Introduction

Email: liming@cad.zju.edu.cn

QQ group: 368350056

Assessment:

- Homework Due on Monday 15%
- Quiz 1–3 times 10–15%
- Midterm 20–25%
- Final 50%

Logic and Proofs

Propositional logic

Proposition: A *declarative* sentence that is *either true or false*, but not both.

Statements that can not be decided is not a proposition.

Propositional/Statement variable: p, q, r, s

Truth value: T, F

Propositional Calculus/Logic: The area of logic that deals with propositions.

Compound proposition: Proposition formed from existing propositions using operators.

Operator:

- Not

  ¬p

  e.g. It is not the case that ⋯

- Conjunction/And

  p ∧ q

  e.g. ⋯ and/but ⋯

- Disjunction/Or

  p ∨ q

  e.g. ⋯ or ⋯

- Exclusive or

  p ⊕ q

- Conditional Statement / Implication

  p → q

  - if p, then q
  - *p only if q*
  - p implies q
  - p is a sufficient condition for q
  - q is a necessary condition for p
  - q *follows* from / if / when / whenever p
  - *q unless ¬p*

  False only when p is true, but q is false; when p is false, p → q is defined to be true.

- Converse

  $p \rightarrow q$ to $q \rightarrow p$

- Inverse

  $p \rightarrow q$ to $\neg p \rightarrow \neg q$

- Contrapositive

  Converse and inverse

  $p \rightarrow q$ to $\neg q \rightarrow \neg p$

- Biconditional statement

  $p \leftrightarrow q$

  - p if and only if q
  - p is necessary and sufficient for q
  - if p then q, and conversely
  - p iff q

  $p \leftrightarrow q == (p \rightarrow q) \wedge (q \rightarrow p)$, only true when p and q have the same value

Operator precedence: (), $\neg$, $\wedge$ $\vee$, $\rightarrow$ $\leftrightarrow$

Bit: 1 for T, 0 for F.

Bit string: a sequence of zero or more bits.

Bit operation: Bitwise AND, Bitwise OR, Bitwise XOR

Applications of propositional logic

Translating English sentences: to logical proposition.

Consistent system specification: no conflicting requirements; a set of value to satisfy all the statements translated; *does not necessarily has a real usage*.

Propositional equivalences

Classification of compound proposition:

- Tautology

  Always true

- Contradiction

  Always false

- Contingency

  Neither a tautology nor a contradiction

Logical equivalence:

Compound propositions that have the same truth values in all possible cases

$p \leftrightarrow q$ is a tautology

$p \equiv q$, $p \Leftrightarrow q$

Logical Equivalences:

Equivalence	Name
$p \wedge T \equiv p$	Identity laws
$p \vee F \equiv p$	
$p \vee T \equiv T$	

$p \wedge F \equiv F$	Domination laws
$p \vee p \equiv p$	Idempotent laws
$p \wedge p \equiv p$	
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$	Commutative laws
$p \wedge q \equiv q \wedge p$	
$(p \vee q) \vee r \equiv p \vee (q \vee r)$	Associative laws (*Within the same operator*)
$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	Distributive laws
$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	
$\neg(p \wedge q) \equiv \neg p \vee \neg q$	De Morgan’s laws
$\neg(p \vee q) \equiv \neg p \wedge \neg q$	
$p \vee (p \wedge q) \equiv p$	Absorption laws
$p \wedge (p \vee q) \equiv p$	
$p \vee \neg p \equiv T$	Negation laws
$p \wedge \neg p \equiv F$	
$p \rightarrow q \equiv \neg p \vee q$	Implication laws
$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$	

Mine: $\neg(p \to q) \equiv p \wedge \neg q$ *Mine:* $p \vee (\neg p \wedge q) \equiv p \vee q$ See Section 3.1 No.30

Prove Logical Equivalence:

Truth table or developing a series of logical equivalences.

Prove Not Logically Equivalent:

(Simplify first) and find a counterexample.

Propositional Satisfiability:

An assignment of truth values that makes it true.

Truth table, or whether its negation is a tautology. (or say whether itself is a contradiction?)

Predicates and Qualifiers

Predicate: Statements involving variables, the variables are called subjects, the other is called a predicate.

The statement of P(x) is also called the value of Propositional function P at x.

Precondition: Conditions for valid input

Postcondition: Conditions for correct output

Qualification: Express the extent to which a predicate is true over a range of elements.

((Domain / universe) of discourse) / domain: A predicate is true for a variable in a particular domain.

Predicate calculus: The area of logic that deals with predicates and

quantifiers

Universal qualification: For every element.

$\forall x\ P(x)$: For all / every x P (x)”

An element for which P(x) is false is called a counterexample of $\forall x\ P(x)$.

Existential qualification: For one or more element.

$\exists x\ P(x)$: There exists an element x in the domain such that P(x).

Uniqueness qualification: $\exists !x\ P(x)$ or $\exists \_1\ x\ P(x)$

There exists a unique x such that P (x) or There is one and only one x such that P (x)

Abbreviated qualifier notation: Use condition for domain.

Or use $\forall x\ P(x) \rightarrow Q(x)$ for using P(x) as condition for domain.

Quantifiers ($\forall$ and $\exists$) have higher precedence than all logical operators from propositional calculus. i.e. They absorb less.

Occurrence of variable is bound: Quantifier is used on the variable.

All the variables that occur in a propositional function must be bound or set equal to a particular value to turn it into a proposition.

Scope of quantifier: The part of a logical expression to which a quantifier is applied.

The same letter is often used to represent variables bound by different quantifiers with scopes that do not overlap.

Statements involving predicates and quantifiers are logically equivalent: If and only if they have the same truth value *no matter* which *predicates* are substituted into these statements and which *domain* of discourse is used for the variables in these propositional functions.

$S \equiv T$

$\forall x\,(P(x) \wedge Q(x)) \equiv \forall x\,P(x) \wedge \forall x\,Q(x)$

$\exists x\,(P(x) \vee Q(x)) \equiv \exists x\,P(x) \vee \exists x\,Q(x)$

De Morgan’s laws for quantifiers:

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

$\neg \exists x Q(x) \equiv \forall x\,\neg Q(x)$

Translating from English into Logical Expression: ⋯

Using Quantifiers in System Specifications: ⋯

Nested Qualifiers

Nested quantifiers: One quantifier is within the scope of another.

Understanding Statements Involving Nested Quantifiers: ⋯

The order of the quantifiers is important, unless all the quantifiers are universal quantifiers or all are existential quantifiers.

