COMP 182 All Notes

1 Predicated Logic and Proofs

1.1 Propositions

Declarative statement which is either True or False, where Propositional variables$(p, q, r, s, ...)$ represent propositions

If p is a proposition, $\neg p$ is the negation.

If p and q are two propositions, then

- $p \wedge q$ is the conjunction
- $p \vee q$ is their disjunction
- $p \oplus q$ is true if **only** one proposition is true
- $p \rightarrow q$ is true if p implies q
- $p \iff q$ is the implication both ways, where p implies q and q implies p

To establish the truth value of a proposition, we can use a truth table, testing every possibility.

A tautology is a compound proposition whose truth value is T under all truth assignments

A contradiction is a compound proposition whose truth value is F under all truth assignments

A compound proposition is satisfiable if it is not a contradiction(is able to evaluate to True), where a solution is a truth assignment that makes the compound proposition true.

Propositional Satisfiability Problem:

- Input: A compound proposition ϕ
- Output: "Yes" if ϕ is satisfiable, and "No" otherwise

1.2 Propositional Equivalences

Compound propositions p and q are logically equivalent if $p \iff q$ is a tautology

De Morgan's Laws:

- $\neg(p \wedge q) = \neg p \vee \neg q$
- $\neg(p \vee q) = \neg p \wedge \neg q$

Common Equivalences:

- $p \implies q \equiv \neg p \vee q$
- $\neg(\neg p) \equiv p$

- $p \vee q \equiv q \vee p$
- $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
- $p \vee (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
- $p \implies q \equiv \neg q \implies \neg p$
- $p \iff q \equiv (p \implies q) \wedge (q \implies p)$

We can prove equivalences by using other equivalences previously established

1.3 Predicates and Quantifiers

Predicate Logic - statements that involve variables
Can be denoted by prepositional function $P(x)$

- Evaluates to either T or F once a value has been assigned to variable x, in which case $P(x)$ becomes a proposition

Quantification can turn propositional functions to propositions
Predicate Calculus is the area of logic dealing with predicates and quantifiers
Universal Quantification - "$P(x)$ for all values of x in the domain of discourse" - $\forall x P(x)$
Existential Quantification - "$P(x)$ for some value of x in the domain of discourse" - $\exists x P(x)$
Quantifier Semantics

- $\forall x P(x)$ is true when $P(x)$ evaluates to T for all values of x in the domain, false when $P(x) = F$ for atleast one value of x in the domain
- $\exists x P(x)$ is true when $P(x)$ evaluates to T for atleast one value of X in the domain, false when $P(x)$ evaluates to F for all values of x in the domain

De Morgan's Laws for Quantifiers

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

1.4 Nested Quantifiers

$\forall x \forall y P(x, y)$ is True if $P(x, y) = T$ for every pair of (x, y), F if $P(x, y) = F$ for atleast one pair of (x, y)
$\exists x \exists y P(x, y)$ is true if $P(x, y)$ evaluates to T for atleast one pair of values of x and y and is F if $P(x, y) = F$ for every pair of (x, y)
$\forall x \exists y P(x, y) = T$ for every x there exists a y such that $P(x, y) = T$. F if there eixsts one x such that for all y $P(x, y) = F$
$\exists x \forall y P(x, y) = T$ if atleast one x such that for all y $P(x, y)$ evaluates to T. F if for every x there exists atleast one y such that $P(x, y) = F$

- $\forall x \forall y P(x, y) = \forall y \forall x P(x, y)$
- $\exists x \exists y P(x, y) = \exists y \exists x P(x, y)$
- $\forall x \exists y P(x, y) \neq \exists y \forall x P(x, y)$

1.5 Rules of Inference

Proofs are valid arguments that establish the truth of a mathematical statement

- Argument - a sequence of statements that end with a conclusion
- Valid - the conclusion must follow from the truth of the preceding statements or premises of the argument

An argument is a sequence of prepositions

- All but final proposition are called premises, final proposition is the conclusion
- An argument is valid if truth of all premises implies the conclusion is true

Argument form is the sequence of compound propositions involving propositional variables - it is valid if, no matter which particular propositions are substituted for propositional variables, the conclusion is true if the premises are all true

Inferring a true proposition from the truth of other propositions is the rules of inference

- Simplification: $p \wedge q \implies p$
- Modus Ponens: $p \wedge (p \implies q) \implies q$
- Hypothetical Syllogism: $(p \implies q) \wedge (q \implies r) \implies (p \implies r)$
- Modus Tollens: $\neg p \wedge (p \implies q) \implies \neg p$
- Disjunction Syllogism: $p \vee q \wedge \neg p \implies q$
- Conjunction: $p \; and \; q \implies p \wedge q$
- Resolution: $p \vee q \wedge \neg p \vee r \implies q \vee r$
- Addition: $p \implies p \vee q$

