CSE 191

Introduction to Discrete Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Course Summary and Review

Final Exam Logistics

- The final is Tuesday 5/16/23 in **Davis 101** from 3:30PM to 6:30PM
  - If you have any official conflicts (2 exams at same time, or 3 same day) please let me know ASAP
  - Seating will be randomized
  - All bags/electronics will be placed in the front of the room
- **What's provided:** Equivalence laws and inference rules
- **What to bring:**
  - UB ID card
  - Pen/pencil
  - One 8.5x11 cheat sheet (front and back)

Topic List

Emphasis on bolded topics (they were not covered by the midterm)

1. Propositional Logic
2. Logical Equivalence
3. Predicates and Quantifiers
4. Logical Reasoning/Proofs
5. **Sets**
6. **Functions and Relations**
7. **Sequences**
8. **Counting**
9. **Graphs**
10. **Finite Automata**

Propositional Logic [Chapter 1.1, 1.2]

Propositions

A **proposition** is a declarative statement

- Must be either **TRUE (T)** or **FALSE (F)**
  - Cannot be both…
  - Referred to as the **truth value** of the proposition
- An opinion of a specific person is a proposition
  - Their opinion would determine the true value
- The bits 0/1 are used for F/T
  - Digital logic uses 0/1, LOW/HIGH, or OFF/ON
  - Computers use bits and logic gates for **all** computation

Propositional Variables

Propositional variables are variables that represent propositions

- Commonly used letters are **p**, **q**, **r**, **s**, …
    - Alternatively, the first letter of what we are trying to represent
- May be associated a specific proposition or left as a placeholder for an arbitrary proposition

Compound propositions are formed by using propositional variables and logical operators

- A compound proposition is itself a proposition

Logical Operators

Logical Operators allow combining propositions into new ones

- Going forward: combine propositions to form new ones
- Going backward: decompose proposition into atomics

Example Compound Proposition

If I am at work**, then** I am wearing sneakers

Logical operator (if …, then …)

Negation Operator

Let $\boldsymbol{p}$ be a proposition.

The *negation of* $\boldsymbol{p}$, denoted by $\neg \boldsymbol{p}$ (or sometimes $\overline{\boldsymbol{p}}$) is the statement:

"It is not the case that $\boldsymbol{p}$"

- $\neg \boldsymbol{p}$ is a new proposition, read as "not $\boldsymbol{p}$"
- $\neg$ is referred to as the negation operator. It is a ***unary*** operator
  - Unary operators only operate on one proposition
- The truth value of $\neg \boldsymbol{p}$ is the opposite of the truth value of $\boldsymbol{p}$

Binary Logical Operators: Conjunction

Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be propositions.

The *conjunction of* $\boldsymbol{p}$ and $\boldsymbol{q}$, denoted by $\boldsymbol{p} \land \boldsymbol{q}$ is the statement:

"$\boldsymbol{p}$ and $\boldsymbol{q}$"

and is only TRUE when $\boldsymbol{p}$ and $\boldsymbol{q}$ are both TRUE, and is FALSE otherwise

Binary Logical Operators: Disjunction

Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be propositions.

The *disjunction of* $\boldsymbol{p}$ and $\boldsymbol{q}$, denoted by $\boldsymbol{p} \lor \boldsymbol{q}$ is the statement:

"$\boldsymbol{p}$ or $\boldsymbol{q}$"

and is TRUE when $\boldsymbol{p}$ is TRUE, $\boldsymbol{q}$ is TRUE, or both are TRUE

$\boldsymbol{p} \lor \boldsymbol{q}$ is FALSE only when both $\boldsymbol{p}$ and $\boldsymbol{q}$ are FALSE

Binary Logical Operators: Exclusive Or

Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be propositions.

The *exclusive or of* $\boldsymbol{p}$ and $\boldsymbol{q}$, denoted by $\boldsymbol{p} \oplus \boldsymbol{q}$ (read XOR) is the statement:

"$\boldsymbol{p}$ or $\boldsymbol{q}$, but not both"

and is TRUE when exactly one of $\boldsymbol{p}$ and $\boldsymbol{q}$ is TRUE, and FALSE otherwise

Binary Logical Operators: Implication

Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be propositions.

The *implication of* $\boldsymbol{p}$ on $\boldsymbol{q}$, denoted by $\boldsymbol{p} \to \boldsymbol{q}$ is the statement:

"$\boldsymbol{p}$ implies $\boldsymbol{q}$" or "if $\boldsymbol{p}$, then $\boldsymbol{q}$"

and is FALSE when $\boldsymbol{p}$ is TRUE, $\boldsymbol{q}$ is FALSE, and TRUE otherwise

$\boldsymbol{p}$ is called the <u>hypothesis</u> or <u>antecedent</u> or <u>precedent</u>

$\boldsymbol{q}$ is called the <u>conclusion</u> or <u>consequence</u>

Binary Logical Operators: Bidirectional Implication

Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be propositions.

The *bidirectional implication of* $\boldsymbol{p}$ on $\boldsymbol{q}$, denoted by $\boldsymbol{p} \Leftrightarrow \boldsymbol{q}$ is the statement:

"$\boldsymbol{p}$ if and only if $\boldsymbol{q}$"

and is only TRUE when $\boldsymbol{p}$ and $\boldsymbol{q}$ have same truth value, FALSE otherwise

Truth Table

A **Truth Table** lists all possible combinations of truth values of the operands, as well as the resulting truth value in the rightmost column

Truth Tables: Negation Operator

- The negation operator has a single operand
  - This operand can either be TRUE or FALSE
- The truth value of $\neg p$ is the opposite of the truth value of p

p	$\neg p$
F	T
T	F

Truth table for negation

Truth Tables: Binary Logical Operators

p	q	$p \land q$
F	F	F
F	T	F
T	F	F
T	T	T

Conjunction/AND

Truth Tables: Binary Logical Operators

p	q	$p \vee q$
F	F	F
F	T	T
T	F	T
T	T	T

Disjunction/OR

p	q	$p \oplus q$
F	F	F
F	T	T
T	F	T
T	T	F

Exclusive Or/XOR

Truth Tables: Binary Logical Operators

p	q	$p \rightarrow q$
F	F	T
F	T	T
T	F	F
T	T	T

Implication/If …, then …

p	q	$p \Leftrightarrow q$
F	F	T
F	T	F
T	F	F
T	T	T

Bidirectional Implication/IFF

Logical Equivalence
[Chapter 1.3]

Tautologies and Contradictions

A **tautology** is a compound proposition that is ALWAYS TRUE, no matter what the truth values of the propositional variables that occur in it.

A **contradiction** is a compound proposition that is ALWAYS FALSE.

A **contingency** is a compound proposition that is neither a contradiction or a tautology. There at least one assignment of truth values to the atomics that can result in TRUE, and at least one that can result in FALSE.

Logical Equivalence

Two propositions, p and q are **logically equivalent** if $p \Leftrightarrow q$ is a tautology

- In other words, p and q are logically equivalent if their truth values in their truth table are all the same
- Two compound propositions are logically equivalent if their truth values agree for all combinations of the truth values of their atomics
- We write equivalence as $p \equiv q$
  - $\equiv$ is NOT a logical operator
  - $p \equiv q$ is NOT a compound proposition

Logical Equivalence Rules

Equivalence	Name
$p \land T \equiv p$ $\quad$ $p \lor F \equiv p$	Identity laws
$p \lor T \equiv T$ $\quad$ $p \land F \equiv F$	Domination laws
$p \lor p \equiv p$ $\quad$ $p \land p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \lor q \equiv q \lor p$ $\quad$ $p \land q \equiv q \land p$	Commutative laws
$(p \lor q) \lor r \equiv p \lor (q \lor r)$ $\quad$ $(p \land q) \land r \equiv p \land (q \land r)$	Associative laws
$p \lor (q \land r) \equiv (p \lor q) \land (p \lor r)$ $\quad$ $p \land (q \lor r) \equiv (p \land q) \lor (p \land r)$	Distributive laws
$\neg(p \lor q) \equiv \neg p \land \neg q$ $\quad$ $\neg(p \land q) \equiv \neg p \lor \neg q$	De Morgan's laws
$p \lor (p \land q) \equiv p$ $\quad$ $p \land (p \lor q) \equiv p$	Absorption laws
$p \lor \neg p \equiv T$ $\quad$ $p \land \neg p \equiv F$	Negation laws

Logical Equivalence Rules

Equivalences with Implication
$p \rightarrow q \equiv \neg p \vee q$
$p \rightarrow q \equiv \neg q \rightarrow \neg p$
$p \vee q \equiv \neg p \rightarrow q$
$p \wedge q \equiv \neg(p \rightarrow \neg q)$
$\neg(p \rightarrow q) \equiv p \wedge \neg q$
$(p \rightarrow q) \wedge (p \rightarrow r) \equiv p \rightarrow (q \wedge r)$
$(p \rightarrow q) \wedge (q \rightarrow r) \equiv (p \vee q) \rightarrow r$
$(p \rightarrow q) \vee (p \rightarrow r) \equiv p \rightarrow (q \vee r)$
$(p \rightarrow q) \vee (q \rightarrow r) \equiv (p \wedge q) \rightarrow r$

Equivalences with Bidirectional Implication
$p \Leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$
$p \Leftrightarrow q \equiv q \Leftrightarrow p$
$p \Leftrightarrow q \equiv \neg p \Leftrightarrow \neg q$
$p \Leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
$\neg(p \Leftrightarrow q) \equiv p \Leftrightarrow \neg q$

Proving Logical Equivalence

- By using equivalence laws, we can prove two propositions are logically equivalent without having to construct large truth tables
- The logical equivalences shown in the tables can be used to construct additional logical equivalences

Proving Logical Equivalence

In General:

- Each line should be equivalent to the previous
- Each line should list the law that led to it
  - Exactly one law applied per line
- Start with LHS and go until you reach the RHS

Note: Logical equivalence proofs are very exact. Later proofs will be less restrictive.

Propositional Satisfiability

A compound proposition is **satisfiable** if there is an assignment of truth values to its variables that makes it true.

A compound proposition is **unsatisfiable** when no such assignment exists

- A compound proposition is unsatisfiable iff its negation is a tautology
- An assignment of truth values that make a compound proposition true is called a solution to that satisfiability problem

Predicates and Quantifiers [Chapter 1.4, 1.5]

Predicates

A **predicate** is a *function* that takes some *variable(s) as arguments*; it returns either TRUE or FALSE, but never both, depending on the combination of the combination of values passed as arguments.

Example: $P(x)$: x is an even number.

$\boldsymbol{P}$ is the function, $\boldsymbol{x}$ is the variable

$\boldsymbol{P(x)}$ is the value of the predicate $\boldsymbol{P}$ at $\boldsymbol{x}$

Domain of Discourse

Given a predicate, ***P*(*x*)**, the **domain of discourse** (often just called the **domain**) is the set of all possible values for the variable ***x***.

- Predicates with multiple variables may have:
  - Multiple domains of discourse (one for each variable)
  - A single domain of discourse for all variables

Quantifiers

Quantification expresses the extent to which a predicate is true over a range of elements. For example, in English: all, some, none, many, few, …

Universal Quantification

Suppose ***P*(*x*)** is a predicate on some domain, ***D***.

The **<u>universal quantification</u>** of ***P*(*x*)** is the *proposition*:

"***P*(*x*)** is true for all ***x*** in the domain of discourse ***D***."

Written as: **∀x, *P*(*x*)**

Read as: "For all ***x*, *P*(*x*)**" or "For every ***x*, *P*(*x*)**"

∀x, *P*(*x*) is TRUE if ***P*(*x*)** is TRUE for *<u>every</u>* ***x*** in ***D***.

