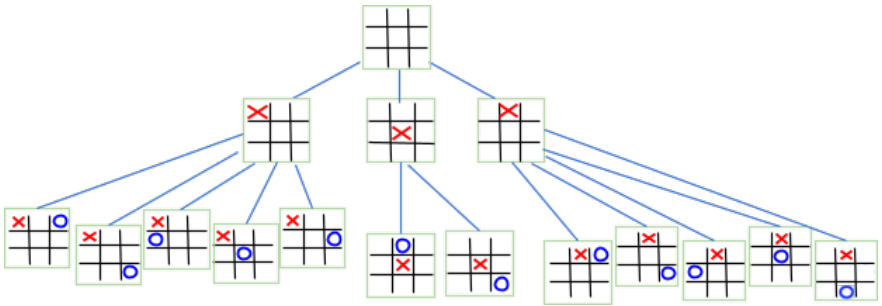
Forests

A **forest** is a graph that has no cycles but is not necessarily connected. The picture below shows a forest with 3 connected components.



14.2 Tree application examples

Game Trees



The root of the tree is the initial configuration of the game. In the case of tic-tac-toe, the initial configuration is an empty grid. The odd levels of the game tree represent choices of play for the X player and the even levels represent choices of play for the O player. The children of a configuration c are all the configurations that can be reached from c by a single move of the correct player. A configuration is a leaf vertex in the tree if the game is over.

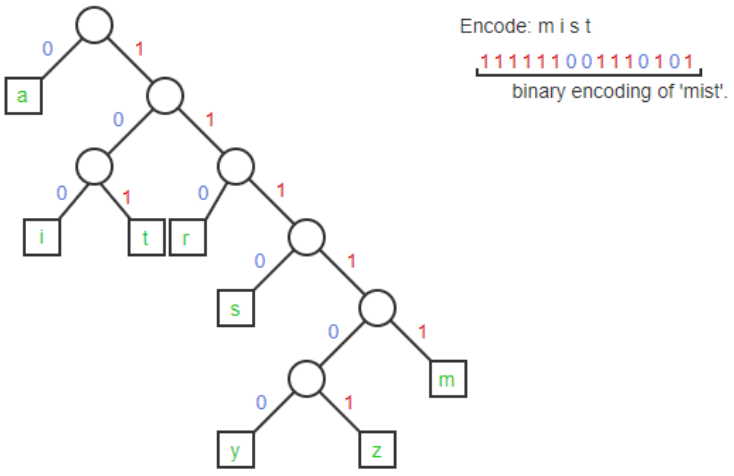
Theoretically, a game tree can be systematically analyzed to determine optimal playing strategies for each player. However, in practice for most games, the corresponding game tree is much too large to build and analyze in its entirety. Programs like Deep Blue build partial game trees starting from the current configuration in the game and estimate the best next move based on the results of the partial tree.

Tic-tac-toe and chess are examples of deterministic games whose outcome is completely determined by the choices made by each player. A game that involves rolling a pair of dice or shuffling a deck of cards introduces randomization into the game. Probability theory as well as game trees are required to analyze games with an element of chance.

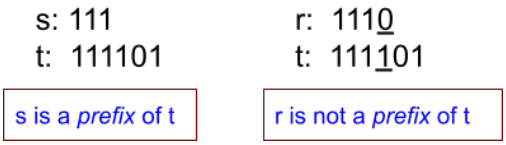
Prefix Codes

Space-efficient encodings can be achieved by **variable length codes**, in which the number of bits for each character can vary. Characters such as "a" and "e" that occur frequently are represented with fewer bits, while characters such as "z" or "q" are represented using more bits.

Trees are a convenient way to represent variable length codes for translating between text and binary.



The code illustrated in the animation is an example of a prefix code. A string s is a **prefix** of another string t if all the characters in string s appear at the beginning of string t.



A **prefix code** has the property that the code for one character *can not* be a prefix of the code for another character. The fact that the codes are organized as a tree in which characters are only stored at the leaves ensures the prefix property.

14.3 Properties of trees

Theorem: Unique Paths in Trees

There is a unique path between every pair of vertices in a tree.

Theorem: Number of Leaves in a Tree

Any free tree with at least two vertices has at least two leaves.

Theorem: Number of Edges in a Tree

Let T be a tree with v vertices and e edges. Then

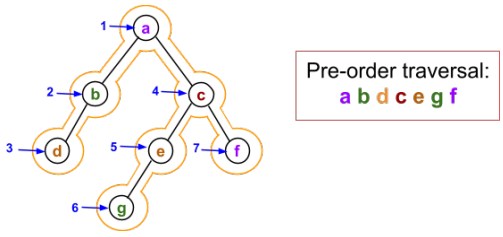
$$e = v - 1.$$

14.4 Tree traversals

A common procedure performed on a tree, called a **tree traversal**, is to process the information stored in the vertices by systematically visiting each vertex. In a **pre-order traversal**, a vertex is visited before its descendants. In a **post-order traversal**, a vertex is visited after its descendants.