Statement	Condition for true
$\forall x \forall y\,P(x, y)$	
$\forall y \forall x\,P(x, y)$	P (x, y) is true for every pair x, y.
$\forall x \exists y\,P(x, y)$	For every x there is a y for which P (x, y) is true.
$\exists x \forall y\,P(x, y)$	There is an x for which P (x, y) is true for every y.
$\exists x \exists y\,P(x, y)$	
$\exists y \exists xP\,(x, y)$	There is a pair x, y for which P (x, y) is true.

Translating: ⋯

Negation: Recursively⋯

Normal forms

(Disjunctive / conjunctive) clause: Disjunctions / Conjunctions with literals (optionally negated) as its disjuncts / conjuncts.

Disjunctive / Conjunctive normal form (DNF / CNF): A disjunction / conjunction with conjunctive / disjunctive clauses as its disjuncts / conjuncts.

Full disjunctive / conjuctive normal form: Each of its variables appears exactly once in every clause. Obtained by adding $\wedge (\neg P(x) \vee P(x))$ to its conjuction disjuncts / $\vee (\neg P(x) \wedge P(x))$ to its disjunction conjuncts.

Prenex normal form: Qualifier⋯Predicate_without_quaulifier

Inference

Argument: a sequence of propositions.

Premise: All but the final proposition in the argument.

Conclusion: The final proposition in the argument.

Valid argument: An argument is valid if the truth of all its premises implies that the conclusion is true.

Fallacy: Common forms of incorrect reasoning which lead to invalid arguments.

Argument form: A sequence of compound propositions involving propositional variables.

Valid argument form: An argument form is valid no matter which

particular propositions are substituted for the propositional variables in its premises, the conclusion is true if the premises are all true.

The key to showing that an argument in propositional logic is valid is to show that its argument form is valid.

Using truth table to show that an argument form is valid is tedious.

Modus ponens (Mode that affirms) / the law of detachment: $(p \land (p \to q)) \to q$

Argument can be valid, but if any of its premise is false, its conclusion is false.

Tautology	Name
$(p \land (p \to q)) \to q$	Modus ponens
$(\neg q \land (p \to q)) \to \neg p$	Modus tollens
$((p \to q) \land (q \to r)) \to (p \to r)$	Hypothetical syllogism
$((p \lor q) \land \neg p) \to q$	Disjunctive syllogism
$p \to (p \lor q)$	Addition
$(p \land q) \to p$	Simplification
$((p) \land (q)) \to (p \land q)$	Conjunction
$((p \lor q) \land (\neg p \lor r)) \to (q \lor r)$	Resolution

$p \land q \to (r \to s) \equiv p \land q \land r \to s$, r is additional premise.

Resolution is commonly used: $((p \lor q) \land (\neg p \lor r)) \to (q \lor r)$

Also rewrite premises into separate clauses can help.

Fallacy

- Affirming the conclusion

  $((p \to q) \land q) \to p$ is not a tautology.

- Denying the hypothesis

  $((p \to q) \land \neg p) \to \neg q$ is not tautology.

- Begging the question (Circular reasoning)

  one or more steps of a proof are based on the truth of the statement being proved.

Rules of inference for quantified statements

Rule of Inference	Name
$\forall x\, P(x) \therefore P(c)$	Universal instantiation
$P(c)$ for an arbitrary $c \therefore \forall x\, P(x)$	Universal generalization
$\exists x\, P(x) \therefore P(c)$ for some element c	Existential instantiation
$P(c)$ for some element $c \therefore \exists x\, P(x)$	Existential generalization

Universal modus ponens: $\forall x\,(P(x) \to Q(x)), P(a), \therefore Q(a)$

Universal modus tollens: $\forall x\,(P(x) \to Q(x)), \neg Q(a), \therefore \neg P(a)$

Introduction to Proofs

Theorem / Fact / Result (定理): A statement that can be shown to be true.

Propositions: Less important theorems.

Axiom / Postulate (公理): Statements we assume to be true.

Lemma (引理): A less important theorem that is helpful in the proof of

other results. (plural lemmas or lemmata)

Corollary (推论): Theorem that can be established directly from a theorem that has been proved.

Conjecture (猜想): Statement that is being proposed to be a true statement.

Direct Proof: Direct proof of a conditional statement $p \rightarrow q$ is constructed when the first step is the assumption that p is true; subsequent steps are constructed using rules of inference, with the final step showing that q must also be true.

Indirect Proofs: Proofs that do not start with the premises and end with the conclusion.

Proof by Contraposition (证明逆否命题): We take $\neg q$ as a premise, and using axioms, definitions, and previously proven theorems, together with rules of inference, we show that $\neg p$ must follow.

Vacuous Proofs: If we can show that p is false, then we have a proof of the conditional statement $p \rightarrow q$.

Trivial Proof: By showing that q is true, it follows that $p \rightarrow q$ must also be true.

Proofs by Contradiction (反证): We can prove that p is true if we can show that $\neg p \rightarrow (r \wedge \neg r)$ is true for some proposition r. i.e. r is premise and $\neg r$ is proved if $\neg p$.

To rewrite a proof by contraposition of $p \rightarrow q$ as a proof by contradiction, we suppose that both p and $\neg q$ are true. Then, we use the steps from the proof of $\neg q \rightarrow \neg p$ to show that $\neg p$ is true.

Proofs of Equivalence: $(p \leftrightarrow q) \leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$. (if and only if)

$(p_1 \leftrightarrow p_2 \leftrightarrow \cdots \leftrightarrow p_n) \leftrightarrow (p_1 \rightarrow p_2) \wedge (p_2 \rightarrow p_3) \wedge \cdots \wedge (p_n \rightarrow p_1)$.

Counterexamples: To show that a statement of the form $\forall x\, P(x)$ is false, we only need to find a counterexample.

Mistakes in proofs: Division by zero, Affirming the conclusion, Denying the hypothesis, Begging the question.

Proof Methods and Strategy

Exhaustive Proof (Proofs by exhaustion): $[(p_1 \vee p_2 \vee \cdots \vee p_n) \rightarrow q] \leftrightarrow [(p_1 \rightarrow q) \wedge (p_2 \rightarrow q) \wedge \cdots \wedge (p_n \rightarrow q)]$

Eliminate cases when using exhaustive proof.

Without loss of generality (WLOG): Other cases can be proved with the same method as this case.

Exhaustive proof can be invalid if not all the cases are covered.

Existence proof: A proof of a proposition of the form $\exists x P(x)$.

Constructive: Given by finding a witness.