∀x, *P*(*x*) is FALSE if ***P*(*x*)** is FALSE for *<u>some</u>* ***x*** in ***D***.

Existential Quantification

Suppose ***P***(***x***) is a predicate on some domain, ***D***.

The **existential quantification** of ***P***(***x***) is the *proposition*:

"***P***(***x***) is true for some ***x*** in the domain of discourse ***D***."

Written as: $\exists x, P(x)$

Read as: "There exists an ***x*** such that, ***P(x)***" or "For some ***x***, ***P(x)***"

$\exists x, P(x)$ is TRUE if ***P***(***x***) is TRUE for *some* ***x*** in ***D***.

$\exists x, P(x)$ is FALSE if ***P***(***x***) is FALSE for *every* ***x*** in ***D***.

Binding Variables

The occurrence of a variable, ***x***, is said to be **bound** when a quantifier is used on that variable.

The occurrence of a variable, ***x***, is said to be **free** when it is not bound by a quantifier or set to a particular variable.

The part of a logical expression to which a quantifier is applied is called the **scope** of this quantifier.

Note: A variable is free if it is outside of the scope of all quantifiers in the formula that specify this variable.

Quantifier Negation Rule

Quantifier Negation

In general we have, for any predicate ***P*(*x*)**:

$$\neg \forall x, P(x) \equiv \exists x, \neg P(x) \text{ and } \neg \exists x, P(x) \equiv \forall x, \neg P(x)$$

De Morgan's Law for Quantifiers			
Negation	**Equivalent statement**	**Negation is TRUE when…**	**Negation is FALSE when…**
$\neg \exists x, P(x)$	$\forall x, \neg P(x)$	For every x, P(x) is FALSE	There is an x where P(x) is TRUE
$\neg \forall x, P(x)$	$\exists x, \neg P(x)$	There is an x where P(x) is FALSE	P(x) is true for every x

Nested Quantifiers

A logical expression with more than one quantifier that bind different variables in the same predicate is said to have **nested quantifiers**.

In order to evaluate them we must consider their *ordering* and *scope*

Nested Quantifiers: Scope

The portion of the formula a quantifier covers is called the **scope**

- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

$$\forall i \ \exists j, (P(i,j) \rightarrow \ \forall k, Q(k,j))$$

Nested Quantifiers: Scope

The portion of the formula a quantifier covers is called the **scope**

- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

$$\boxed{\forall i\ \exists j, (P(i,j) \rightarrow\ \forall k, Q(k,j)}\,)$$

The scope of $\forall i$ is the entire formula

Nested Quantifiers: Scope

The portion of the formula a quantifier covers is called the **scope**

- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

$$\forall i\ \boxed{\exists j, (P(i,j) \rightarrow \forall k, Q(k,j)}\,)$$

The scope of $\exists j$ is the entire formula (other than $\forall i$)

Nested Quantifiers: Scope

The portion of the formula a quantifier covers is called the **scope**

- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

$$\forall i\ \exists j, (P(i,j) \rightarrow \boxed{\forall k, Q(k,j)})$$

The scope of $\forall k$ is the limited to $Q(k,j)$

Nested Quantifiers: Scope

The portion of the formula a quantifier covers is called the **scope**

- The scope of the quantifier is the predicate immediately following
- Precedence is just below parenthesis
- Any variable not covered by any quantifier is a free variable

Consider the following formula:

$$\boxed{\forall i\ \boxed{\exists j, (P(i,j) \rightarrow \boxed{\forall k, Q(k,j)})}}$$

Logical Reasoning/Proofs [Chapter 1.6-1.8]

Logical Reasoning: What is it?

Suppose the following are TRUE statements:

1. You will buy your friend lunch if they drive you to work
2. They drove you to work

What can you conclude?

You will buy your friend lunch.

Note: This differs from logical equivalence

- Statements derived are not always equivalent
- Can derive new knowledge from multiple facts

Logical Reasoning: Arguments

Arguments are:

- a list of propositions, called **hypotheses** (also called **premises**)
- a final proposition, called the **conclusion**

$$\begin{array}{rl} \text{Hypotheses} \rightarrow & \boxed{\begin{array}{l} p_1 \\ p_2 \\ \dots \\ p_n \end{array}} \\ \hline \text{Conclusion} \rightarrow & \therefore c \end{array} \quad \begin{array}{c} (p_1 \land p_2 \land \dots \land p_n) \\ \downarrow \\ c \end{array}$$

An argument is **valid** if $(p_1 \land p_2 \land \dots \land p_n) \rightarrow c$ is a tautology

- Otherwise, it is **invalid**
- **Fallacies** are incorrect reasonings which lead to invalid arguments

Logical Reasoning: Proof Definition

A **logical proof** of an argument is a sequence of steps, each of which consists of a proposition and a justification

Each line should contain one of the following:

- a hypothesis (assumption)
- a proposition that is equivalent to a previous statement
- a proposition that is derived by applying an argument to previous statements

Jusificiations should state one of the following:

- hypothesis
- the equivalence law used (and the line it was applied to)
- the argument used (and the line(s) it was applied to)

The last line should be the conclusion

Logical Reasoning: Invalid Argument

Remember: An argument is valid if $(p_1 \land p_2 \land \ldots \land p_n) \rightarrow c$ is a tautology

Therefore to show it is invalid, we need a **counterexample**. A counterexample is a situation where the hypotheses are all TRUE, and the conclusion is FALSE.

Example consider the converse as an argument.

$$\begin{array}{c} p \rightarrow q \\ \hline \therefore q \rightarrow p \end{array}$$

Suppose p: FALSE and q: TRUE.
Then $p \rightarrow q$ is TRUE, but $q \rightarrow p$ is FALSE.
Therefore the argument is invalid.

Logical Reasoning: Rules of Inference

Rule of Inference	Name
$\begin{array}{c} p \\ p \rightarrow q \\ \hline \therefore q \end{array}$	Modus Ponens
$\begin{array}{c} \neg q \\ p \rightarrow q \\ \hline \therefore \neg p \end{array}$	Modus Tollens
$\begin{array}{c} p \rightarrow q \\ q \rightarrow r \\ \hline \therefore p \rightarrow r \end{array}$	Hypothetical Syllogism

Logical Reasoning: Rules of Inference

Rule of Inference	Name
$\begin{array}{c} p \lor q \\ \neg p \\ \hline \therefore q \end{array}$	Disjunctive Syllogism
$\begin{array}{c} p \\ \hline \therefore p \lor q \end{array}$	Addition
$\begin{array}{c} p \land q \\ \hline \therefore p \end{array}$	Simplification

Logical Reasoning: Rules of Inference

Rule of Inference	Name
$\begin{array}{c} p \\ q \\ \hline \therefore p \land q \end{array}$	Conjunction
$\begin{array}{c} p \lor q \\ \neg p \lor r \\ \hline \therefore q \lor r \end{array}$	Resolution

Mathematical Proofs

- A **mathematical proof** is usually "informal"
- More formal than everyday language, less formal than logical proofs
  - More than one rule may be used in a step
  - (**Some**) steps may be skipped
  - Axioms may be assumed
  - Rules for inference need not be explicitly stated
- Proofs must be a self-contained line of reasoning containing only:
  - facts (axioms)
  - Theorems, lemmas, corollaries (previously proven statements), or
  - statements derived from the above

You cannot use something as fact within a proof if you are not certain that it is

Some Terminology

- **Theorem**: statement that can be shown true
  - **Proposition**: less important theorem
  - **Lemma**: less important theorem used to prove other theorems
  - **Corollary**: theorem that trivially follows another theorem
- **Conjecture**: statement proposed to be true, but not yet proven
- **Axiom**: statement assumed to be true (does not need a proof)
- Most axioms, theorems, etc are universal over some domain
  - ie all perfect squares are non-negative
  - the domain should be clear from context, or explicitly stated

Proof Method: Proof by Exhaustion

A **proof by exhaustion** for $p \rightarrow q$ starts by considering each element of the domain of discourse and showing the predicate is true

Only useful when dealing with a small domain

- Small is relative, but must be finite
- Example: {2,4,6} is a small and finite domain

This is a special type of *proof by cases*

Direct Proofs

A **direct proof** for $P(x) \to Q(x)$ starts by assuming $P(x)$ as fact, and finishes by establishing $Q(x)$

It makes use of axioms, previously proven theorems, inference rules, etc

Same approach as proving a logical argument is valid

- $P(x)$ is the hypothesis
- $Q(x)$ is the conclusion

Proof by Contraposition

A **proof by contraposition** for $P(x) \rightarrow Q(x)$ is a proof where you:

1. Write a direct proof for $\neg Q(x) \rightarrow \neg P(x)$
2. Conclude that the contrapositive, $P(x) \rightarrow Q(x)$, is also true

Proof Layout:
Assume $\neg Q(x)$
Perform your derivations (using theorems, axioms, etc)
$\therefore \neg P(x)$

Since $\neg Q(x) \rightarrow \neg P(x)$ is TRUE, we may conclude that our original statement $P(x) \rightarrow Q(x)$ is also TRUE

Proof by Contradiction

Note that $\boldsymbol{p}$ is logically equivalent to $\neg \boldsymbol{p} \rightarrow (\boldsymbol{r} \wedge \neg \boldsymbol{r})$

A **proof by contradiction** for $\boldsymbol{p}$ is actually a proof for $\neg \boldsymbol{p} \rightarrow (\boldsymbol{r} \wedge \neg \boldsymbol{r})$ where you:

1. Write a proof starting with the asumption $\neg \boldsymbol{p}$
2. Find some proposition $\boldsymbol{r}$ where you can derive both $\boldsymbol{r}$ and $\neg \boldsymbol{r}$ to be TRUE (a contradiction)

Proof Layout:
Assume $\neg \boldsymbol{p}$
Find something that breaks
$\therefore$ contradiction, so $\boldsymbol{p}$ has to be true

Additional Review
[Chapter 1 Review/Summary Exercises]

Sets
[Chapter 2.1, 2.2]

Sets

A **set** is a **collection of objects** that do NOT have an order

Each object is called an **element** or **member** of the set

Notation:

- $\mathbf{e} \in \boldsymbol{S}$ means that **e** is an element of ***S***
- $\mathbf{e} \notin \boldsymbol{S}$ means that **e** is not an element of ***S***

Common Sets

- $\mathbb{N} = \{1, 2, 3, \ldots\}$: the set of **natural numbers**
  - Sometimes 0 is considered a member, which some disagree with
- $\mathbb{Z} = \{0, -1, 1, -2, 2, \ldots\}$: the set of **integers**
- $\mathbb{Z}^+ = \{1, 2, 3, \ldots\}$: the set of **positive integers**
- $\mathbb{Q} = \{ p/q \mid p \in \mathbb{Z}, q \in \mathbb{Z}, q \neq 0\}$: the set of **rational numbers**
  - Numbers that can be written as a fraction of integers
- $\mathbb{Q}^+ = \{ x \mid x \in \mathbb{Q}, x > 0\}$: the set of **positive rational numbers**
- $\mathbb{R}$: the set of **real numbers**
- $\mathbb{R}^+ = \{x \mid x \in \mathbb{R}, x > 0\}$: the set of **positive real numbers**
- $\mathbb{C}$: the set of complex numbers

Cardinality (for Finite Sets)

If a set ***A*** contains exactly ***n*** elements, where ***n*** is a non-negative integer, then ***A*** is a **finite set**.

n is called the **cardinality** of ***A***, denoted by |***A***|.