For statements with quantifiers, we want to get rid of quantifiers:

- Universal Instantiation: $\forall x P(x) \implies P(a)$ for arbitrary a
- Existential Instantiation: $\exists x P(x) \implies P(a)$ for some a
- Universal Generalization: $P(a)$ for arbitrary $a \implies \forall x P(x)$
- Existential Generalization: $P(a)$ for some $a \implies \exists x P(x)$

A theorem is a statement that can be show to be true with a proof
A proof is a valid argument that establishes the truth of a theorem
The statements used in a proof include axioms, which are assumed to be true

- Lemmas - less important theorems that are helpful in a proof
- Corollary - theorem that can be established directly from a theorem
- Conjecture - statement that is being proposed to be true, usually on basis of evidence

When proof is found for conjecture, conjecture becomes a theorem

1.6 Direct and Contrapositive Proof

Direct Proof – showing $p \implies q$
Contrapositive Proof – showing $\neg q \implies \neg p$
Exhaustive Proofs exhaust all possibilities when it is physically possible to do so while a proof by cases covers all cases(as opposed to instances) that exist in a theorem

1.7 Proof by Contradiction

Showing $p \implies q$ by proving that $\neg p \implies q$

1.8 Existence Proofs and Counterexamples

Many theorems are assertions that objects of a particular type exists
Constructive Proofs finds the element such that $P(x)$ is True
Counterexample Proofs find one counterexample showing that $\forall x P(x)$ is false.

2 Sets, Functions, and Relations

2.1 Sets

A set is an unordered collection of items where all elements are unique

- Roster Method of describing set – listing all elements($|\{1, 2, 3, 4\}$)
- Set Comprehension – describing set in terms of a property that its element satisfies($\{n : n > 5\}$

$a \in S$: a is an element of set S
$a \notin S$ a is not an element of set S
Special Sets

- Natural Numbers: $\mathbb{N} = \{0, 1, 2, ...\}$

- Integers: $\mathbb{Z} = \{... -2, -1, 0, 1, 2, ...\}$
- Positive Integers: $\mathbb{Z}^+ = \{1, 2, 3, ...\}$
- Set of Rational Numbers: $\mathbb{Q} = \{\frac{p}{q} : p, q \in \mathbb{Z} \land q \neq 0\}$
- Real Numbers $\mathbb{R}$
- Positive Real Numbers $\mathbb{R}^+$

The empty set is a set with no elements, denoted by $\emptyset$ or $\{\}$

Cardinality of a set, denoted by $|S|$ is the number of elements in S

Not all sets are finite, can have countably or uncountably infinite

A is a subset of B denoted by $A \subseteq B$ if $\forall x(x \in A \implies x \in B)$

The powerset, denoted by $\mathbb{P}(\mathbb{A})$ or 2^A is the set of all subsets of A

A is a proper subset of B, denoted by $A \subset B$ if $\forall x(x \in A \implies x \in B) \land \exists x(x \in A \land x \in B)$ – B must have atleast one more element than A

The rectangle denotes the universal set U, specifies sets we drawn as circles inside rectangle

2.2 Set Operations

Union: $A \cup B = \{x | x \in A \lor x \in B\}$ - elements in either A or B

Intersection: $A \cap B = \{x | x \in A \land x \in B\}$ - elements in both A and B, when $A \cap B = \emptyset$, A and B are disjoint

Difference: $A/B = A - B = \{x | x \in A \land \in B\}$ - set of all elements in A but not in B

Complement: $\bar{A} = U/A$, elements in universal set that are not in A

Cartesian Product: $A \times B = \{(x, y) | x \in A \land y \in B\}$ - set of all pairs (x, y) such that x is in A and y is in B - order matters for ordering of pair

2.3 Set Identities

Sets A and B are equal, denoted $A = B$ if $\forall x(x \in A \implies x \in B)$, equivalently, if $A \subseteq B \land B \subseteq A$

De Morgan's Laws:

- $A \barcap B = \bar{A} \cup \bar{B}$
- $A \barcup B = \bar{A} \cap \bar{B}$

Complement Law: $\bar{\bar{A}} = A$

Commutative Laws: $A \cup B = B \cup A$ and $A \cap B = B \cap A$

Associative Laws: $A \cup (B \cup C) = (A \cup B) \cup C$ and $A \cap (B \cap C) = (A \cap B) \cap C$

Distributive Laws: $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ and $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

2.4 Functions

Mapping elements of one set to another, let A and B be nonempty sets, a function f from A to B is the assignment of exactly one element of B to every element of A, $f(a) = b$

$f : A \to B$, A is the domain of f, B is the codomain

If $f(a) = b$, b is the image of a, and a is the preimage of b

The range of $f : A \to B$ is the set $\{b \in B : \exists a \in A, f(a) = b\}$, where range $\subseteq$ codomain