Nonconstructive: Some other way, e.g. negation leads to contradiction.

Uniqueness Proof: Assert that there is exactly one element with this property. $\exists x(P(x) \wedge \forall y(y = x \rightarrow \neg P(y)))$.

Forward and backward reasoning: ⋯

Adapting existing proofs: ⋯

Look for counterexamples: ⋯

Tiling: Color the board with n colors. If top-right and bottom-left square is removed, the number of squares each color is unequal, but each

domino / polymino must cover exactly one square of each color, so tiling is impossible.

Basic Structures

Set

Set: A set is an unordered collection of objects. $a \in A$, $a \notin A$.

Roster method: List all the members of a set. Can use $\cdots$ when the general pattern is obvious.

Set builder pattern: $\{x \in set \mid predicate(x)\}$ or $\{x \mid predicate(x), x \in set\}$

Name | Description $\mathbb{N}$ | the set of natural numbers $\mathbb{Z}$ | the set of integers $\mathbb{Z}^+$ | the set of positive integers $\mathbb{Q}$ | the set of rational numbers $\mathbb{R}$ | the set of real numbers $\mathbb{R}^+$ | the set of positive real numbers $\mathbb{C}$ | the set of complex numbers

Interval | Set a, b | $\{x \mid a \le x \le b\}$ [a, b) | $\{x \mid a \le x < b\}$ (a, b] | $\{x \mid a < x \le b\}$ (a, b) (Open) | $\{x \mid a < x < b\}$

Equal: Two sets are equal if and only if they have the same elements. $\forall x(x \in A \leftrightarrow x \in B). A = B.$

The order and repetition of elements does not matter: {5, 3, 3, 1} = {1, 3, 5}

Empty/null set: $\varnothing$

Singleton set: Set with a single element.

$\varnothing$ is not $\{\varnothing\}$.

Venn diagram: Rectangle for universal set, circle for set, dot for element.

Subset: The set A is a subset of B if and only if every element of A is also an element of B. If and only if $\forall x(x \in A \rightarrow x \in B)$. $A \subseteq B$.

Showing that A is a Subset of B: To show that $A \subseteq B$, show that if x belongs to A then x also belongs to B.

Showing that A is Not a Subset of B: To show that $A \subsetneq B$, find a single $x \in A$ such that $x \notin B$.

For every set S, $\varnothing \subseteq S$ and $S \subseteq S$.

Proper subset: $A \subseteq B$ and $A \neq B$. $A \subset B$.

Showing Two Sets are Equal: To show that two sets A and B are equal, show that $A \subseteq B$ and $B \subseteq A$.

Size of a set: If there are exactly n distinct elements in S where n is a nonnegative integer, then S is a finite set and that n is the cardinality of S, written as $|S|$.

Infinite set: A set is said to be infinite if it is not finite.

Power set: Given a set S, the power set of S is the set of all subsets of the set S. P(S).

e.g. $P(A) \in P(B) \Rightarrow P(A) \subseteq B \Rightarrow A \in B$

We can reconstruct the original set from the union of all element sets in its power set. So power set uniquely identifies a set.

$\varnothing$ cannot be a power set.

Ordered n-tuple: The ordered collection that has a_1 as its first element, a_2 as its second element, $\cdots$, and a_n as its nth element. (a_1, a_2, $\cdots$,a_n).

Cartesian Product: $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$.

$|A \times B| = |A||B|$

$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \cdots, a_n) \mid a_i \in A_i \text{ for } i = 1, 2, \cdots, n\}$.

Relation: A subset R of the Cartesian product A × B is called a relation.

Using Set Notation with Quantifiers: $\forall x \in S(P(x))$ is shorthand for $\forall x(x \in S \rightarrow P(x))$.

Truth set: The truth set of P to be the set of elements x in D for which P(x) is true.

Set Operations

Union: The union of the sets A and B, denoted by $A \cup B$, is the set that contains those elements that are either in A or in B, or in both. $A \cup B = \{x \mid x \in A \vee x \in B\}$.

Intersection: The intersection of the sets A and B, denoted by $A \cap B$, is the set containing those elements in both A and B. $A \cap B = \{x \mid x \in A \wedge x \in B\}$.

Disjoint: Two sets are called disjoint if their intersection is the empty set.

Difference: The difference of A and B, denoted by $A - B$ (or $A \setminus B$), is the set containing those elements that are in A but not in B. The difference of A and B is also called the complement of B with respect to A. $A - B = \{x \mid x \in A \wedge x \in / B\}$.

Complement: Let U be the universal set. The complement of the set A, denoted by $\overline{A}$, is the complement of A with respect to U . Therefore, the complement of the set A is U − A. $A - B = A \cap \overline{B}$.

Set Identities:

Identity	Name
$A \cap U = A$	
$A \cup \varnothing = A$	Identity laws
$A \cup U = U$	
$A \cap \varnothing = \varnothing$	Domination laws
$A \cup A = A$	
$A \cap A = A$	Idempotent laws
$\overline{(\overline{A})} = A$	Complementation law
$A \cup B = B \cup A$	
$A \cap B = B \cap A$	Commutative laws
$A \cup (B \cup C) = (A \cup B) \cup C$	
$A \cap (B \cap C) = (A \cap B) \cap C$	Associative laws
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	Distributive laws
$\overline{A \cap B} = \overline{A} \cup \overline{B}$	
$\overline{A \cup B} = \overline{A} \cap \overline{B}$	De Morgan’s laws
$A \cup (A \cap B) = A$	
$A \cap (A \cup B) = A$	Absorption laws
$A \cup \overline{A} = U$	
$A \cap \overline{A} = \varnothing$	Complement laws

Prove set identity: Use set builder pattern and definition.

Membership table: Set identities can also be proved using membership tables. We consider each combination of sets that an element can belong to and verify that elements in the same combinations of sets belong to both the sets in the identity. To indicate that an element is in a set, a 1 is used; to indicate that an element is not in a set, a 0 is used.

Generalized Unions and Intersections: Because unions and intersections of sets satisfy associative laws, the sets $A \cup B \cup C$ and $A \cap B \cap C$ are well defined.

The union of a collection of sets: The set that contains those elements that are members of at least one set in the collection. \cup_{i=1}{n}{A_i}.

The intersection of a collection of sets is the set that contains those elements that are members of all the sets in the collection. \cap_{i=1}{n}{A_i}.

Computer Representation of Sets: One of the methods is to use bit strings.

Function

Function/Mapping/Transformation: Let A and B be nonempty sets. A function f from A to B is an assignment of exactly one element of B to each element of A. $f : A \to B$. $f(a) = b$.