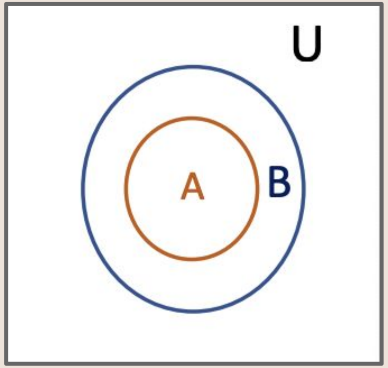
The **empty set** or **null set** is the set that contains no elements, denoted by ∅ or {}. It has size 0.

Subsets

A set $\boldsymbol{A}$ is a **<u>subset</u>** of $\boldsymbol{B}$ if and only if every element of $\boldsymbol{A}$ is also in $\boldsymbol{B}$.
Denoted by $\boldsymbol{A} \subseteq \boldsymbol{B}$
If $\boldsymbol{A} \subseteq \boldsymbol{B}$, then $\forall \boldsymbol{x} \in \boldsymbol{A}, \boldsymbol{x} \in \boldsymbol{B}$

Note: for any set $\boldsymbol{A}$, $\varnothing \subseteq \boldsymbol{A}$ and $\boldsymbol{A} \subseteq \boldsymbol{A}$

If $\boldsymbol{A} \subseteq \boldsymbol{B}$ but $\boldsymbol{A} \neq \boldsymbol{B}$, then $\boldsymbol{A}$ is a **<u>proper subset</u>** of $\boldsymbol{B}$.
Denoted by $\boldsymbol{A} \subset \boldsymbol{B}$ or $\boldsymbol{A} \subsetneq \boldsymbol{B}$



Venn Diagram showing A ⊆ B

Set Equality

Fact: Suppose A and B are sets. Then $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$

To Prove Set Equality

Prove $A \subseteq B$:

Assume x in A

…

$\therefore$ x in B as well

Conclude that $A \subseteq B$

Prove $B \subseteq A$:

Assume y in B

…

$\therefore$ y in A as well

Conclude that $B \subseteq A$

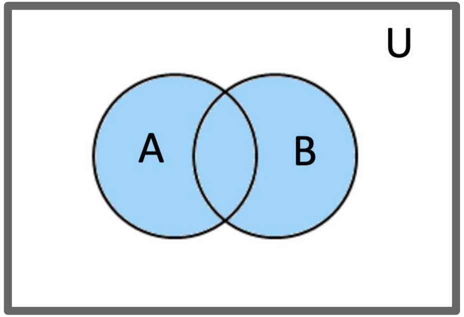
Conclude that since $A \subseteq B$ and $B \subseteq A$ then $A = B$

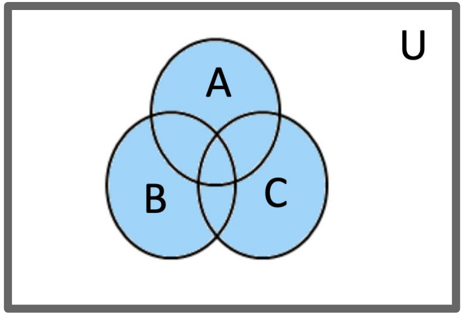
Set Union

The **union** of two sets, ***A*** and ***B***, is the set that contains exactly all elements that are in ***A*** or ***B*** (or in both)

- Denoted by $\boldsymbol{A} \cup \boldsymbol{B}$
- Formally, $\boldsymbol{A} \cup \boldsymbol{B} = \{ x \mid x \in \boldsymbol{A} \text{ or } x \in \boldsymbol{B} \}$

$\boldsymbol{A} \cup \boldsymbol{B}$ is shaded →





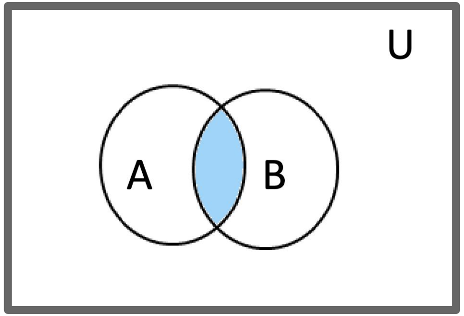
← $\boldsymbol{A} \cup \boldsymbol{B} \cup \boldsymbol{C}$ is shaded

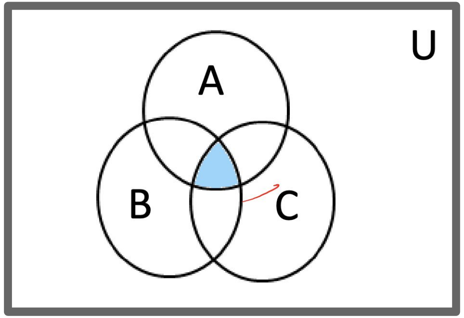
Set Intersection

The **intersection** of two sets, ***A*** and ***B***, is the set that contains exactly all elements that are in ***A*** and ***B***

- Denoted by $\boldsymbol{A \cap B}$
- Formally, $\boldsymbol{A \cap B} = \{ x \mid x \in \boldsymbol{A} \text{ and } x \in \boldsymbol{B} \}$

$\boldsymbol{A \cap B}$ is shaded →





← $\boldsymbol{A \cap B \cap C}$ is shaded

Set Intersection

The **intersection** of two sets, $\boldsymbol{A}$ and $\boldsymbol{B}$, is the set that contains exactly all elements that are in $\boldsymbol{A}$ and $\boldsymbol{B}$

- Denoted by $\boldsymbol{A} \cap \boldsymbol{B}$
- Formally, $\boldsymbol{A} \cap \boldsymbol{B} = \{ x \mid x \in \boldsymbol{A} \text{ and } x \in \boldsymbol{B} \}$

Two sets are **disjoint** if their intersection is the empty set

Principle of Inclusion-Exclusion

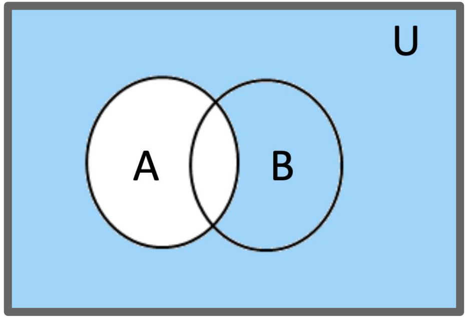
$|\boldsymbol{A} \cup \boldsymbol{B}| = |\boldsymbol{A}| + |\boldsymbol{B}| - |\boldsymbol{A} \cap \boldsymbol{B}|$

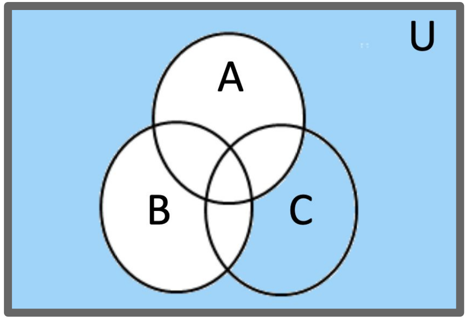
Set Complement

The **complement** of set $\boldsymbol{A}$ is the set that contains exactly all the elements that are not in $\boldsymbol{A}$.

- Denoted by $\bar{\boldsymbol{A}}$
- Formally, $\bar{\boldsymbol{A}} = \{ x \mid x \notin \boldsymbol{A} \}$

$\bar{\boldsymbol{A}}$ is shaded →





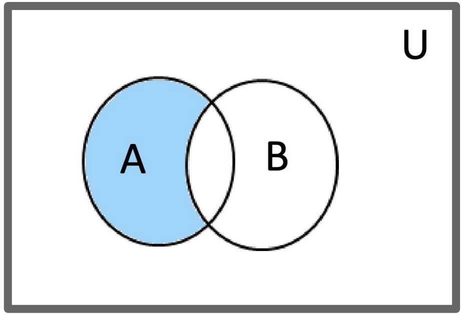
← $\overline{\boldsymbol{A} \cup \boldsymbol{B}}$ is shaded

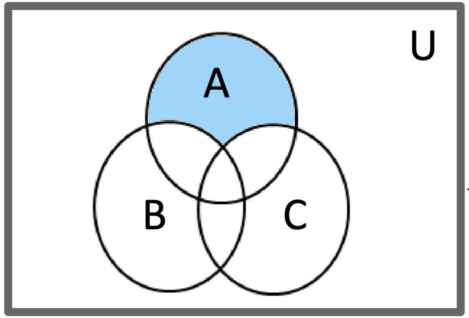
Set Difference

The **difference** of set ***A*** and set ***B*** is the set that contains exactly all elements that are in ***A*** but not in ***B***

- Denoted by ***A* - *B*** (or ***A* \ *B***)
- Formally, $\boldsymbol{A - B} = \{\, x \mid x \in \boldsymbol{A} \text{ and } x \notin \boldsymbol{B} \,\} = \boldsymbol{A} \cap \overline{\boldsymbol{B}}$

A* - *B is shaded →





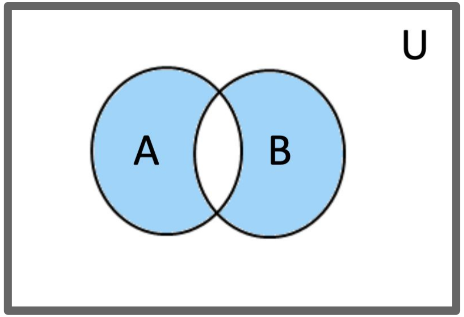
← ***A* - (*B* ∪ *C*)** is shaded

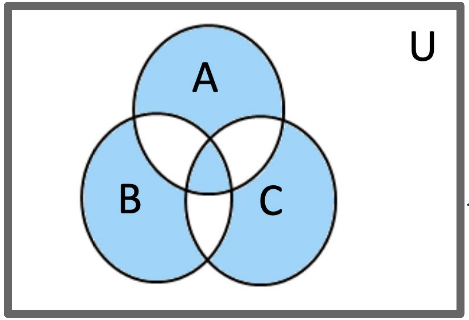
Symmetric Difference

The **symmetric difference** of set ***A*** and set ***B*** is the set that contains all elements that are in exactly one of ***A*** or ***B***

- Denoted by $\boldsymbol{A \oplus B}$ (or $\boldsymbol{A} \bigcirc \boldsymbol{B}$)
- Formally, $\boldsymbol{A \oplus B = (A - B)} \cup \boldsymbol{(B - A)}$

$\boldsymbol{A \oplus B}$ is shaded $\rightarrow$





$\leftarrow$ $\boldsymbol{A \oplus B \oplus C}$ is shaded

It includes values that are in an odd number of sets, ie $\{ x \mid x \in \boldsymbol{A} \oplus x \in \boldsymbol{B} \oplus x \in \boldsymbol{C} \}$

Power Set

The **power set** of set $\boldsymbol{A}$ is the set of all possible subsets of $\boldsymbol{A}$

Denoted by $\mathscr{P}(\boldsymbol{A})$

In general, $|\mathscr{P}(\boldsymbol{A})| = 2^{|\boldsymbol{A}|}$

For any set $\boldsymbol{A}$, it is always the case that:

- $\varnothing \in \mathscr{P}(\boldsymbol{A})$ (the empty set is a subset of $\boldsymbol{A}$ …and every other set)
- $\boldsymbol{A} \in \mathscr{P}(\boldsymbol{A})$ ($\boldsymbol{A}$ is a subset of itself…every elements of $\boldsymbol{A}$ is in $\boldsymbol{A}$)

Imposing Order on Elements

Sometimes order **is** important…

How can we impose order on elements?

An **<u>ordered *n* tuple</u>** $(\boldsymbol{a_1}, \boldsymbol{a_2}, \ldots, \boldsymbol{a_n})$ has $\boldsymbol{a_1}$ as its first element, $\boldsymbol{a_2}$ as its second, …, and $\boldsymbol{a_n}$ as its $\boldsymbol{n^{th}}$ element.