Two functions $f : A \to B$ and $g : C \to D$ are equal when:

- $A = C$
- $B = D$
- $\forall a \in A(f(a) = g(a))$

Let $f : A \implies \mathbb{R}$ and $g : A \implies \mathbb{R}$ be two functions. $f + g$ and fg defined, respectively by $(f + g)(a) = f(a) + g(a)$ and $fg(a) = f(a)g(a)$ are also functions from A to $\mathbb{R}$

2.5 Injections, Bijections, Surjections

A function $f : A \to B$ is an injection or one-to-one if $\forall a, b \in A(f(a) = b \implies a = b)$, where every element is mapped to a unique value in B. $|A| \leq |B|$

Surjection: A function $f : A \to B$ is a surjection, or onto if $\forall b \in B \exists a \in A, f(a) = b$, every element of B has a source and can be generated from A. $|A| \geq |B|$

A function $f : A \to B$ is a bijection if its one-to-one and onto, where $|A| = |B|$. There exists a bijection $f : A \to B \iff |A| = |B|$

2.6 Function Inverse and Composition

If $f : A \to B$ is a bijection, then f^{-1} from $B \to A$ is defined by $f^{-1}(b) = a) \implies (f(a) = b)$. In this case, we say that f is invertible. The inverse of a function must be a function

The composition of f and g where $g : A \to B$ and $f : B \to C$ is a function defined from A to C defined by $(f \circ g)(a) = f(g(a))$

2.7 Relations and Their Properties

Relations are the generalization of functions

Binary Relation - Let A and B be two sets, a binary relation from A to B is a subset of $A \times B$.

We write $(a, b) \in R$ or aRb to denote that a is related to b by relation R and $(a, b) \notin R$ to denote that a is not related to b by relation R

Let A be a set, a binary relation on A is a subset of $A \times A$ where the relation holds

A relation R on set A is reflexive if and only if $\forall a \in A, (a, a) \in R)$

A relation R on set A is symmetric if $\forall a, b((a, b) \in R \implies (b, a) \in R)$

A relation R on set A is antisymmetric if $\forall a, b((a,b) \in R \land (b,a) \in R)) \implies (a = b)$ A relation is transitive if $\forall a, b, c((a,b) \in R \land (b,c) \in R)) \implies (a,c) \in R)$ - a friend of my friend is my friend

Set operations apply when combining relations

- The Composition of R and S denoted by $S \circ S$ is $\{(a,c) | a \in A \land c \in C \land \exists b \in B((a,b) \in R \land (b,c) \in S)\}$

Relations can be defined on any number of sets, the relation on sets A, B, C, D is a subset of $A \times B \times C \times D$. Every element in such relation is a tuple (a, b, c, d). $\emptyset$ is transitive, stymmetric, and antisymmetric.

2.8 Representing Relations

Can be represented through a matrix. Relation R from $A \to B$ can be represented with an $m \times n$ matrix M where $m_{ij} = \begin{cases} 1 & (a_i, b_j) \in R \\ 0 & (a_i, b_j) \notin R \end{cases}$

Can also be represented by a digraph, where each node is an element in the set and a directed edge between nodes a and b exists if aRb.

2.9 Closure of Relations

Let R be a relation on set A and let P be some property of relations, if there is a relation S with property P containing R such that S is a subset of every relation with property P containing R, then S is called the closure of R with respect to P.

Let R be a relation on set A. The reflexive closure of R is $S = R \cup \{(a, a,) : a \in A\}$ - add the minimum number of elements to make reflexive

Symmetric Closure: $B = R \cup \{(a,b) : (b,a) \in R\}$

Transitive Closure: Let R be a relation on set A. The transitive closure of R is $R^* = \bigcup_{i=1}^{\infty} R^i$, where R is a relation on set A. We define $R^1 = R, R^2 = R \circ R, R^3 = R \circ R \circ R$

2.10 Equivalence Relations

A relation R on set A is called equivalence relation if it is reflexive, symmetric, and transitive

If a and b are related by an equivalence relation, we say they are equivalent such that a b

Let R be an equivalence relation on set A and let a be an element of A. The equivalence class of a, denoted $[a]$ is the set $\{b \in A : (a,b) \in R\}$

Sets $A_1, A_2, A_3, ...$ form a partition if

- $A_1, A_2, A_3, ...$ are nonempty sets
- The intersection of any two sets $A_1, A_2, A_3, ...$ is empty
- The union of all sets $A_1, A_2, ...$ equals A

Fundamental Theorem of Equivalence Relations - Let R be an equivalence relation on set A. The set of all equivalence classes of R forms a partition of A

3 Matrices and Graphs

3.1 Matrices

Matrices are a rectangular array of numbers, where an $m \times n$ matrix has m rows and n columns

Denoted by boldface capital letter - entry in row i and column j by italic small letter