If f is a function from A to B, we say that A is the domain of f and B is the codomain of f. If f (a) = b, we say that b is the image of a and a is a preimage of b. The range, or image, of f is the set of all images of elements of A. Also, if f is a function from A to B, we say that f maps A to

B.

Two functions are equal: When they have the same domain, have the same codomain, and map each element of their common domain to the same element in their common codomain.

Let f_1 and f_2 be functions from A to R, then f_1 + f_2 and f_1 f_2 are also functions from A to R defined for all $x \in A$ by : (f_1 + f_2)(x) = f_1(x) + f_2(x), (f_1f_2)(x) = f_1(x)f_2(x).

The image of S under the function f: $f(S) = \{t \mid \exists s \in S (t = f(s))\} = \{f(s) \mid s \in S\}$.

One-to-one/injection (单射): if and only if f (a) = f (b) implies that a = b for all a and b in the domain of f. A function is said to be injective if it is one-to-one. $\forall a \forall b(f(a) = f(b) \rightarrow a = b)$

Increasing if $f(x) \leq f(y)$, and strictly increasing: if f (x) <f (y), whenever x < y and x and y are in the domain of f.

Decreasing if $f(x) \geq f(y)$, and strictly decreasing if f (x) > f (y), whenever x < y and x and y are in the domain of f.

Onto/surjection (满射): if and only if for every element $b \in B$ there is an element $a \in A$ with f (a) = b. A function f is called surjective if it is onto.

One-to-one correspondence / bijection (双射): if it is both one-to-one and onto. We also say that such a function is bijective.

To show that f is injective: Show that if f (x) = f (y) for arbitrary $x, y \in A$ with $x \neq y$, then x = y.

To show that f is not injective: Find particular elements $x, y \in A$ such that $x \neq y$ and f (x) = f (y).

To show that f is surjective: Consider an arbitrary element $y \in B$ and

find an element $x \in A$ such that $f(x) = y$.

To show that f is not surjective: Find a particular $y \in B$ such that $f(x) \neq y$ for all $x \in A$.

Inverse function: Let f be a one-to-one correspondence from the set A to the set B. The inverse function of f is the function that assigns to an element b belonging to B the unique element a in A such that $f(a) = b$. The inverse function of f is denoted by $f^{\{-1\}}$. Hence, $f^{\{-1\}}(b) = a$ when $f(a) = b$.

Composition of functions: Let g be a function from the set A to the set B and let f be a function from the set B to the set C. The composition of the functions f and g, denoted for all $a \in A$ by $f \circ g$, is defined by $(f \circ g)(a) = f(g(a))$.

The graphs of functions: ⋯

Some important functions: Floor(x): [x]

Sequence

Sequence: A function from a subset of the set of integers (usually either the set {0, 1, 2, ⋯}or the set {1, 2, 3, ⋯}) to a set S. We use the notation a_n to denote the image of the integer n. We call a_n a term of the sequence.

Geometric progression: $a, ar, ar^2, \cdots, ar^n, \cdots$ whre a is initial term, r is common ratio.

Arithmetic progression: $a, a + d, a + 2d, \cdots, a + nd, \cdots$ where a is initial term, d is common difference.

String: Finite sequence, where length is the number of terms in the

string.

Empty string: λ.

Recurrence relation: An equation that expresses a n in terms of one or more of the previous terms of the sequence.

Initial condition: Specify the terms that precede the first term where the recurrence relation takes effect.

Fibonacci sequence: f_0, f_1, f_2, ⋯ , is defined by the initial conditions f_0 = 0, f_1 = 1, and the recurrence relation f_n = f_{n−1} + f{n−2}.

Closed formula: We say that we have solved the recurrence relation together with the initial conditions when we find an explicit formula, called a closed formula, for the terms of the sequence.

Finding closed formula: Iteration, forward substitution or backward substitution.

Lucas sequence: Fibonacci sequence with different initial condition.

Summation notation: \sum_{i=m}^n{f(i)}, where i is index of summation, m is lower limit, n is upper limit.

\sum_{i=0}^n(ar^i)=\frac{ar^{n+1}-a}{r-1},r{\neq}1

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

\sum_{i=0}^ni^2=\frac{n(n+1)(2n+1)}{6}

\sum_{i=0}^ni^3=\frac{n^2(n+1)^2}{4}

\sum_{i=0}^{\infty}x^i,|x|<1=\frac{1}{1-x}

\sum_{i=0}^{\infty}ix^{i-1},|x|<1=\frac{1}{(1-x)^2}

Cardinality of Sets

Same cardinality: If and only if there is a one-to-one correspondence from A to B. |A| = |B|

Less cardinality: If there is a one-to-one function from A to B, the cardinality of A is less than or the same as the cardinality of B. $|A| \leq |B|$. When $|A| \leq |B|$ and A and B have different cardinality, we say that the cardinality of A is less than the cardinality of B. $|A| < |B|$.

Countable set: A set that is either finite or has the same cardinality as the set of positive integers is called countable.

Uncountable set: A set that is not countable is called uncountable. When an infinite set S is countable, we denote the cardinality of S by $\aleph_0$(where ℵ is aleph). $|S| = \aleph_0$. S has cardinality “aleph null”.

Prove by showing one-to-one correspondence, or can be listed.

$\mathbb{Z}$ is countable: f(n) = n / 2, when n is even; -(n - 1) / 2, when n is odd.

$\mathbb{Q}^+$ is countable: Use Cantor Diagonalization Argument for listing.

$\mathbb{R}$ is uncountable: $0.d_i, d_i = 0$, if $d_{ii} \neq 0$; 1, if $d_{ii} = 0$ cannot be listed.

If A and B are countable sets, then $A \cup B$ is also countable.

Schröder-Bernstein Theorem: If A and B are sets with $|A| \leq |B|$ and $|B| \leq |A|$, then $|A| = |B|$. In other words, if there are one-to-one functions f from A to B and g from B to A, then there is a one-to-one correspondence between A and B.

Can use two different functions to prove same cardinality, according to Schröder-Bernstein Theorem.

Computable / uncomputable function: If there is a computer program in

some programming language that finds the values of this function. If a function is not computable we say it is uncomputable.

Proof for existence of uncomputable function: Programs are countable (Binary can be listed), but functions are not, e.g. $\mathbb{Z}^+$ -> $\mathbb{Z}^+$ is not.