Order is important for tuples. Assume $\boldsymbol{a_1} \neq \boldsymbol{a_2}$

- $(\boldsymbol{a_1}, \boldsymbol{a_2}) \neq (\boldsymbol{a_2}, \boldsymbol{a_1})$ ← tuple comparison
- $\{\boldsymbol{a_1}, \boldsymbol{a_2}\} = \{\boldsymbol{a_2}, \boldsymbol{a_1}\}$ ← set comparison

Imposing Order: Cartesian Product

The **Cartesian product** of two sets $\boldsymbol{A_1}$ and $\boldsymbol{A_2}$ is defined as the set of ordered tuples $(\boldsymbol{a_1}, \boldsymbol{a_2})$ where $\boldsymbol{a_1} \in \boldsymbol{A_1}$ and $\boldsymbol{a_2} \in \boldsymbol{A_2}$

- Denoted by $\boldsymbol{A_1} \times \boldsymbol{A_2}$
- Formally, $\boldsymbol{A_1} \times \boldsymbol{A_2} = \{(\boldsymbol{a_1}, \boldsymbol{a_2}) \mid \boldsymbol{a_1} \in \boldsymbol{A_1} \text{ and } \boldsymbol{a_2} \in \boldsymbol{A_2}\}$
- We say "$\boldsymbol{A_1}$ cross $\boldsymbol{A_2}$"

René Descartes

Strings

An **alphabet** is a *non-empty finite* set of symbols

A **string** is a finite sequence of symbols from an alphabet

- Shorthand for a tuple from the Cartesian power of an alphabet

The number of characters in a string is called th **length** of the string

- The length of string ***s*** is denoted by |***s***|

Pairwise Disjoint Sets

Two sets $\boldsymbol{A}$ and $\boldsymbol{B}$ are **disjoint** iff $\boldsymbol{A} \cap \boldsymbol{B} = \varnothing$

A sequence of sets, $\boldsymbol{A_1}, \boldsymbol{A_2}, \boldsymbol{A_3}, \ldots, \boldsymbol{A_n}$ are **pairwise disjoint** if:
for any $\boldsymbol{i}, \boldsymbol{j} \in \{1,2,3,\ldots,n\}$, where $\boldsymbol{i} \neq \boldsymbol{j}$, we have $\boldsymbol{A_i} \cap \boldsymbol{A_j} = \varnothing$

Symbolically we write $\forall \boldsymbol{i},\boldsymbol{j} \in \{1,2,3,\ldots,n\}$: $[(\boldsymbol{i} \neq \boldsymbol{j}) \rightarrow (\boldsymbol{A_i} \cap \boldsymbol{A_j} = \varnothing)]$

Partitions

A **partition** of a non-empty set $\boldsymbol{A}$ is a list of one or more non-empty subsets of $\boldsymbol{A}$ such that each element of $\boldsymbol{A}$ appears in exactly one of the subsets.

Formally, a partition of $\boldsymbol{A}$ is a list of sets, $A_1, A_2, \ldots, A_k$ such that:

1. $\forall i \in [1, k]$: $A_i \neq \varnothing$ (the sets are non-empty)
2. $\forall i \in [1, k]$: $A_i \subseteq A$ (the sets are subsets of $\boldsymbol{A}$)
3. $\forall i,j \in [1, k]$: $i \neq j \rightarrow A_i \cap A_j = \varnothing$ (the sets are pairwise disjoint)
4. $A = A_1 \cup A_2 \cup A_3 \cup \ldots \cup A_k$

Functions and Relations [Chapter 2.3, 9.1, 9.5, 9.6]

Binary Relations

A **binary relation** between two sets $\boldsymbol{A}$ and $\boldsymbol{B}$ is any set $\boldsymbol{R} \subseteq \boldsymbol{A} \times \boldsymbol{B}$

A binary relation **from** $\boldsymbol{A}$ **to** $\boldsymbol{B}$ is a set $\boldsymbol{R}$ of ordered pairs, where the first element of each ordered pair comes from $\boldsymbol{A}$ and the second from $\boldsymbol{B}$

- For any $\boldsymbol{a} \in \boldsymbol{A}$ and $\boldsymbol{b} \in \boldsymbol{B}$ we say that $\boldsymbol{a}$ is related to $\boldsymbol{b}$ iff $(\boldsymbol{a},\boldsymbol{b}) \in \boldsymbol{R}$
- Denoted by $\boldsymbol{a}\ \boldsymbol{R}\ \boldsymbol{b}$

Note: a relation is a binary predicate **R(*a*,*b*):** "***a*** is related to ***b***"

Example

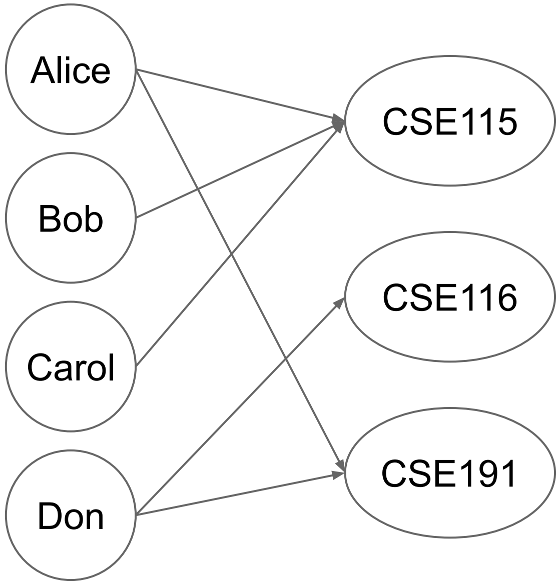
Consider the set of student, ***S* = { Alice, Bob, Carol, Don }**,
and the set of courses, ***C* = { CSE115, CSE116, CSE191 }**

Alice, Bob, and **Carol** are enrolled in **CSE115**

Don is enrolled in **CSE116**

Alice and **Don** are enrolled in **CSE191**

This is called an arrow diagram. It is a visual representation of a binary relation.

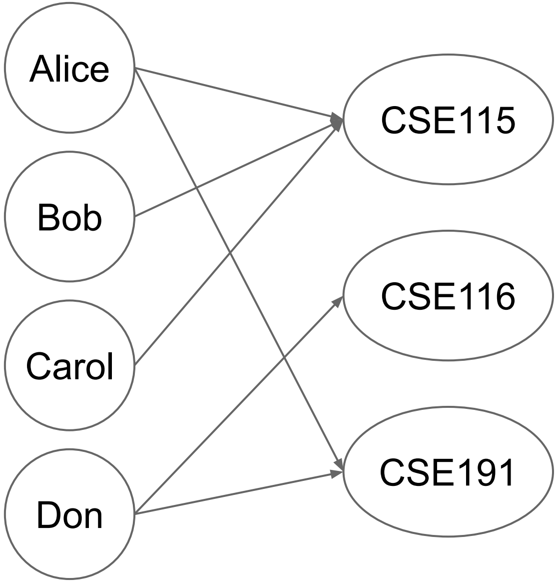


Example

Consider the set of student, ***S*** **= { Alice, Bob, Carol, Don }**,
and the set of courses, ***C*** **= { CSE115, CSE116, CSE191 }**

We can also use **matrix representation** to describe ***E*****:**

$$\begin{array}{cc} & \begin{array}{ccc} \text{CSE115} & \text{CSE116} & \text{CSE191} \end{array} \\ \begin{array}{r} \text{Alice} \\ \text{Bob} \\ \text{Carol} \\ \text{Don} \end{array} & \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \end{array}$$



Binary Relations on a Set

The binary relation R on a set A is a subset of $A \times A$.

The set A is called the **domain** of the binary relation.

Special Properties of Binary Relations

A relation ***R*** on set ***A*** is called **reflexive** if every $\boldsymbol{a} \in \boldsymbol{A}$ is related to itself.

Formally, $\boldsymbol{a}\ \boldsymbol{R}\ \boldsymbol{a}$ for all $\boldsymbol{a} \in \boldsymbol{A}$

Example: Consider the $\leq$ relation on $\mathbb{Z}$

Special Properties of Binary Relations

A relation ***R*** on set ***A*** is called **symmetric** if for every ***a R b***, we also have that ***b R a***.

Example: Consider the = relation on $\mathbb{Z}$

A relation ***R*** on set ***A*** is called **anti-symmetric** if for all ***a, b* ∈ *A*:**
a R b and ***b R a*** implies that ***a* = *b***.

Example: Consider the ≤ relation on $\mathbb{Z}$

Special Properties of Binary Relations

A relation ***R*** on set ***A*** is called **<u>transitive</u>** if for all ***a,b,c*** $\in$ ***A***:
a R b and ***b R c*** implies ***a R c***.

Example: Consider the < relation on $\mathbb{Z}$

Partial Ordering

A relation ***R*** on a set ***A*** is called a **<u>partial order</u>** if it is reflexive, transitive, and antisymmetric.

a R b is denoted ***a*** ≤ ***b*** for partial a ordering ***R***

- We read ***a*** ≤ ***b*** as "***a*** is at most ***b***" or "***a*** precedes ***b***"
- A domain, ***A***, with a partial ordering ≤ can be treated as the object (***A***,≤)
  - (***A***,≤) is called a **<u>partially ordered set</u>** or **<u>poset</u>**

Comparable Elements and Total Ordering

Elements x and y are **comparable** if $x \leq y$ or $y \leq x$ (or both)

A partial order is a **total order** if every pair of elements in the domain are comparable.

In our previous example, $(\mathbb{Z}, R)$ is a total order

- It is a partial order, and for every $x,y \in \mathbb{Z}$, $x\ R\ y$ or $y\ R\ x$
- We say that R is a total ordering of $\mathbb{Z}$

Equivalence Relations

A relation ***R*** on a set ***A*** is called an **equivalence relation** if it is reflexive, transitive, and **symmetric**.

a R b is denoted ***a ~ b*** for an equivalence relation ***R***

- We read ***a ~ b*** as "***a*** is equivalent to ***b***"

Equivalence Classes

We can partition the domain of an equivalence relation into equivalent elements. These partitions are called **equivalence classes**.

If $\boldsymbol{e} \in \boldsymbol{D}$ then the equivalence class containing $\boldsymbol{e}$ is denoted $[\boldsymbol{e}]$

$[\boldsymbol{e}] = \{ \boldsymbol{x} \mid \boldsymbol{x} \in \boldsymbol{D}, \boldsymbol{x} \sim \boldsymbol{e} \}$

Function Definition

Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be nonempty sets. A **function**, f, from $\boldsymbol{A}$ to $\boldsymbol{B}$ is an assignment of exactly one element of $\boldsymbol{B}$ to each element of $\boldsymbol{A}$.