Square Matrix - same number of rows and columns

Two matrices A and B are equal if they have the same number of rows and columns, say both are $m \times n$, and $\forall 1 \leq i \leq m, \forall 1 \leq j \leq n, a_{ij} = b_{ij}$

Let A and B be two $m \times n$ matrices, then $A + B = C$ and $A - B = D$, where $c_{ij} = a_{ij} + b_{ij}$ and $d_{ij} = a_{ij} - b_{ij}$, where the corresponding elements are added / subtracted

Let A be an $m \times k$ matrix and B be a $k \times n$ matrix. Then $C = AB$ is an $m \times n$ matrix where $c_i j = \sum_{1 \leq l \leq k} a_{il} b_{lj}$, i^{th} row in A times the j^{th} column in B, multiplying and summing corresponding values

The identity matrix of order n is the $n \times n$ matrix $I_n = [a_{ij}]$ where $a_{ii} = 1$ for all $1 \leq i \leq n$ and $a_{ij} = 0$ for $i \neq j$. 1's on diagonals, 0's elsewhere.

Let A be an $n \times n$ matrix, then $A^0 = I_n, A^1 = 1, A^2 = AA, ..., A^r = A^{r-1}A$

Let A be an $m \times n$ matrix. The transpose of A, denoted by A^t is the $n \times m$ matrix $[b_{ij}]$ where $b_{ij} = a_{ji}$ for every $1 \leq i \leq n$ and $1 \leq j \leq m$. A matrix is symmetric if $A^T = A$

A boolean matrix is where each entry is $0(False)$ or $1(True)$, can be defined on disjunction, conjunction, boolean product and power

3.2 Graphs and Graph Models

An Undirected Graph is a pair (V, E) where V is a nonempty set of nodes and $E \subseteq \{A : A \in \mathbb{P}(\mathbb{V}) \wedge |A| = 2\}$, every edge in E is a set with two elements.

A directed graph(digraph) is a pair (V, E) where V is a nonempty set of nodes and $E \subseteq (V \times V) - \{(u, u) : u \in V\}$ is a set of directed edges

3.3 Graph Terminology

Let $g = (V, E)$ be a graph. Two nodes $u, v \in V$ are adjacent or neighbors if $\{u, v\} \in E$. The degree of node $u \in V$, denoted by $deg(u)$ is the number of $u's$ neighbors, where $deg(u) = |\{v \in V : \{u, v\} \in E\}|$

Let $g = (V, E)$ be a directed graph. For edge $(u, v) \in E$ we call u the tail of the edge and v the head of the edge.

The in-degree of node $u \in V$ denoted by $indeg(u)$ is the number of edges whose head is node u

The out-degree of node $u \in V$, denoted by $outdeg(u)$ is the number of edges whose tail is the node u

Let $g = (V, E)$ be a graph. We say that $g' = (V', E')$ is a subgraph of g if $V' \subseteq V$ and $E' \subseteq E$

Let $g = (V, E)$ be a graph, we say that $g' = (V', E')$ is an induced subgraph of g if $V' \subseteq V$ and $E' = \{\{u, v\} \in E : u, v \in V'\}$

3.4 Special Types of Graphs

A complete graph of n nodes(denoted by K_n) is a graph that contains en edge between every two nodes

A cycle on $n \geq 3$ nodes (denoted C_n) is a graph with nodes $v_1, v_2, ...v_n$ and n edges $\{v_1, v_2\}, \{v_2, v_3\}, ...\{v_{n-1}, v_n\}, \{v_n, v_1\}$

A graph g is bipartite if V can be bipartitioned into two sets V_1 and V_2 such that every edge in E connects two nodes, one of which is in V_1 and the other is in V_2

Theorem: A graph $g = (V, E)$ is bipartite if and only if there exists a function $f : V \to \{1, 2\}$ such that $\forall\{u, v\} \in E, f(u) \neq f(v)$ - can color nodes of graph into 1 and 2 such that non adjacent nodes have the same color

Complete Biparitite Graph($K_{m,n}$) is a graph $g_2 = (V, E)$ whose set of nodes can be bipartitioned into two sets V_1, V_2 such that $|V_1| = m$ and $|V_2| = n$ and there is an edge between every node in V_1 and every V_2.

A graph $g = (V, E)$ is a forest if g is cyclic, and it is a tree if it is connected and acyclic.

3.5 Graph Connectivity

Let k be a nonnegative integer. A path of length k between nodes v_0 and v_k in graph $g = (V, E)$ is a sequence of k edges $e_1, e_2, ...e_k \in E$ such that $e_1 = \{v_0, v_1\}, e_2 = \{v_1, v_2\}, ..., e_k = \{v_{k-1}, v_k\}$ and $v_1, v_2, ...v_{k-1} \in V$. Denoted as $(v_0, v_1, ...v_k)$

In Digraphs, let $k \leq 0 \in \mathbb{Z}$. The path length k from node v_0 to v_k in digraph $g = (V, E)$ is a sequence of k edges $e_1, e_2, ..., e_k \in E$ such that $e_1 = (v_0, v_1), e_2 = (v_1, v_2), ...e_k = (v_{k-1}, v_k)$ and $v_1, v_2, ...v_{k-1} \in V$.