The continuum hypothesis: (We have |P(ℤ^+)| = 2^{\aleph_0} = |ℝ| = c.) There is no cardinality such that it is greater than \aleph_0 and less than c, or say, c = aleph_1 in sequence aleph_0, aleph_1, aleph_2, ...

Proving power set related cardinality: Use bit string for element sets of power set.

(0, 1) and decimal representation rocks for proof.

|(0, 1)| = |ℝ|, can be proved with Schröder-Bernstein Theorem.

Algorithms

Algorithms

Algorithm: An algorithm is a finite sequence of precise instructions for performing a computation or for solving a problem.

Properties of algorithms:

1. Input. An algorithm has input values from a specified set.
2. Output. From each set of input values an algorithm produces output values from a specified set. The output values are the solution to the problem.
3. Definiteness. The steps of an algorithm must be defined precisely.
4. Correctness. An algorithm should produce the correct output values

for each set of input values.

5. Finiteness. An algorithm should produce the desired output after a finite (but perhaps large) number of steps for any input in the set.
6. Effectiveness. It must be possible to perform each step of an algorithm exactly and in a finite amount of time.
7. Generality. The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

Max: Use temporary variable max and iterate.

Linear Search: Use temporary variable location and iterate.

Binary search: ⋯

Optimization problems: The goal of such problems is to find a solution to the given problem that either minimizes or maximizes the value of some parameter.

Greedy algorithms: Algorithms that make what seems to be the “best” choice at each step.

Greedy penny exchange: Proved by contradiction.

The halting problem: ⋯

The Growth of Functions

Big-O Notation: Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that f (x) is O(g(x)) if there are constants C and k such that $|f(x)| \leq C|g(x)|$ whenever x > k. [Read as “f (x) is big-oh of g(x).”]

Let f(x) = $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, where $a_0, a_1, \cdots, a_n$

−1}, a_n are real numbers. Then f(x) is $O(x^n)$.

f_1(x) is O(g_1(x)), f_2(x) is O(g_2(x)) => (f_1 + f_2)(x) is O(max(|g_1(x)|, |g_2(x)|)).

f_1(x) is O(g_1(x)), f_2(x) is O(g_2(x)) => (f_1f_2)(x) is O(g_1(x)g_2(x))

We say that f(x) is Ω(g(x)) if there are positive constants C and k such that |f(x)| ≥ C|g(x)| whenever x > k. [Read as “f(x) is big-Omega of g(x).”]

We say that f(x) is Θ(g(x)) if f(x) is O(g(x)) and f(x) is Ω(g(x)). When f(x) is Θ(g(x)) we say that f is big-Theta of g(x), that f(x) is of order g(x), and that f(x) and g(x) are of the same order.

Complexity of Algorithms

Time complexity: ⋯

Space complexity: ⋯

Worst case complexity: ⋯

Average case complexity: ⋯

Tractable: A problem that is solvable using an algorithm with polynomial worst-case complexity is called tractable, otherwise it is intractable.

Solvable: Can be solved, otherwise unsolvable.

P: Tractable.

NP: Solution can be checked in polynomial time.

NP-Complete: If any is P, all NP is P.

Induction and Recursion

Mathematical Induction

Principle Of Mathematical Induction: To prove that P (n) is true for all positive integers n, where P (n) is a propositional function, we complete two steps:

Basis Step: We verify that P (1) is true.

Inductive Step: We show that the conditional statement P (k) $\rightarrow$ P (k + 1) is true for all positive integers k.

$(P(1) \wedge \forall k(P(k) \rightarrow P(k+1))) \rightarrow \forall n P(n)$

Can rely on multiple basis and induct on each one.

Strong Induction and Well-Ordering

In a proof by strong induction, the inductive step shows that if P (j) is true for all positive integers not exceeding k, then P (k + 1) is true.

Strong induction / second principle of mathematical induction / complete induction: To prove that P(n) is true for all positive integers n, where P(n) is a propositional function, we complete two steps:

Basis step: We verify that the proposition P (1) is true.

Inductive step: We show that the conditional statement $[P(1) \wedge P(2) \wedge \cdots \wedge P(k)] \rightarrow P(k+1)$ is true for all positive integers k.

The well-ordering property: Every nonempty set of nonnegative integers has a least element.

Recursive definitions and Structural Induction

Recursion : Defining a object in terms of itself.

Recursive / Inductive definition:

- Basis step: Specify the value of the function at zero.
- Recursive step: Give a rule for finding its value at an integer from its values at smaller integers.

Structural Induction:

Prove statement about recursive defined structures with mathematical induction.

Counting

The Basics of Counting

The product rule: Suppose that a procedure can be broken down into a sequence of two 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 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 set of n_1 ways is the same as any of the set of n_2 ways, then there are $n_1 + n_2$ ways to do the task.

The subtraction rule: If a task can be done in either n_1 ways or n_2 ways, then the number of ways to do the task is $n_1 + n_2$ minus the number of ways to do the task that are common to the two different ways.

The division rule: There are n/d ways to do a task if it can be done using

a procedure that can be carried out in n ways, and for every way w, exactly d of the n ways correspond to way w.

As in Example 20 in the textbook, page 394 in pdf.

Counting problems can be solved using tree diagrams.

The Pigeonhole Principle

The pigeonhole principle: If k is a positive integer and k + 1 or more objects are placed into k boxes, then there is at least one box containing two or more of the objects.

Corollary 1: A function f from a set with k + 1 or more elements to a set with k elements is not one-to-one.

The generalized pigeonhole principle: If N objects are placed into k boxes, then there is at least one box containing at least $\lceil N/k \rceil$ objects.

Reverse minimum: (m - 1) * k + 1

Elegant applications: ⋯

Subsequence: Picking elements while the original order is preserved.

Theorem: Every sequence of n^2+1 distinct real numbers contains a subsequence of length $n+1$ that is either strictly increasing or strictly decreasing.

The Ramsey number R(m, n), where m and n are positive integers greater than or equal to 2, denotes the minimum number of people at a party such that there are either m mutual friends or n mutual enemies, assuming that every pair of people at the party are friends or enemies.

Permutations and Combinations

If n is a positive integer and r is an integer with $1 \leq r \leq n$, then there are $P(n, r) = n(n - 1)(n - 2) \cdots (n - r + 1) = n! / (n - r)!$ r-permutations of a set with n distinct elements.

$P(n, 0) = 1$

The number of r-combinations of a set with n elements, where n is a nonnegative integer and r is an integer with $0 \leq r \leq n$, equals $C(n, r) = n! / (r!(n - r)!)$.

Binomial coefficient: $(n, r)^T = C(n, r)$.