Denoted by f: $\boldsymbol{A} \longrightarrow \boldsymbol{B}$

We write $f(\boldsymbol{a}) = \boldsymbol{b}$ if $\boldsymbol{b}$ is the **unique element** of $\boldsymbol{B}$ assigned by f to the element $\boldsymbol{a}$ of $\boldsymbol{A}$

The set $\boldsymbol{A}$ is the **domain** of f

The set $\boldsymbol{B}$ is the **codomain** of f

Function Range

If f is a function from $\boldsymbol{A}$ to $\boldsymbol{B}$, the set range(f) = $\{\, \boldsymbol{y} \mid \boldsymbol{\exists x} \in \boldsymbol{A}, f(x) = y \,\}$ is called the **range** of f

It is the set of all values in the codomain that have an element from the domain mapped to it

- For any function f: $\boldsymbol{A} \rightarrow \boldsymbol{B}$, range($f$) $\subseteq \boldsymbol{B}$
- It does not have to be the whole codomain

Injective Functions

A function f: $\boldsymbol{A} \rightarrow \boldsymbol{B}$ is **<u>injective</u>** if $\forall \boldsymbol{x_1}, \boldsymbol{x_2} \in \boldsymbol{A}, (f(\boldsymbol{x_1}) = f(\boldsymbol{x_2}) \rightarrow \boldsymbol{x_1} = \boldsymbol{x_2})$

Also known as **<u>one-to-one</u>** or **<u>1-1</u>**

- Each element in the domain is mapped to a unique element from the codomain (no element in the codomain is hit twice)
- To prove a function is 1-1
  - Take an arbitrary $\boldsymbol{x}$ and $\boldsymbol{y}$ such that $f(\boldsymbol{x}) = f(\boldsymbol{y})$
  - Conclude that $\boldsymbol{x} = \boldsymbol{y}$
- To prove a function is not 1-1
  - Find a counterexample where $\boldsymbol{x} \neq \boldsymbol{y}$ but $f(\boldsymbol{x}) = f(\boldsymbol{y})$

Surjective Functions

A function f: $A \rightarrow B$ is **surjective** if $\forall y \in B,\ \exists x \in A, f(x) = y$

Also known as **onto**

- Every element in the codomain has an element that maps to it
- To prove a function is onto:
  - Take arbitrary y in the codomain
  - Find the value of x in the domain such that $f(x) = y$
- To prove a function is not onto:
  - Find a counterexample, element y in codomain s.t. no element maps to it

Bijective Functions

A function $f\colon \boldsymbol{A} \rightarrow \boldsymbol{B}$ is **bijective** if it is injective and surjective

A bijective function is called a **bijection**, or a **one-to-one correspondence**

Inverse of Functions

For any function f: $\boldsymbol{A} \to \boldsymbol{B}$, the **inverse mapping** of f, denoted by f^{-1}, is defined by the mapping f^{-1}: $\boldsymbol{B} \to \boldsymbol{A}$ where: $f^{-1} = \{ (\boldsymbol{y}, \boldsymbol{x}) \mid (\boldsymbol{x}, \boldsymbol{y}) \in f \}$

If f is a **bijection** then f^{-1} is a function (otherwise it is just a mapping)

- f^{-1} maps codomain elements of f to domain elements of f
- If f(x) = y then f^{-1}(y) = x

Floor Functions

The floor function is the function floor : $\mathbb{R} \to \mathbb{Z}$ defined by

$$\text{floor}(\boldsymbol{x}) = \max\{\, \boldsymbol{y} \mid \boldsymbol{y} \in \mathbb{Z}, \boldsymbol{y} \leq \boldsymbol{x} \,\}$$

Evaluates to the maximum integer below the given number.

Denoted by: floor($\boldsymbol{x}$) = $\lfloor \boldsymbol{x} \rfloor$

Examples

$\lfloor 4.5 \rfloor = 4$

$\lfloor 17 \rfloor = 17$

$\lfloor -8.7 \rfloor = -9$

$\lfloor \boldsymbol{\pi} \rfloor = \lfloor 3.14159 \rfloor = 3$

Ceiling Function

The ceiling function is the function floor : $\mathbb{R} \to \mathbb{Z}$ defined by

$$\text{ceiling}(\boldsymbol{x}) = \min\{\, \boldsymbol{y} \mid \boldsymbol{y} \in \mathbb{Z}, \boldsymbol{y} \geq \boldsymbol{x} \,\}$$

Evaluates to the minimum integer above the given number.

Denoted by: ceiling($\boldsymbol{x}$) = $\lceil \boldsymbol{x} \rceil$

Examples

$\lceil 4.5 \rceil = 5$

$\lceil 17 \rceil = 17$

$\lceil -8.7 \rceil = -8$

$\lceil \boldsymbol{\pi} \rceil = \lceil 3.141159 \rceil = 4$

Divides

Let x and y be integers. Then x **<u>divides</u>** y if there is an integer k s.t. $y = kx$.

Denoted by $x \mid y$

- x does not divide y is denoted by $x \nmid y$

If $x \mid y$, then we say:

- y is a multiple of x
- x is a factor or divisor of y

Integer Division Definition

The Division Algorithm for $\boldsymbol{n} \in \mathbb{Z}$ and $\boldsymbol{d} \in \mathbb{Z}^+$ gives unique values

$$\boldsymbol{q} \in \mathbb{Z} \text{ and } \boldsymbol{r} \in \{0, \dots, \boldsymbol{d} - 1\}.$$

- The number ***q*** is called the quotient.
- The number ***r*** is called the remainder.

The operations ***div*** and ***mod*** produce the quotient and the remainder, respectively, as a function of ***n*** and ***d***.

- ***n div d = q***
- ***n mod d = r***

In programming, ***n*** % ***d*** = ***r*** denotes ***n mod d*** = ***r***

Addition mod n

For any integer $n > 0$, $x \bmod n$ can be seen as a function $mod_n(x)$:

- $mod_n: \mathbb{Z} \rightarrow \{0, 1, 2, \ldots, n - 1\}$, where $mod_n(x) = x \bmod n$.

Addition $mod\ n$ is defined by adding two numbers and then applying mod_n

- All results in the range $\{0, 1, \ldots, n - 1\}$

Suppose $n = 7$

$+mod_7(4, 6) = (4 + 6) \bmod 7 = 10 \bmod 7 = 3$

$+mod_7(15, 17) = (15 + 17) \bmod 7 = 32 \bmod 7 = 4$

$+mod_7(8, 20) = (8 + 20) \bmod 7 = 28 \bmod 7 = 0$

Multiplication mod *n*

Multiplication ***mod n*** is defined by multiplying two numbers and then applying $\boldsymbol{mod}_n$.

- All results in the range { 0, 1, ... , ***n*** − 1 }

Suppose ***n*** = 11.

$* \; \boldsymbol{mod}_{11}(4, 6) = (4 * 6) \; \boldsymbol{mod} \; 11 = 24 \; \boldsymbol{mod} \; 11 = 2$

$* \; \boldsymbol{mod}_{11}(5, 7) = (5 * 7) \; \boldsymbol{mod} \; 11 = 35 \; \boldsymbol{mod} \; 11 = 2$

$* \; \boldsymbol{mod}_{11}(8, 23) = (8 * 23) \; \boldsymbol{mod} \; 11 = 184 \; \boldsymbol{mod} \; 11 = 8$

Congruence Modulo

If ***a*** and ***b*** are integers and ***m*** is a positive integer, then ***a* is congruent to *b* modulo *m*** if ***m*** divides ***a*** − ***b***.

The notation ***a*** ≡ ***b*** (***mod m***) indicates that ***a*** is congruent to ***b*** modulo ***m***.

- ***a*** ≡ ***b*** (***mod m***) is a congruence.
- Indicates that a and b are in the same equivalence class.

Is 17 congruent to 5 modulo 6? **Yes**, because 6 divides 17 - 5.

- 17 ***mod*** 6 = 5; it is in the equivalence class for 5 in ***mod*** 6.
- 5 ***mod*** 6 = 5; it is in the equivalence class for 5 in ***mod*** 6.

Composition of Functions

If f and g are two functions, where f: $X \rightarrow Y$ and g: $Y \rightarrow Z$, the **composition of g with f,** denoted by $g \circ f$, is the function:

$$(g \circ f): X \rightarrow Z, \text{ s.t. for all } x \in X, (g \circ f)(x) = g(f(x))$$

Sequences
[Chapter 2.4, 5.1, 5.2]

Sequences: Terminology

A **sequence** is created by a special type of function with a domain of consecutive integers…ie no gaps in the domain

OK: $\mathbb{N}$, $\mathbb{Z}^+$, $\mathbb{Z}^+ \cup \{0\}$, $\mathbb{Z}$, $\{1,2,34,5\}$

Not OK: $\{1,3,5,7\}$, $\{ x \in \mathbb{N} \mid x \text{ is even} \}$

Increasing Sequences

A sequence $\{a_k\}$ is **increasing** if, $\forall i, a_i < a_{i+1}$

A sequence $\{a_k\}$ is **non-decreasing** if, $\forall i, a_i \le a_{i+1}$

$d_k = k$ for $1 \le k \le 10$ is **increasing**

$e_k = 2k$ for $k \ge 1$ is **increasing**

$f_k = 2^k$ for $k \ge 0$ is **increasing**

What about the sequence $\{ h_k \} = 1, 2, 2, 2, 3$? **Non-Decreasing**

Every increasing sequence is non-decreasing

Not every non-decreasing sequence is increasing

Decreasing Sequences

A sequence $\{a_k\}$ is **decreasing** if, $\forall i, a_i > a_{i+1}$

A sequence $\{a_k\}$ is **non-increasing** if, $\forall i, a_i \geq a_{i+1}$

$s_k = 10 - k$ for $1 \leq k \leq 10$ is **decreasing (and non-increasing)**

$t_k = -2k$ for $k \geq 1$ is **decreasing (and non-increasing)**

$u_k = 2^{-k}$ for $k \geq 0$ is **decreasing (and non-increasing)**

What about the sequence $\{vh_k\} = 3, 2, 2, 2, 1$? **Non-Increasing**

Geometric Sequences

A **geometric sequence** is a sequence formed by successively multiplying the initial term by a fixed number called the **common ratio**.

Examples:

$\{a_k\}$ is 1, -1, 1, -1, 1, -1, 1, -1, … $\rightarrow a_k = a_0 \cdot r^k = 1 \cdot (-1)^k$ for all $k \geq 0$

$\{b_k\}$ is 1, ½, ¼, ⅛, … $\rightarrow b_k = b_0 \cdot r^k = 1 \cdot (½)^k$ for all $k \geq 0$

What are the explicit formulas for the above sequences?

For any geometric sequence $\{s_k\}$ with initial term s_0 and common ratio r:

$$s_k = s_0 \cdot r^k, \text{ for } k \geq 0$$

Arithmetic Sequences

An **arithmetic sequence** is a sequence formed by successively adding a fixed number, called the **common difference**, to the initial term.

Examples:

$\{a_k\}$ is 5, 15, 25, 35, 45, … $\rightarrow a_k = a_0 \cdot kd = 5 + 10 \cdot k$ for all $k \geq 0$

$\{b_k\}$ is 49, 42, 35, 28, 21, … $\rightarrow b_k = b_0 \cdot kd = 49 + (-7)k$ for all $k \geq 0$

What are the explicit formulas for the above sequences?

For any arithmetic sequence $\{s_k\}$ with initial term s_0 and common diff d:

$$s_k = s_0 + kd, \text{ for } k \geq 0$$

Summations

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

Consider the sequence: $a_0, a_1, a_2, a_3, \ldots, a_k$

We can express the sum of all elements in the sequence as: $$\sum_{i=0}^{k} a_i$$

What this represents is: $$\sum_{i=0}^{k} a_i = a_0 + a_1 + a_2 + \ldots + a_k$$

Sequences: Recurrence Relations

A **recurrence relation** for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more of the previous terms (a_0, a_1, a_2, … a_{n-1}), for all integers n with $n \geq n_0$, where n_0 is a nonnegative integer.