We denote such path by the sequence $v_0, v_1, ..., v_k$ where the path follows the direction of edges

A path is simple if it does not contain the same node more than once

A cycle is a simple path beginning and ending on the same node

A path the begins and ends at the same node is a circuit - does not have to be simple

A graph is acyclic if it does not contain a cycle

A graph is connected if there is a path between every pair of nodes in the graph

A connected component(CC) of a graph G is a maximal subset of nodes C such that there is a path between any pair of nodes in C(Path only uses nodes in C). Can get to any node from any node.

Strongly-Connected - edge connecting every other

3.6 Graph Isomorphism

Graph $g_1 = (V_1, E_1)$ is isomorphic to graph $g_2 = (V_2, E_2)$ if there is a bijection $f : V_1 \to V_2$ such that $\{u, v\} \in E \iff \{f(u), f(v)\} \in E_2$, where the function f is an isomorphism.

If there is an edge in a between $f(x)$ and $f(y)$, then there must be an edge between x and y

Graph $g_1 = (V_1, E_1)$ is subgraph isomorphic to graph $g_2 = (V_2, E_2)$ if g_1 is isomorphic to a usbgraph of g_2

Set A = $\{all\ graphs\ on\ n\ nodes\}$

$R = \{(g_1, g_2) : g_1$ is isomorphic to $g_2\}$

Every graph is isomorphic to itself $(g_1, g_1) \in R$

3.7 Representing Graphs

An adjacency list specifies the neighbors of each node - like a dictionary where the keys are the nodes and the values is a list of neighbors

If the nodes of a graph $g = (V, E)$ are $v_1, v_2, ...v_n$, then there is an adjacency matrix of the graph, denoted by A_g, is an $n \times n$ binary matrix such that

$A_g[i, j] = \begin{cases} 0 & \{v_i, v_j\} \notin E \\ 1 & \{v_i, v_j\} \in E \end{cases}$ Input Size: If the graph is represented by its adjacency list, then the input size is often given by m(the number of edges) and n(the number of nodes). If the graph is represented by its adjancency matrix, then the input size is given by n alone.

The choice of input size depends on algorithms and assumptions on input, but in general

- Sparse Graph - Adjacency List
- Dense Graph - Adjacency Matrix

4 Sequences and Summations

4.1 Sequences and Summations

A sequence is a function from a subset of the set of integers to a set S.

Notation a_n is used to denote the n^{th} term, where the sequence itself is defined as $\{a_n\}$.

A geometric progression is a sequence of the form $a, ar, ar^2, ..., ar^n$, where the initial term a and the common ratio r are real numbers.

$\frac{a_n}{a_{n-1}} = \frac{ar^n}{ar^{n-1}} = r$

An arithmetic progression is a sequence of form $a, a + d, a + 2d, ...a + nd$. The initial term a and common difference d are real numbers

Recurrence Relation for sequence $\{a_n\}$ is an equation that expresses a_n in terms of previous terms of sequence, namely $a_9, a_1, ..., a_{n-1}$ for all integers n with $n \geq n_0$ where n_0 is nonnegative.

A sequence is a solution of recurrence relation if its terms satisfy recurrence relation

A summation is the sum of terms of sequences
Three Types of sums of interest

- Infinite Sum: $\sum_{n=0}^{\infty} a_n$
- Finite Sum from i to j: $\sum_{n=1}^{n=j} a_n$
- Summation of n satisfying a constraint: $\sum_{n \in S} a_n$

A sum of sequence yields the series S_n as follows

- $S_n = \sum_{i=1}^{n} a_i$ for $n = 1, 2, 3, ...$

And if a_n is defined for $n = 0$:

- $S_n = \sum_{i=0}^{n} a_i$ for $n = 0, 1, 2, ...$

A telescoping sum can be defined as $\sum_{j=1}^{n}(a_j - a_{j-1} = (a_1 - a_0) + (a_2 - a_1) + +... + (a_{n-1} - a_{n-2}) + (a_n - a_{n-1}) = a_n - a_0$
$\sum_{j=0}^{n} a + dj = \sum_{j=0}^{n} a + \sum_{j=0}^{n} d_j$
$\sum_{i=1}^{n} j = \frac{n^2+n}{2}$
Can perturb sum by creating another summation and considering the differences – wanting the difference to cancel out the majority of terms
$S_n = \sum_{j=0}^{n} ar^j = \frac{a(r^{n+1}-1)}{r-1}$
$\sum_{j=0}^{\infty} ar^j = \frac{a}{1-r}$ $|r| < 1$