$P(n, r) = C(n, r) \cdot P(r, r)$.

$C(n, r) = C(n, n - r)$.

A combinatorial proof of an identity is a proof that uses counting arguments to prove that both sides of the identity count the same objects but in different ways, or a proof that is based on showing that there is a bijection between the sets of objects counted by the two sides of the identity. These two types of proofs are called double counting proofs and bijective proofs, respectively.

Binomial Coefficient

The binomial theorem: $(x+y)^n=\sum_{j=0}^{n}\binom{n}{j}x^{n-j}y^j$

$\sum_{k=0}^{n}\binom{n}{k}=(1+1)^n=2^n$

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

$\sum_{k=0}^{n}2^k\binom{n}{k}=(2+1)^n=3^n$

Pascal’s identity: \binom{n+1}{k}=\binom{n}{k-1}+\binom{n}{k}

So Pascal’s triangle.

Vandermonde’s identity: \binom{m+n}{r}=\sum_{k=0}^{r}\binom{m}{r-k}\binom{n}{k}

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

\binom{n+1}{r+1}=\sum_{j=r}^{n}\binom{j}{r}

Prove by combinatorial argument: Use choosing subset, use bit string.

Generalized Permutations and Combinations

The number of r-permutations of a set of n objects with repetition allowed is n^r.

There are C(n + r − 1, r) = C(n + r − 1, n − 1) r-combinations from a set with n elements when repetition of elements is allowed.

Proved by stars and bars.

Method: Stars and bars abstraction.

P426 Counting solutions to equation

Notice non-negative or positive integer, the latter implies x_i \ge 1

P427 Nested loop

The number of different permutations of n objects, where there are n_1 indistinguishable objects of type 1, n_2 indistinguishable objects of type 2, ⋯ , and n_k indistinguishable objects of type k, is \frac{n!}{n_1!n_2! ⋯ n_k!} = \frac{A(n,n)}{A(n_1,n_1)A(n_2,n_2)...A(n_k,n_k)}.

P427 Word letter reordering

Distinguishable objects and distinguishable boxes: The number of ways to distribute n distinguishable objects into k distinguishable boxes so that n_i objects are placed into box i, i = 1, 2, ⋯ , k, equals $\frac{n!}{n_1!n_2!\cdots n_k!}$

Indistinguishable objects and distinguishable boxes: The number of ways to distribute n indistinguishable objects into k distinguishable boxes so that n_i objects are placed into box i, i = 1, 2, ⋯ , k, equals the n-combination from a set with k elements when repetition is allowed, according to its proof, C(k + n - 1, n).

Distinguishable objects and indistinguishable boxes:

Can enumerate by n into m, ⋯ , but no simple closed formula.

Stirling numbers of the second kind: S(n, j) denote the number of ways to distribute n distinguishable objects into j indistinguishable boxes so that no box is empty.

Then the number of ways to distribute n distinguishable objects into k indistinguishable boxes equals $\sum_{j=1}^{k}S(n,j)$.

$S(n,j)=\frac{1}{j!}\sum_{i=0}{j-1}(-1)^i\binom{j}{i}(j-i)^n$

Indistinguishable objects and indistinguishable boxes:

List partition by decreasing order.

If $p_k(n)$ is the number of partitions of n into at most k positive integers, then there are $p_k(n)$ ways to distribute n indistinguishable objects into k indistinguishable boxes.

Generating Permutations and Combinations

Generating permutations: Lexicographic.

Next permutation: Find last pair such that \(a_j, put minimum of a_{j+1}, ⋯, a_n that is greater than a_j at a_j, and list remaining in increasing order.

Generating subsets: Use bit string.

Generating r-combinations: Lexicographic.

Next permutation of {1, 2, ⋯ , n}: Find last a_i such that a_i{\ne}n-r+i, replace a_i with a_i+1, a_j with a_i+j-i+1 (increasing from a_i+1). (This is natural.)

Advanced Counting Techniques

Applications of Recurrence Relations

Recurrence relation: A rule for determining subsequent terms from those that precede them.

Solution of a recurrence relation: A sequence is called a solution of a recurrence relation if its terms satisfy the recurrence relation.

Rabbits: f_n = f_{n-1} + f_{n-2}

Hanoi: H_n=2H_{n-1}+1, H_1=1

Bit string without two consecutive zeros: a_n = a_{n-1} + a_{n-2}

Dynamic programming.

Solving Linear Recurrence Relations

Linear homogeneous recurrence relation of degree k with constant coefficients: A recurrence relation of the form a_n=c_1a_{n-1}+c_2a_{n-

2}+⋯+c_ka_{n-k}, where c_1, c_2, ⋯, c_k are real numbers, and c_k{\neq}0

Characteristic equation: Suppose a_n=r^n, then r^k-c_1r^{k-1}-c_2r^{k-2}-...-c_{k-1}r-c_k=0 is the characteristic equation. The solutions are called characteristic roots.

Theorem 1: Let c_1 and c_2 be 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} if and only if a_n=α_1r_1^n+α_2r_2^n for n = 0, 1, 2, ⋯, where α_1 and α_2 are constants.

And then solve with initial conditions.

Theorem 2: Let c_1 and c_2 be real numbers with c_2\neq0. Suppose that r^2-c_1r-c_2=0 has only one root r_0. A sequence \{a_n\} is a solution of the recurrence relation a_n=c_1a_{n-1}+c_2a_{n-2} if and only if a_n=α_1r_0^n+α_2nr_0^n, for n = 0, 1, 2, ⋯, where α_1 and α_2 are constants.

And then solve with initial conditions.

Theorem 3: Let c_1, c_2, ..., c_k be real numbers. Suppose that the characteristic equation r^k-c_1r^{k-1}-...-c_k=0 has k distinct roots r_1, r_2, ..., r_k. Then a sequence \{a_n\} is a solution of the recurrence relation a^n=c_1a^{n-1}+c_2a^{n-2}+...+c_ka^{n-k} if and only if a^n = α_1r_1^n+α_2r_2^n+⋯+α_kr_k^n for n = 0, 1, 2, ⋯, where α_1, α_2, ..., α_k are constants.