- A recurrence relation is said to ***recursively define*** a sequence
- It may have one or more initial terms
- A sequence is called a ***solution*** of a recurrence relation if its terms satisfy the recurrence relation

Mathematical Induction

Principle of Mathematical Induction:

Let $\boldsymbol{P(n)}$ be a statement defined for any $\boldsymbol{n} \in \mathbb{N}$. If the following hold:

- $\boldsymbol{P}(1)$ is true
- For all $\boldsymbol{k} \in \mathbb{N}$, $\boldsymbol{P(k)} \rightarrow \boldsymbol{P(k+1)}$

Then $\boldsymbol{P(n)}$ is true for all $\boldsymbol{n} \in \mathbb{N}$

Note: We can relax this to apply to any domain $\boldsymbol{D}$ of consecutive integers

To prove the inductive step, assume $\boldsymbol{P(k)}$, then derive $\boldsymbol{P(k+1)}$

Strong Induction

Principle of Strong Mathematical Induction

Let $\boldsymbol{a}, \boldsymbol{b}$ be integers with $\boldsymbol{a} \leq \boldsymbol{b}$

Let $\boldsymbol{P}(\boldsymbol{n})$ be a statement defined for any integer $\boldsymbol{n} \geq \boldsymbol{a}$

Then $\boldsymbol{P}(\boldsymbol{n})$ is true for all $\boldsymbol{n} \geq \boldsymbol{a}$ if the following two conditions hold:

1. $\boldsymbol{P(a), P(a+1), \ldots, P(b)}$ are all individually true (the base cases)
2. For all $\boldsymbol{k} \geq \boldsymbol{b}$, $\boldsymbol{P(a) \land P(a+1) \land \ldots \land P(k) \rightarrow P(k+1)}$ (the inductive case)

Proof Template

To formally prove something via strong induction, you must do **all** of the following:

1. Express the statement being proved in the form " $\forall n \geq a, P(n)$ for a fixed integer a
2. Prove the **Base Cases**: show that $P(a), P(a + 1), \ldots, P(b)$ are all true.
3. Prove the **Inductive Case**
   a. Inductive Hypothesis: Assume $P(i)$ is true for all i, where $a \leq i \leq k$ for an arbitrary $k \geq b$
   b. State what must be proved under this assumption; write out $P(k + 1)$
   c. Prove the statement $P(k + 1)$ is true by using the inductive assumption
   d. Clearly identify the conclusion
4. Now that you have proven the Base Case and Inductive Case, conclude that $P(n)$ is true for all $n \geq a$ by the principle of strong mathematical induction

Counting
[Chapter 6.1-6.3]

Product Rule

The Product Rule

Suppose that a procedure can be broken down into a sequence of 2 tasks.

If there are n_1 ways to do the first task, and for each of these ways of doing the first task, there are n_2 ways to do the second task, then there are $n_1 \cdot n_2$ ways to do the procedure.

The product rule can be phrased in terms of sets:

Let $A_1, A_2, \ldots, A_n$ be finite sets. Then $|A_1 \times A_2 \times \ldots \times A_n| = |A_1| \cdot |A_2| \cdot \ldots \cdot |A_n|$

Sum Rule

The Sum Rule

If a task can be done either in one of n_1 ways **OR** in one of n_2 ways, where none of the of n_1 ways is the same as any of the n_2 ways, then there are $n_1 + n_2$ ways to do the task.

The sum rule can be phrased in terms of sets:

Let $A_1, A_2, \dots, A_n$ be **mutually disjoint**. Then $|A_1 \cup A_2 \cup \dots \cup A_n| = |A_1|+|A_2|+\dots+|A_n|$

Permutations

A **permutation** of a set of *distinct* objects is an **ordered arrangement** of these objects.

An ordered arrangement of ***r*** elements of a set is called an ***r*-permutation**. The number of ***r*-permutations** of a set with ***n*** elements is denoted by ***P*(*n*,*r*)** or ***nPr***.

Factorial

Let $n \geq 0$ be an integer. The **factorial** of n, denoted by $n!$ is defined by:

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdot (n - 3) \cdot \ldots \cdot 2 \cdot 1$$

Note: For convenience, we define $0! = 1$.

We can also write it as a recurrence relation:

$$a_0 = 1$$

$$a_n = n \cdot a_{n-1} \text{ for } n > 0$$

Permutations and Factorial

<u>Theorem</u>

If n is a positive integer, and r is an integer s.t. $1 \leq r \leq n$, then there are

$$P(n,r) = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \ldots \cdot (n-r+1)$$

***r*-permutations** of a set with n distinct elements

Combinations

A **combination** of a set of *distinct* objects is an **unordered arrangement** of these objects. An ***r*-combination** is simply a subset with ***r*** elements

The number of ***r*-combinations** of a set with ***n*** elements is denoted by

C*(*n*,*r*)** or ***nCr or $\binom{n}{r}$. Sometimes referred to as ***n*** choose ***r***.

Combinations

Theorem

For any non-negative integers ***n*** and ***r*** s.t. **0 ≤ *r* ≤ *n***:

$$C(n,r) = \frac{n!}{r!(n-r)!} = \frac{n \cdot (n-1) \cdots (n-r+1)}{r \cdot (r-1) \cdots 2 \cdot 1}$$

…Therefore **C(5,3) = 5!/(3!2!) = 120 / (6*2) = 10**

Pigeonhole Principle

Pigeonhole Principle: If you put k pigeons into n pigeonholes, with $k > n$, then at least one pigeonhole contains at least two pigeons.

In terms of functions: If f: $A \rightarrow B$ where the codomain has size $|B| = n$ and the domain $|A| = k$ where $k > n$, then f must map at least 2 domain items to the same codomain element.

Pigeonhole Principle

Generalized Pigeonhole Principle:

If you put k objects into n boxes, then at least one box contains at least $\lceil k/n \rceil$ objects.

Basically, you cannot put a fraction of an item in a box (or more gruesomely…you cannot split up one pigeon into multiple boxes).

The fractional item gets rounded up (ceiling function)

Contrapositive Pigeonhole Principle

Contrapositive of the generalized pigeonhole principle:

Suppose you have ***k*** elements and ***n*** boxes. In order to guarantee that there is a box that contains at least ***b*** items, ***k*** must be at least ***n**(b-1) + 1.**

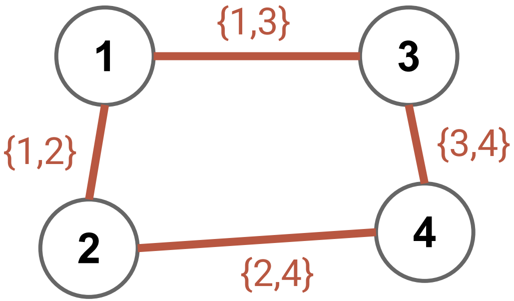
Graphs
[Chapter 10.1 - 10.5, 10.8, 11.1]

Graph Definition

An <u>**(undirected) graph**</u> ***G* = (*V*, *E*)** consists of ***V***, a nonempty set of **vertices** (or **nodes**) and a set of **edges**.

- Each edge has one or two vertices associated with it, called ***endpoints***
- An edge, **{*u*, *v*}** is said to **connect** its endpoints ***u*** and ***v***

Vertices (V)
Edges (E)



Undirected vs Directed

Undirected Edge: $\{u, v\}$ represented as a set

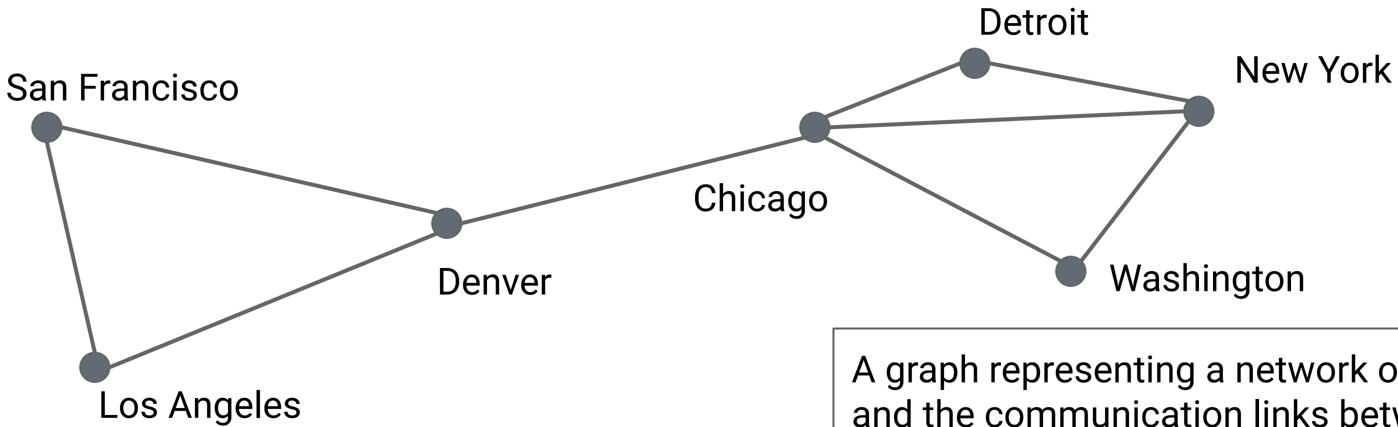
- Order doesn't matter ($\{u, v\} = \{v, u\}$)
- Represents a symmetric relationship

Directed Edge: (u, v) represented as a tuple

- Order does matter, (u, v) and (v, u) are not the same
- Both edges may not exist
- Represents an asymmetric relationship (ie a one-way street)

Simple Graph

A **Simple Graph** is a graph in which every edge connects two **different** vertices and where no two edges connect the same pair of vertices

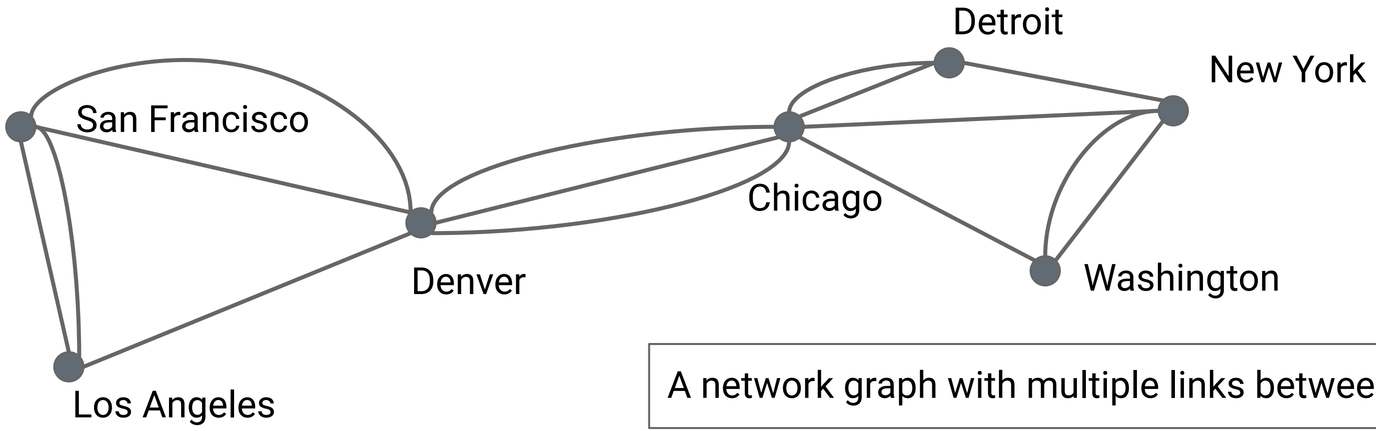


A graph representing a network of data centers and the communication links between them

Multigraph

A graph that may have **multiple edges** connected the same vertices are called **<u>multigraphs</u>**.