5 Mathematical Induction

5.1 Mathematical Induction

$[P(1) \wedge \forall k P(k) \implies P(k+1)] \implies \forall n P(n)$
Uses fact that the set of positive integers is sell ordered

- $P(1)$ is true
- $P(k) \implies P(k+1)$

To prove $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function, we complete 2 steps:

- Basis Step: Verify that $P(1)$ is True
- Inductive Step: Show that $P(k) \implies P(k+1)$ is True

5.2 Strong Mathematical Induction

To prove $P(n)$ is true for all positive integers n, where $P(n)$ is a propositional function, two steps:

- Verify that $P(1)$ is true
- Show that $P(1) \wedge P(2) \wedge P(3)... \wedge P(k) \implies P(k+1)$ is true

5.3 Recursive Definition

Define function / structure by invoking some structure on previously defined elements.
Recursively defined functions: $f : \mathbb{N} \to \mathbb{N}, \mathbb{N} = \{0, 1, 2, ...\}$

- Basis Step / Base Case - specify what $f(0)$ is
- Recursive Step: Specify rule / set of rules to define $f(n)$ based on a combination of the elements, $f(0), f(1), ...f(n-1)$

5.4 Structural Induction

Suppose we want to prove: $P(w) \forall w \in \Sigma^*$.

- Basis Step: Show that $p(\epsilon)$ is True
- Inductive / Recursive Step: Show that if $P(w)$ is true and $\sigma \in \Sigma, w \in \Sigma^*$ then $P(w\sigma)$ is true

6 Algorithms and Algorithm Complexity

6.1 Algorithms

Finite sequence of precise instructions for performing a computation

- Must have a proper name
- Must have input / output - both take values from specified sets
- Steps of algorithm should be unambiguous
- Should produce output in a finite number of steps and every step must take a finite amount of time to execute
- Should produce correct output values for each set of input values according to the computation it is supposed to perform

6.2 Growth of Functions

Differences in running times on small inputs is not what distinguishes an efficient algorithm from inefficient ones.

When analyzing complexity of algorithm, pay attention to order of growth of number of steps that algorithm takes as input size increases

Beyond running time, interested not in specific values of function but in its behavior

Do not distinguish between functions that are "related by a multiplicative constant"

$f(n) = n, g(n) = 5n$ - growth at some order, do not distinguish between each other

Assume that every function is asymptotically nonnegative, there exists a constant k such that $f(x) \geq 0$ for every $x \geq k$

Big-O: Asymptotic $\leq$

- Let f and g be 2 real valued functions. We say that $f(x) = O(g(x))$ if there exists positive constants c and k such that $f(x) \leq Cg(x)$ for all $x \geq k$
- To prove that $f(x) = O(g(x))$, we have to find positive constants C and k such that $f(x) \leq Cg(x)$ for all $x \leq k$
- TO prove that $f(x) \neq O(g(x))$, one has to show that for any positive constant C and for any positive constant k, there exists an $x \geq k$ such that $f(x) > Cg(x)$

Big Omega: Asymptotic $\geq$

- Let f and g be two real-valued functions. We say that $f(x) = \Omega(g(x))$ if there exists positive constants C and k such that $f(a) \geq Cg(a)$ for all $a \geq k$ - lower bound
- Atleast a certain running time

Big Theta: Asymptotic $=$

- Let f and g be two real-valued functions. We say that $f(x) = \theta(g(x))$ if there exist positive constants C_1, C_2, k such that $C_1 g(x) \leq f(x) \leq C_2 g(x)$ for all $x \geq k$

6.3 Asymptotic Notation: Helpful Results

Theorem: $f(x) = O(g(x)) \iff g(x) = \Omega f(a)$

Theorem: $f(x) = \theta g(x) \iff (f(x) = O(g(x)) \wedge f(x) = \Omega(g(x))$

Theorem: Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$ where $a_0, a_1, ...a_n \in \mathbb{R}$. Then $f(x) = O(x^n)$

Theorem: Assume that $f_1(x) = O(g(x))$ and $f_2(x) = O(h(x))$. Then $(f_1 + f_2)(x) = O(max(g(x), h(x)))$

Theorem: Assume that $f_1(x) = O(g(x))$ and $f_2(x) = O(h(x))$. Then $f_1 \cdot f_2)(x) = O(g(x) \cdot h(x))$

6.4 Asymptotic Notation: Definitions Using Limits

$f(x) = O(g(x)) \iff \lim_{x\to\infty} \frac{f(x)}{g(x)} < \infty$

$f(x) = \Omega(g(x)) \iff \lim_{x\to\infty} \frac{f(x)}{g(x)} > 0$

$f(x) = \theta(g(x)) \iff \lim_{x\to\infty} \frac{f(x)}{g(x)} = d$ for some constant $0 < d < \infty$.