Theorem 4: Let c_1, c_2, ⋯, c_k be real numbers. Suppose that the characteristic equation r^k - c_1r^k-1 - ⋯ - c_k = 0 has t distinct roots r_1, r_2, ⋯, r_t with multiplicities m_1, m_2, ⋯, m_t, respectively, so that

m_i\geq1 for i = 1, 2, ⋯, t and m_1+m_2+⋯+m_t=k. Then a sequence {a_n} is a solution of the recurrence relation a_n=c_1a_{n-1}+c_2a_{n-2}+⋯+c_ka_{n-k} if and only if a_n=(α_{1, 0}+α_{1, 1}n+⋯+α_{1, m_1-1}n^{m_1-1})r_1^n+(α_{2, 0}+α_{2, 1}n+⋯+α_{2, m_2-1}n^{m_2-1})r_2^n+⋯+(α_{t, 0}+α_{t, 1}^n+⋯+α_{t, m_t-1}n^{m_t-1})r_t^n for n = 0, 1, 2, ⋯, where α_{i, j} are constants for 1\leq{i}\leq{t} and 0\leq{j}\leq{m}_i-1.

Theorem 5: If \{a_n^{(p)}\} is a particular solution of the nonhomogeneous linear recurrence relation with constant coefficients a_n=c_1a_{n-1}+c_2a_{n-2}+...+c_ka_{n-k}+F(n), then every solution is of the form \{a_n^{(p)}+a_n^{(h)}\}, where \{a_n^{(h)}\} is a solution of the associated homogeneous recurrence relation a_n=c_1a_{n-1}+c_2a_{n-2}+...+c_ka_{n-k}.

Theorem 6: If F(n)=(b_tn^t+b_{t-1}n^{t-1}+...+b_1n+b_0)s^n, when s is not a root of the characteristic equation of the associated linear homogeneous recurrence relation, there is a particular solution of the form (p_tn^t+p_{t-1}n^{t-1}+...+p_1n+p_0)s^n; When s is a root of this characteristic equation and its multiplicity is m, there is a particular solution of the form n^m(p_tn^t+p_{t-1}n^{t-1}+...+p_1n+p_0)s^n.

Generating Functions

The (ordinary) generating function for the sequence a_0,a_1,...,a_k,... of real numbers is the infinite series G(x)=a_0+a_1x+...+a_kx^k+...=\sum_{k=0}^\infty{a}_kx^k.

We can define generating functions for finite sequences of real numbers by setting a_{n+1}=a_{n+2}=...=0

$f(x)=\frac{1}{1-x}$ is the generating function of $\{1\}$ for $|x|<1$.

$f(x)=\frac{1}{1-ax}$ is the generating function of $\{a^n\}$ for $|ax|<1$.

Let $f(x)=\sum_{k=0}^\infty{a}_kx^k$ and $g(x)=\sum_{k=0}^\infty{b}_kx^k$, then $f(x)+g(x)=\sum_{k=0}^\infty(a_k+b_k)x^k$ and $f(x)g(x)=\sum_{k=0}^\infty(\sum_{j=0}^ka_jb_{k-j})x^k$.

Let u be a real number and k a nonnegative integer. Then the extended binomial coefficient $\binom{u}{k}$ is defined by $\binom{u}{k}=u(u-1)...(u-k+1)$ if $k>0$; 1 if $k=0$.

$\binom{-n}{r}=(-1)^r\binom{n+r-1}{r}$

The extended binomial theorem: Let x be a real number with $|x|<1$ and let u be a real number. Then $(1+x)^u=\sum_{k=0}^\infty\binom{u}{k}x^k$.

Find number of solutions: $e_1+e_2+...+e_n=C$, $l_i\leq{e}_i\leq{u_i}$, then it is the coefficient of x^C from $(x^{l_i}+...+x^{u_i})...(...)$.

Form value r with tokens of value t_i: When order matters, ways of exactly n tokens is the coefficient of x^r from $(x^{t_i}+...)^n$, so for all it is the coefficient of x^r from $1+...+(x^{t_i}+...)^n$; else, it is the coefficient of x^r from $(1+...+(x^{t_i})^n)+...$.

More powerful and constraint-friendly then simple permutation and combination.

Solve recurrence relations: Multiply x^n to the recurrence relation. Substitute the multiplied relation into $G(x)=\sum_{k=0}^\infty{a}_kx^k=...$, solve for $G(x)$, then make it a summation to see a^n.

Proving identity: Take combination as a coefficient of certain term.

Inclusion-Exclusion

|A\cup{B}|=|A|+|B|-|A\cap{B}|.

|A\cup{B}\cup{C}|=|A|+|B|+|C|-|A\cap{B}|-|B\cap{C}|-|C\cap{A}|+|A\cap{B}\cap{C}|

...

Number of integers divisible: \lfloor{n}/a\rfloor+\lfloor{n}/b\rfloor-\lfloor{n}/ab\rfloor.

Applications of Inclusion-Exclusion

Asking element count having none of some properties: Use inclusion-exclusion.

The number of primes (The sieve of Eratosthenes): A composite number is divisible by a prime smaller than its square root.

The number of onto functions: The shouldn’t have properties are not having element i in the range.

Let m and n be positive integers with m\geq{n}. Then, there are n^m-C(n,1)(n-1)^m+C(n,2)(n-2)^m-...+(-1)^{n-1}C(n,n-1)1^m onto functions from a set with m elements to a set with n elements.

Derangement: A permutation of objects that leaves no object in its original position.

The number of derangements of a set with n elements is D_n=n![1-\frac{1}{1!}+\frac{1}{2!}-\frac{1}{3!}+...+(-1)^n\frac{1}{n!}]\to{n}!e^{-1}.

For arranging differently between two times, the number is n!D_n=

(n!)^2[1-\frac{1}{1!}+\frac{1}{2!}-\frac{1}{3!}+...+(-1)^n\frac{1}{n!}] because the first arrangement can have n! ways.

Relations

Relations, Their Properties and Representations

Binary relation: Let A and B be sets. A binary relation from A to B is a subset of A\times{B}. A{R}B or A\not{R}B.

Matrix representation: M_R,m_{ij}=(a_i,b_j)\in{R}.

Functions can be relations.

Relation on a set: A relation on a set A is a relation from A to A.

Reflexive: A relation R on a set A is called reflexive if (a,a)\in{R} for every element a\in{A}.

Irreflexive: A relation R on the set A is irreflexive if for every a\in{A}, (a,a)\not\in{R}.

Symmetric: A relation R on a set A is called symmetric if (b,a)\in{R} whenever (a,b)\in{R}, for all a,b\in{A}. In matrix it is 1 to 1 and 0 to 0 mirrored by the main diagonal, or M_R=(M_R)^T

Asymmetric: A relation R is called asymmetric if (a,b)\in{R} implies that (b,a)\not\in{R}. (So the main diagonal are all zeros.)