- When there are ***m*** different edges associated with the same pair of vertices, **{*u*, *v*}**, we say that edge **{*u*, *v*}** has **<u>multiplicity</u> *m*.**

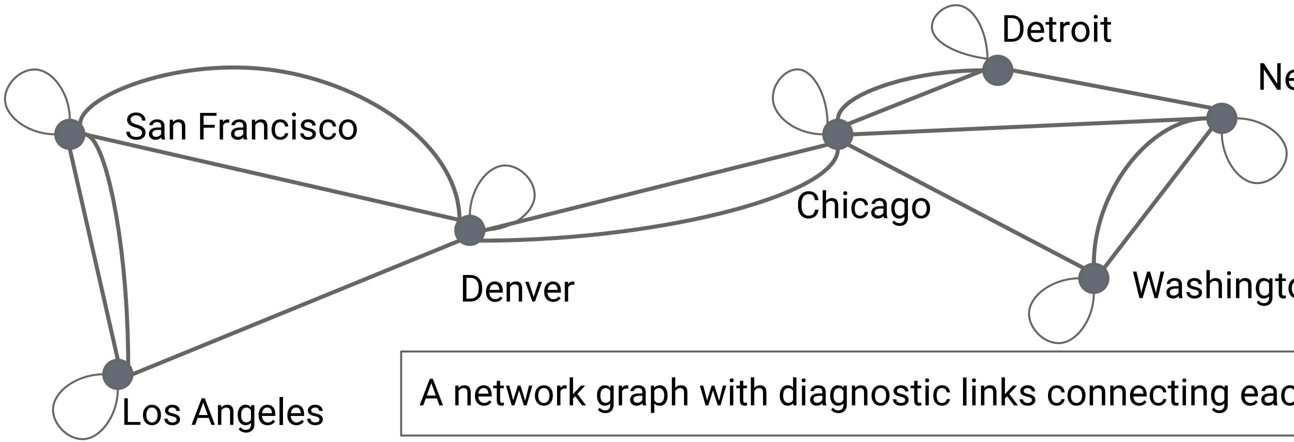


A network graph with multiple links between computers

Graphs with Loops

The edges that connect a given vertex to itself are called **loops** or sometimes **self-loops**.

- Graphs that may include loops and/or multiple edges between the same pair of vertices are sometimes called **pseudographs**.

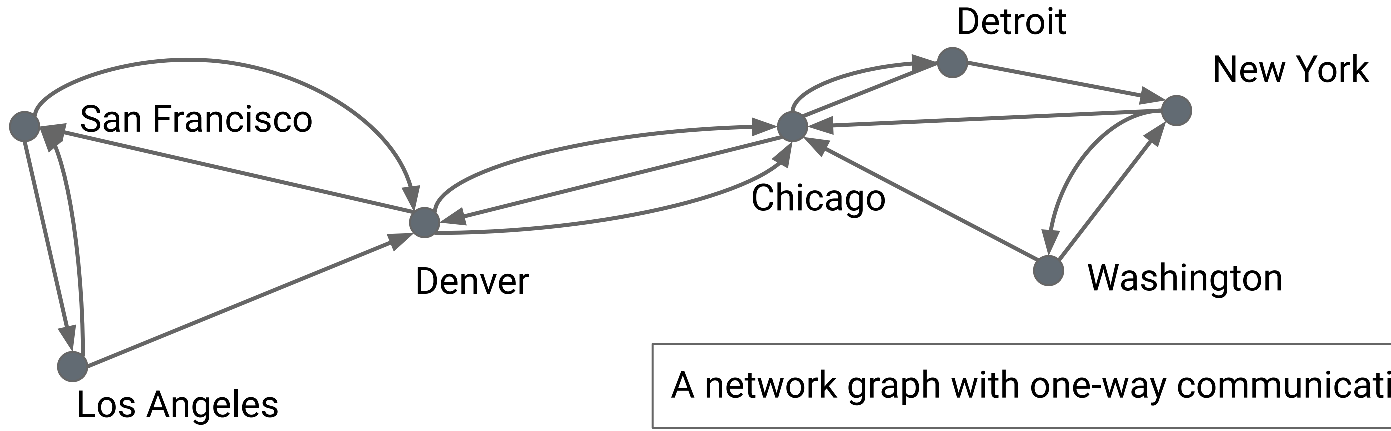


A network graph with diagnostic links connecting each data center to itself

Directed Graph

A **directed graph** (or **digraph**) **(*V*, *E*)** consists of a nonempty set of vertices ***V*** and a set of ***directed edges*** (or ***arcs***) ***E***.

- Each directed edge is associated with an ordered pair of vertices
- The directed edge **(*u*, *v*)** is said to start at ***u*** and end at ***v***



A network graph with one-way communication links

More Terminology

In an undirected graph:

Two vertices are **<u>adjacent</u>** (or **<u>neighbors</u>**) if they are endpoints of an edge

An edge is **<u>incident with</u>** (or **<u>connecting</u>**) its endpoints

The **<u>degree of a vertex</u>** is ***v***, denoted by **deg(*v*)** is the number of edges incident with ***v***

Note: *A **loop** is an edge of the form **{v,v}**, and it adds to the degree of **v** twice*

More Terminology

The set of all neighbors of a vertex ***v*** of ***G* = (*V*, *E*)**, denoted by ***N*(*v*)**, is called the **<u>neighborhood</u>** of ***v***

If ***A*** is a subset of ***V***, then the neighborhood of ***A***, or ***N*(*A*)** is the set of all vertices that are adjacent to at least one vertex in ***A***. So $\boldsymbol{N(A)} = \bigcup_{v \in A} \boldsymbol{N(v)}$

More Graph Terminology

In a graph with directed edges

The **in-degree** of a vertex ***v***, denoted by $\deg^{-}(\boldsymbol{v})$, is the number of edges with ***v*** as their terminal (or ending) vertex

The **out-degree** of a vertex ***v***, denoted by $\deg^{+}(\boldsymbol{v})$, is the number of edges with ***v*** as their initial (or starting) vertex

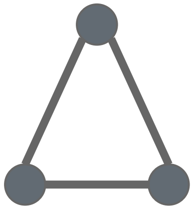
Note: *A loop at vertex* ***v*** *contributes 1 to both the in and out degree of* ***v***

Special Simple Graphs

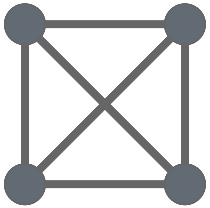
A **complete graph on *n* vertices**, denoted by K_n, is a simple graph that contains exactly one edge between each pair of distinct vertices.

K_1

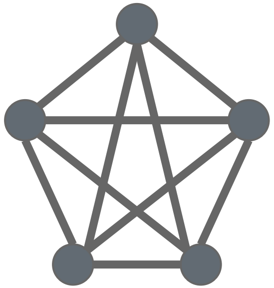
K_2



K_3



K_4



K_5

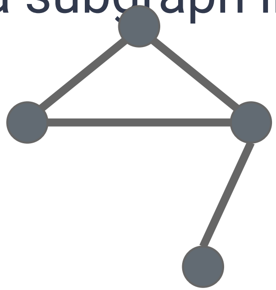
Subgraphs

A **subgraph** of a graph ***G*=(*V*,*E*)** is a graph ***H*=(*W*,*F*)** where $W \subseteq V$ and $F \subseteq E$.

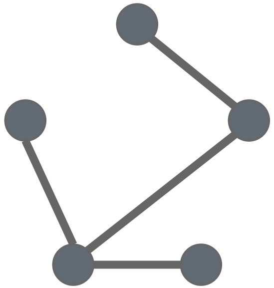
A subgraph, ***H*** of ***G*** is a **proper subgraph** of ***G*** if $H \neq G$.

A **spanning subgraph** is a subgraph in which $W = V$

K_5



A subgraph of K_5

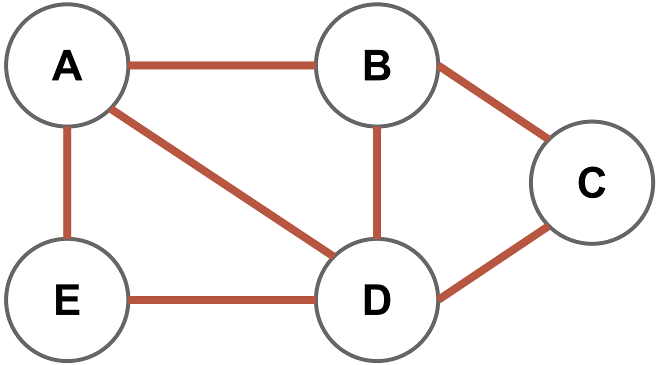


A spanning subgraph of K_5

Adjacency List

In the **Adjacency List** representation of a graph, each vertex has a list of all of its neighbors.

Note: *if the graph is undirected, then if* ***a*** *is in* ***b****'s list of neighbors,* ***b*** *must also be in* ***a****'s list of neighbors*

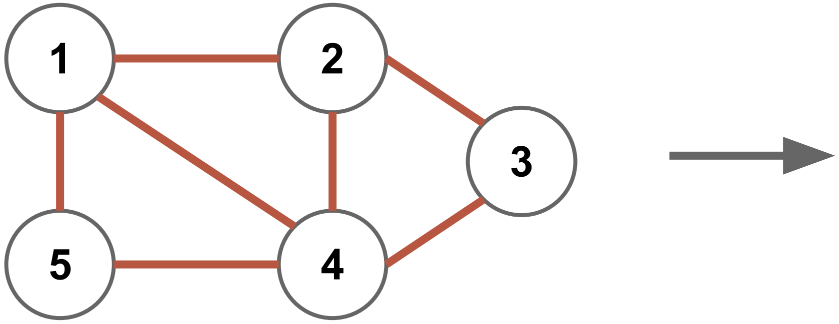


Vertex	Adjacent Vertices
A	B,D,E
B	A,D,C
C	B,D
D	A,B,C,E
E	A,D

Adjacency Matrix

The **Adjacency Matrix** for a graph ***M*** with ***n*** vertices is an ***n*** by ***n*** matrix, whose entries are 0 or 1, indicating if an edge is present.

- $\boldsymbol{M_{i,j}} = \mathbf{1}$ iff $\{\boldsymbol{i}, \boldsymbol{j}\}$ is an edge in the graph
- The vertices of ***M*** are labeled with integers in the range **1** to ***n***
- If ***M*** is undirected, $\boldsymbol{M_{i,j}} = \boldsymbol{M_{j,i}}$ because edge $\{\boldsymbol{i}, \boldsymbol{j})$ = edge $\{\boldsymbol{j}, \boldsymbol{i}\}$



$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 1 & 1 \\
2 & 1 & 0 & 1 & 1 & 0 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 1 & 1 & 1 & 0 & 1 \\
5 & 1 & 0 & 0 & 1 & 0 \\
\hline
\end{array}
$$

Handshake Theorem

Theorem

For any undirected graph, ***G* = (*V*, *E*)**: $\sum_{v \in V} deg(v) = 2|E|$

Note: *If* ***V*** *=* {$v_1, v_2, \ldots, v_n$} *then*

$$\sum_{v \in V} deg(v) = deg(v_1) + deg(v_2) + \ldots + deg(v_n)$$

Isomorphism of Graphs

The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are **isomorphic** if there exists a one-to-one and onto function f from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2 for all a and b in V_1

- Such a function is called an **isomorphism**
- Two simple graphs that are not isomorphic are called **nonisomorphic**

Walks, Trails, and Paths

A **walk** from ***s*** to ***t*** is a sequence of vertices $s, v_1, v_2, \ldots, v_k, t$ such that there is an edge between any two consecutive vertices in the list

- If the first and last vertices are different it's an **open walk**
- If they are the same, then it's a **closed walk**

A **trail** from ***s*** to ***t*** is an open walk that has no repeated edges

A **path** from ***s*** to ***t*** is a trail from ***s*** to ***t*** where all vertices are unique

- Note that a path is a special case of a trail

Circuits and Cycles

A **circuit** is a closed walk that has no repeated edges

A **cycle** is a circuit with length at least three such that there are no repeated vertices other than the first and the last.