Little-o: Asymptotic ¡: $f(x) = o(g(x)) \iff \lim_{x\to\infty} \frac{f(x)}{g(x)} = 0$

Little-omega: Asymptotic ¿: $f(x) = \omega(g(x)) \iff \lim_{x\to\infty} \frac{f(x)}{g(x)} = \infty$

6.5 Equivalences

$f(x) = O(g(x))$ is notation abuse, O is not a function.
$O(g) = \{h \exists c, k, \forall x \geq k, h(x) \leq cg(x)\}$, $f \in O(g(x))$
Asymptotic Equality(θ) is an equivalence relation

- Reflexive: $f = \theta(f(x))$
- Symmetric: $f = \theta(g) \implies g = \theta(f)$
- Transitive: $f = \theta(g) \wedge g = \theta(h) \implies f = \theta(h)$

6.6 Big-O of Important Functions

$n \leq 2^n, n \geq 1 : n = O(2^n); n = o(2^n)$
$log_2 n \leq n : logn = O(n)$
$n! = O(n^n)$
$logn! = O(nlogn)$
if $a < b, a^n = O(b^n)$
$logn = O(n^\epsilon) \forall \epsilon > 0$ - bases for logs do not matter by change of base rule

6.7 Asymptotic of General Functions

How do we modify definitions that are not necessarily asymptotically nonnegative?
$f, g : \mathbb{R} \to \mathbb{R}, f = O(g)$ if $\exists c, k$ such that $|f(x)| \leq c|g(x)| \forall x \geq k$

6.8 Complexity of Algorithms

Choosing most efficient algorithm among correct ones - might choose slightly slower algorithm that is easier to implement

- Time Efficiency: How fast an algorithm runs
- Space Efficiency: How much extra space the algorithm takes

Often a trade off between the two

All a function of input size - investigate an algorithms complexity in terms of some parameter n which indicates input size

- For sorting lists - n is the number of elements in the list
- For cardinality - n is the number of elements in the set

More than one parameter might be needed to capture input size. For an algorithm that computes intersection of A and B for sizes m and n, the input size is captured by m and r

Units for measuring Run Time

- Focus on basic operations that algorithm performs - compute the number of times the basic operation is executed
- Do not something dependent on hardware
- Complexity Analysis Framework ignores multiplicative constants and concentrates on order of growth of the number of steps to within a constant multiplies for largest-size inputs

The Complexity of a Problem is not a statement about a specific algorithm, while the complexity of an algorithm pertains to that specific algorithm

For algorithm and input size n :

- Worst Case: Largest number of operations that the algorithm takes on an input of size n, gives upper-bound, most commonly used. Polynomial Complexity: $O(n^a)$ or better is good, while Exponential Complexity:$O(a^n)$ is bad
- Best Case: Smallest number of operations that the algorithm takes on input of size n, usually uninformative, gives lower bound
- Average Case: Expected number of operations that algorithm takes on size n, most informative, requires making assumptions about input and using tools from probability

Caes are defined for given input size n, cannot change size to get worst or best complexities

"Bad Algorithms" and Problems

- If there exists a polynomial time worst case algorithm for a given problem, we say the problem is tractable(class P), otherwise it is intractable

For problems for which solutions can be verified in polynomial time belong to class NP

NP-Complete - problems that if one can be solved by a polynomial time worst case algorithm, then all problems in class NP can be solved by polynomial time worst-case algorithms

6.9 BFS

Graph exploration algorithm, which traverses graph to find characteristics.
When graph is large, exhaustive analysis is infeasible - alternative is to explore a region(or regions) of the graph, repeat results based on explored parts
Exploration can be done:

- Deterministically: using BFS or DFS
- Nondeterministically: Random Walks

BFS : Starting from pre-specific source mode, visit the node and its neighbors; then for each neighbor, visit its neighbors and so until no more nodes can be explored

6.10 Divide and Conquer Algorithms and Recurrence Relations

Divide and Conqur: A problem's instance of size n is divided into smaller instances, ideally of about same size $\frac{n}{b}$
Some(say a of the b subproblems) of the smaller instances are solved(typically recursively)
Solutions to the solves subproblems are combined to get solution to original instance
Master Theorem: Let f be an increasing function that satisfies the recurrence relation $f(n) = af(\frac{n}{b}) + cn^d$, where $n = b^k$, where k is a positive integer, $a \geq 1, b$ is an integer greater than 1, and c and d are real numbers with c positive and d nonnegative. Then

$$\begin{cases} O(n^d) & a < b^d \\ O(n^d log n) & a = b^d \\ O(n^{log_b a}) & a > b^d \end{cases}$$