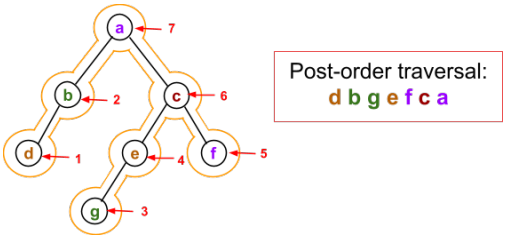
Pseudocode and Example for Pre-order Traversal

```
Pre-order(v)

process(v) // process happens first
For every child w of v:
  Pre-order(w)
End-for
```



Pseudocode and Example for Post-order Traversal

```
Post-order(v)

For every child w of v:
  Post-order(w)
End-for
process(v) // process happens last
```



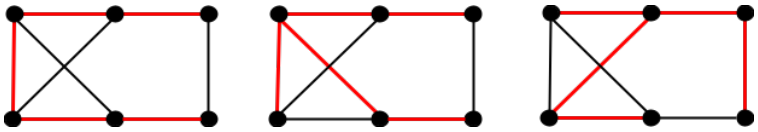
14.5 Spanning trees and graph traversals

Suppose an application needs to broadcast information to every computer (vertex) in a network. The information will be disseminated along the communication links so that every vertex in the network receives the information. Moreover, the goal is to minimize overall congestion, so it would be wasteful to send information along alternate paths to the same location. The information should be broadcast over a spanning tree of the network. Here is the formal definition of a spanning tree:

Definition of a Spanning Tree of a Connected Graph

A **spanning tree** of a connected graph G is a subgraph of G which contains all the vertices in G and is a tree.

The picture below shows several possible spanning trees of the same graph. The edges of the spanning tree are shown in red. The vertex set of the spanning tree is always the entire set of vertices in the graph.



Graph Traversals

There are two common methods for finding spanning trees in a graph: Breadth-First Search and Depth-First Search. Both methods start at a single vertex and incrementally build a connected tree by adding edges and vertices. The resulting tree is rooted at the start vertex. Graph traversal is a process that systematically explores all the vertices of a graph.

Depth-First Search

As the name suggests, Depth-First Search (DFS) favors going deep into the graph and tends to produce trees with longer paths from the start vertex.

```
Depth-first-search

Input: An undirected, connected graph G. A start vertex v[1]
Output: T, a depth-first search tree.

Add v[1] to T.
visit(v[1])

visit(v)

For every neighbor w of v:
  If w is not already in T
    Add w and {w, v} to T.
    visit(w);
  End-if
End-for
```

Note that there is some ambiguity with regard to the order in which the neighbors of a vertex are considered.

Breath-first Search

Breadth-First Search (BFS) explores the graph by distance from the initial vertex, starting with its neighbors and expanding the tree to neighbors of neighbors, etc. Breadth-first-search visits vertices in the graph according to their proximity to the start vertex. The algorithm maintains a list of vertices to be visited soon. Vertices are added to the back of the list and the next vertex to visit is taken from the front of the list.

```
Breadth-first-search

Input: An undirected, connected graph G. A start vertex v[1]
Output: T, a breadth-first search tree.

Add v[1] to T.
Add v[1] to the back of the list.

While the list is not empty:
  Remove vertex v from the front of the list.
  For each neighbor w of v that is not already in T:
    Add w and {w, v} to T.
    Insert w at the back of the list.
```

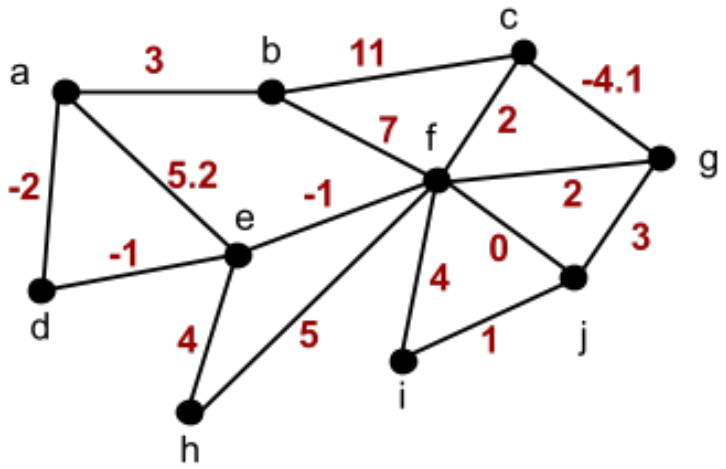
```
  End–for
End–while
```

14.6 Minimum spanning trees

When edges have varying costs, a natural goal is to minimize the total cost of the spanning tree.

Definition of a Weighted Graph

A **weighted graph** is a graph $G = (V, E)$, along with a function $w : E \to \mathbb{R}$. The function w assigns a real number to each edge.



When the goal is to span the vertices of the graph while minimizing the total weight of the edges used, a **minimum spanning tree** can be used.

Definition of a Minimum Spanning Tree

A **minimum spanning tree** (MST) of a weighted graph is a spanning tree T of G whose weight is no larger than any other spanning tree of G.

Prim's Algorithm for Minimum Spanning Trees

This is a classic algorithm for finding minimum spanning trees, developed by mathematician Robert Prim in 1957.

```
Input: An undirected, connected, weighted graph G.
Output: T, a minimum spanning tree for G.

T := {}.
Pick any vertex in G and add it to T.

For j = 1 to n–1
  Let C be the set of edges with one endpoint inside T
      and one endpoint outside T.
  Let e be a minimum weight edge in C.
  Add e to T.
  Add the endpoint of e not already in T to T.
End–for
```

Theorem: Result of Prim's Algorithm

Prim's algorithm finds a minimum spanning tree of the input weighted graph.