- A closed path is a cycle

Euler Trail and Circuit

An **Euler circuit** in a graph ***G*** is a simple circuit containing every edge of ***G***

An **Euler trail** in a graph ***G*** is a trail that visits every edge of ***G*** exactly once

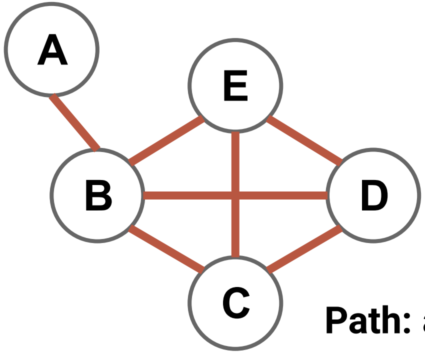
Theorem

A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has an even degree

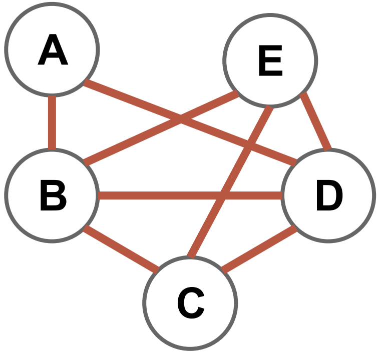
Hamiltonian Path and Cycle

A path in a graph ***G*** that passes through every vertex exactly once is called a **Hamiltonian path**.

A cycle in a graph ***G*** that passes through every vertex exactly once is called a **Hamiltonian cycle**.



Path: a,b,c,d,e



Cycle: a,b,c,e,d,a
Path: a,b,c,d,e

Connectivity

A node ***s*** is **connected** to ***t*** if there is a path from ***s*** to ***t***

A node ***s*** is **isolated** if there is no other vertex connected to ***s***

Connected Components

A set of vertices in a graph is **connected** if every pair of vertices in the set is connected

A graph is said to be **connected** if the entire set of vertices is connected

A graph that is not connected is **disconnected**

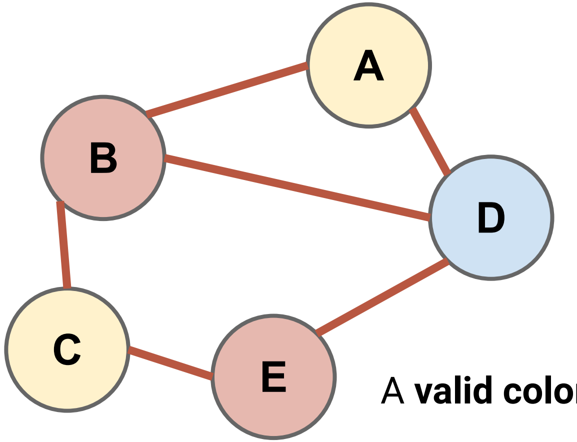
A **connected component** is a ***maximal*** set of connected vertices

Note: *A disconnected graph can be split into more than one connected component*

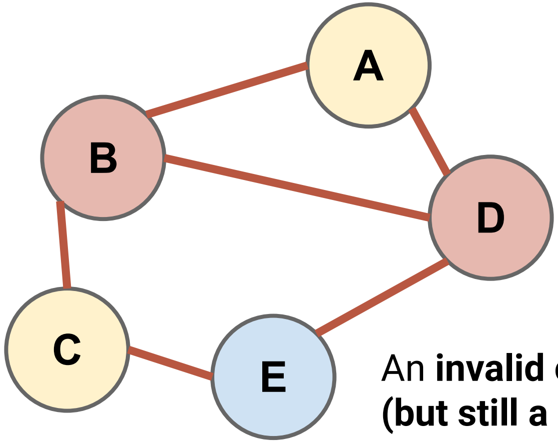
Graph Coloring

Let $G = (V, E)$ be an undirected graph and C be a finite set of colors. A **valid coloring** of G is a function $c: V \to C$ such that for $\forall \{x, y\} \in E$, $c(x) \neq c(y)$

- If $|C| = k$, then c is a **k-coloring** of G



A **valid coloring**



An **invalid coloring (but still a coloring)**

Graph Coloring

The **chromatic number** of a graph G (denoted $\chi(G)$) is the smallest k such that there is a valid **k-coloring** of G

Theorem

Let G be an undirected graph. Let $\Delta(G)$ be the maximum degree of any vertex in G. Then $\chi(G) \leq \Delta(G) + 1$

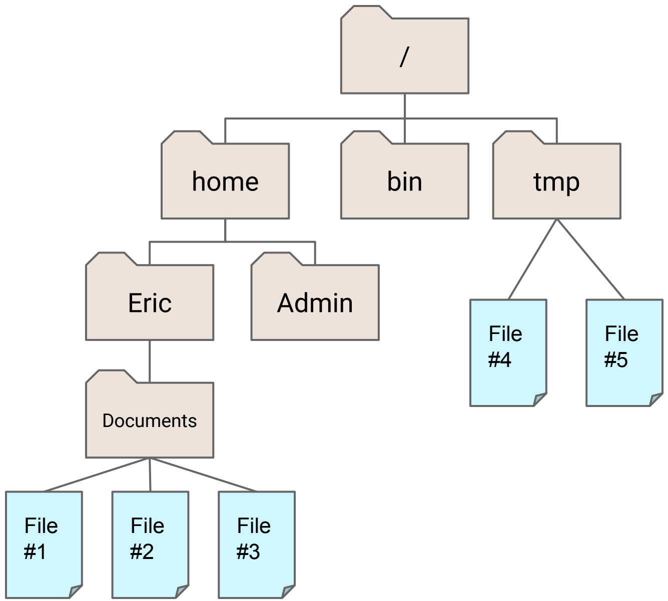
Trees

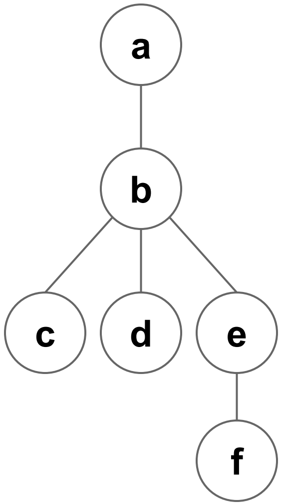
A **tree** is an undirected graph that is ***connected*** and has ***no cycles***

A **free tree** has no organization of vertices and edges

A **rooted tree** has a designated root at the top
(ie the "/" and "**a**" vertices →)

A free tree can be made rooted by choosing a root





Rooted Tree Terminology

Depth of a vertex: Distance from the root to that vertex

Height of a tree: maximum depth of any vertex

Parent of a vertex: the node above that vertex (towards the root)

- When ***u*** is the parent of ***v***, then ***v*** is a **child** of ***u***
- Vertices with the same parent are called **siblings**

Leaf: a vertex with no children

Rooted Tree Terminology

The <u>**ancestors**</u> of a vertex (other than the root) are the vertices in the path from the root to the vertex (including the root, but excluding the vertex). The root has no ancestors.

The <u>**descendants**</u> of a vertex, ***v***, are all vertices that have ***v*** as an ancestor

If ***a*** is a vertex in a tree, the <u>**subtree rooted at *a***</u> is the subgraph of the tree consisting of ***a***, all of its descendants, and all edges incident to those descendants

Finite State Machines [13.2-13.4]

Finite Automata - Finite State Machines (with no output)

A (deterministic) **finite automaton** ***M*** is a 5-tuple (Q, Σ, δ, q_0, F) where:

- The **set of states** Q is finite and non-empty
- The **input alphabet Σ** is finite and non-empty
- The **transition function** $\delta: Q \times \Sigma \rightarrow Q$
- The **starting state** $q_0 \in Q$
- The **set of final states** F

Finite Automata

Let this automata be $\boldsymbol{M_1} = (\boldsymbol{Q_1}, \boldsymbol{\Sigma_1}, \boldsymbol{\delta_1}, \boldsymbol{S_1}, \boldsymbol{F_1})$

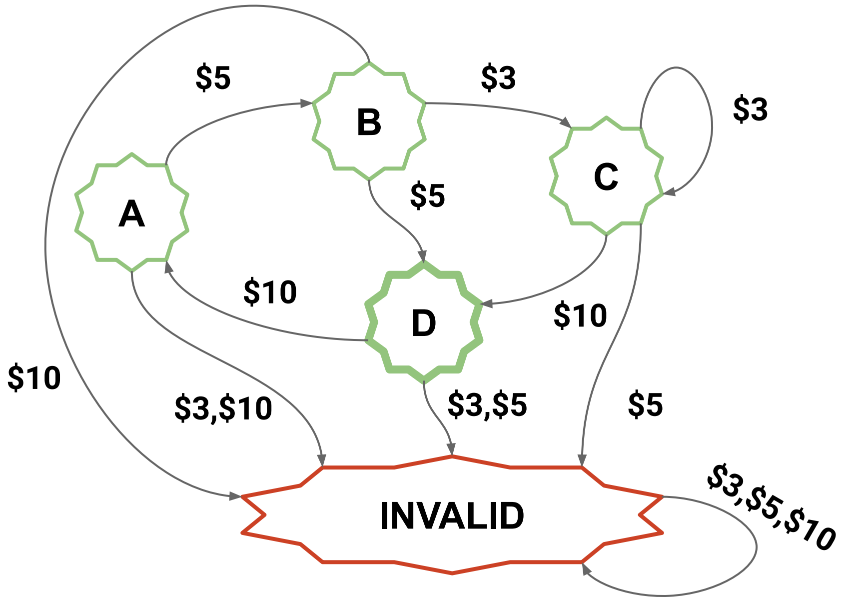
$\boldsymbol{Q_1} = \{ A, B, C, D, \textbf{INVALID} \}$

$\boldsymbol{\Sigma_1} = \{ \$3, \$5, \$10 \}$

$\boldsymbol{S_1} = A$

$\boldsymbol{F_1} = \{ D \}$

$\boldsymbol{\delta_1} = \{ ((A, \$3), \textbf{INVALID}), ((A, \$5), B), \ldots \}$



Finite Automata

A string x is **recognized** or **accepted** by the machine M if it takes the starting state q_0 to a final state.

The **language** that is ***recognized*** or ***accepted*** by the machine M, denoted by $L(M)$ is the set of strings recognized by M.

$$L(M) = \{ x \in \Sigma^* \mid \delta(q_0, x) \in F \}$$

Regular Expressions

A **regular expression**, or ***regex***, ***r*** over alphabet $\Sigma = \{ \mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_k} \}$ is:

- $\boldsymbol{r} = \mathbf{c}_i$ for some $\boldsymbol{i} \in \{ \mathbf{1}, \ldots, \boldsymbol{k} \}$
- $\boldsymbol{r} = \varnothing$
- $\boldsymbol{r} = \boldsymbol{\lambda}$

or, given regular expressions $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$, we can build up a new regex $\boldsymbol{r}$:

- $\boldsymbol{r} = (\boldsymbol{r_1} \mid \boldsymbol{r_2})$ ← $\boldsymbol{r_1}$ **OR** $\boldsymbol{r_2}$, also sometimes written $(\boldsymbol{r_1} \cup \boldsymbol{r_2})$
- $\boldsymbol{r} = (\boldsymbol{r_1}\boldsymbol{r_2})$ ← $\boldsymbol{r_1}$ ***concatenated with*** $\boldsymbol{r_2}$
- $\boldsymbol{r} = (\boldsymbol{r_1})^*$ ← kleene closure (0 or more repetitions)