7 Weighted Graphs and Greedy Algorithms

7.1 Optimization Problems and Greedy Algorithms

Optimization Problem: Finding the minimum / maximum of a function
Greedy Algorithm: Builds the solution piece by piece, choose the next piece to be "locally" optimal among available pieces - must maintain a feasible solution
The end of the chapters deal with examples of greedy algorithms: Compatible Intervals, MST and Prims, Shortest Path, and Dijkstra's - better covered in videos

8 Counting

8.1 Pigeonhole Principle

If you have n pigeonholes and $n + 1$ pigeons, atleast one pigeonhole has to have multiple pigeons.
$f : A \to B$, $|B| < |A| \to \exists a_1, a_2 \in A, a_1 \neq a_2 \wedge f(a_1) = f(a_2)$
Atleast two pigeons must be mapped to a pigeonhole

8.2 Product and Division Rules

When counting numbers of elements of a set A_2, if A can be written as the cartesian product of n sets, say $A = A_1 \times A_2 \times ... \times A_x$, then $|A| = |A_1||A_2||A_3|...|A_n|$
If there exists an onto function $f : A \to B$ that such $\forall b \in B|f^{-1}(b)| = d$, then $|B| = |A|/d$

8.3 Sum and Subtraction Rules

Sum Rule: When counting the number of elements of a set A, if A can be written as the union of pairwise disjoint sets, say $A = A_1 \cup A_2 \cup ... \cup A_n$, then $|A| = |A_1| + |A_2| + ... + |A_n|$
Subtraction Rule: When counting the number of elements of a set A, if A can be written as the union of two sets, say $A = A_1 \cup A_2$, but the two sets are not disjoint, then $|A| = |A_1| + |A_2| - |A_1 \cup A_2|$

8.4 Permutations and Combinations without Repetition

Let n be a positive integer and k be an integer with $0 \leq k \leq n$. Denoted by $P(n, k)$, the number of all $k - permutations$ of a set with n distinct elements. Then, $P(n, k) = \frac{n!}{(n-k)!}$
Let n be positive and k be an integer with $O \leq k \leq n$. Denoted by $C(n, k)$, the number of all k combinations of a set with n distinct elements is $C(n, k) = \frac{n!}{k!(n-k)!}$
$\frac{P(n,k)}{k!} = C(n, k)$

8.5 Permutations and Combinations with Repetition

The number of k permutations of a set of n objects with repetition allowed is n^k.
The number of $k-$combinations of a set of n objects with repetition allowed is $C(n+k-1, k)$

8.6 Inclusion-Exclusion Principle

Let $A_1, A_2, ...A_n$ be finite sets
$|A_1 \cup A_2 \cup ... \cup A_n| = \sum_{i=1}^{n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| + sum_{1 \leq i < j < k \leq n}^{n} |A_i \cap A_j \cap A_k| - ... + ... - ... + ...(-1)^{n+1}|A_1 \cap A_2 \cap ...A_n|$

8.7 Binomial Coefficients and Combinatorial Proofs

Binomial Theorem: Let x and y be variables and let $n \leq 0 \in \mathbb{Z}$. Then
$(x + y)^n = \sum_{i=0}^{n} \binom{n}{i} x^{n-i} y^i$
Combinatorial Proof: Uses counting arguments to prove that both sides of the identity count the same objects but in different ways or is based on showing that there is a bijection between sets of objects counted by the two sets of the identity

Pascal's Identity:
$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$
Vandermorde's Identity:

$\binom{m+n}{k} = \sum_{l=0}^{k} \binom{m}{k-l}\binom{n}{l}$

9 Dynamic Programming

9.1 Modeling with Recurrence Relations

Modeling various real like things with recurrence relations. For example, we can describe the number of moves needed to solve the Tower of Hanoi, for n disks as $H_n = 2H_{n-1} + 1$

9.2 Linear Homogeneous Recurrence Relations

Recurrence Relation where $A_n = c_1a_{n-1} + c_2a_{n-2} + ... + c_ka_{n-k}$, $c_1, c_2, ..., c_k \in \mathbb{R}$ and $c_k \neq 0$
Let c_1 and c_2 be two real numbers. Suppose that $r^2 - c_1r - c_2 = 0$ has two distinct roots r_1 and r_2. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2} \iff a_n = a_1r_1^n + a_2r_2^n$ for $n = 0, 1, 2, ...$ where a_1, a_2 are constants

9.3 Binomial Coefficients

We use a method called Dynamic Programming as a replacement for recursion to optimize solutions which involve calculating values repeatedly by saving this values.
For the Binomial Coefficient, where we want to find $\binom{n}{k}$. We can describe the binomial as $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
DPNcK(n,k)

- for $i = 0$ to n
  - $A[i, 0] = 0$
  - for $i = 1$ to n
    - for $j = 1$ to i
      - $A[i, j] = A[i - 1, j] + A[i - 1, j - 1]$
- return $A[n, k]$

Similar problems include Weighted Compatible Intervals and Longest Common Subsequence, which are better covered in the videos.