Antisymmetric: A relation R on a set A such that for all a,b\in{A}, if (a,b)\in{R} and (b,a)\in{R}, then a=b is called antisymmetric. In matrix it is 1 to 0, 0 to 1 or 0 to 0 mirrored by the main diagonal.

Antisymmetric is not Asymmetric, but Asymmetric is Antisymmetric.

Transitive: A relation R on a set A is called transitive if whenever (a,b)\in{R} and (b,c)\in{R}, then (a,c)\in{R}, for all a,b,c\in{A}. Combining Relations: Relations can be combined like sets.

M_{R_1\cup{R}_2}=M_{R_1}\vee{M}_{R_2}

M_{R_1\cap{R}_2}=M_{R_1}\wedge{M}_{R_2}

M_{R\circ{S}}=M_S\bigodot{M}_R (\bigodot stands for boolean product)

M_{R^n}=(M_R)^n

Symmetric difference: The symmetric difference of A and B, denoted by A\bigoplus{B}, is the set containing those elements in either A or B, but not in both A and B.

M_\bigoplus{R} is the entry-wise XORed matrix.

Composition: The composite of R and S is the relation consisting of ordered pairs (a,c), where a\in{A}, c\in{C}, and for which there exists an element b\in{B} such that (a,b)\in{R} and (b,c)\in{S}. We denote the composite of R and S by S\circ{R}.

S\circ{R} is from right to left (inside to outside)!

Composition can be done by matrix multiplication.

Power: Let R be a relation on the set A. The powers R^n, n=1,2,3,..., are defined recursively by R^1=R and R^{n+1}=R^n\circ{R}.

Theorem: The relation R on a set A is transitive if and only if R^n\subseteq{R} for n=1,2,3,...

Inverse relation: R^{-1}, with pairs inverted.

Relations on a finite set can also be represented by digraphs (directed

graphs).

(R\cup{S})^{-1}=R^{-1}\cup{S}^{-1}

(R\cap{S})^{-1}=R^{-1}\cap{S}^{-1}

(\overline{R})^{-1}=\overline{R^{-1}}

(R-S)^{-1}=R^{-1}-S^{-1}

(A\times{B})^{-1}=B\times{A}

Closures of Relations

Closure of R with respect to P: The relation with property P containing R such that it is a subset of every relation with property P containing R.

Diagonal relation: \Delta=\{(a,a)|a\in{A}\}.

Reflexive closure of R: The smallest reflexive relation that contains R. Formed by R\cup\Delta.

Symmetric closure of R: The smallest symmetric relation that contains R. Formed by R\cup{R}^{-1}.

Transitive closure of R: The smallest transitive relation that contains R.

Path: A sequence of consecutive edges, denoted by x_0,x_1,x_2,...,x_{n-1},x_n, with length n.

Circuit (or cycle): A path of length n\geq{1} that begins and ends at the same vertex.

Path on relation: There is a path from a to b in R if there is a sequence of elements a,x_1,x_2,...,x_{n−1},b with (a,x_1)\in{R}, (x_1,x_2)\in{R}, ⋯, and (x_{n−1},b)\in{R}.

Theorem 1: Let R be a relation on a set A. There is a path of length n,

where n is a positive integer, from a to b if and only if (a,b)\in{R}^n.

Connectivity relation: Let R be a relation on a set A. The connectivity relation R^* consists of the pairs (a,b) such that there is a path of length at least one from a to b in R. R^*=\bigcup_{n=1}^\infty{R}^n

Theorem 2: The transitive closure of a relation R equals the connectivity relation R^*.

Lemma 1: Let A be a set with n elements, and let R be a relation on A. If there is a path of length at least one in R from a to b, then there is such a path with length not exceeding n. Moreover, when a\neq{b}, if there is a path of length at least one in R from a to b, then there is such a path with length not exceeding n−1.

R^*=\bigcup_{i=1}^nR^i

Theorem 3: Let M_R be the zero–one matrix of the relation R on a set with n elements. Then the zero–one matrix of the transitive closure R^* is M_{R^*}=M_R\vee{M}_R^{[2]}\vee{M}_R^{[3]}\vee...\vee{M}_R^{[n]}.

Interior vertices: Vertices of a path excluding the first and the last.

W_0=M_R, W_i=[w_{ij}^{(k)}], where w_{ij} is whether there is a path from v_i to v_j such that all interior vertices are in the first i elements of the list (The list is prepared beforehand).

W_n=M_{R^*}.

Lemma 2: w_{ij}^{[k]}=w_{ij}^{[k-1]}\vee(w_{ik}^{[k-1]}\wedge{w}_{kj}^{[k-1]})

Equivalence Relations

Equivalence Relation: A relation on a set A is called an equivalence

relation if it is reflexive, symmetric, and transitive.

Equivalent: Two elements a and b that are related by an equivalence relation are called equivalent. The notation a\tilde{b} is often used to denote that a and b are equivalent elements with respect to a particular equivalence relation.

Congruence Modulo is an equivalence relation.

Equivalence class: Let R be an equivalence relation on a set A. The set of all elements that are related to an element a of A is called the equivalence class of a. The equivalence class of a with respect to R is denoted by [a]_R. When only one relation is under consideration, we can delete the subscript _R and write [a] for this equivalence class.

Representative of equivalence class: If b\in[a]_R , then b is called a representative of this equivalence class.

Theorem 1: Let R be an equivalence relation on a set A. These statements for elements a and b of A are equivalent:

1. aRb
2. [a]=[b]
3. [a]\cap[b]\neq\varnothing

Partition: Partition of a set S is a collection of disjoint nonempty subsets of S that have S as their union.

Theorem 2: Let R be an equivalence relation on a set S. Then the equivalence classes of R form a partition of S. Conversely, given a partition \{A_i|i\in{I}\} of the set S, there is an equivalence relation R that has the sets A_i,i\in{I}, as its equivalence classes.

The m congruence modulo classes are denoted by [0]_m, [1]_m, ⋯, [m

−1]_m.

Partial Ordering

Partial ordering: A relation R on a set S is called a partial ordering or partial order if it is reflexive, antisymmetric, and transitive.

Partially ordered set (poset): A set S together with a partial ordering R is called a partially ordered set, or poset, and is denoted by (S,R). Members of S are called elements of the poset.

Less/greater than or equal (\leq/\geq), inclusion relation (\subseteq), divisibility relation (|) are all partial orderings.

Less/greater than (</>) are antisymmetric and transitive, but not reflexive, so they are not partial orderings.

Comparable: The elements a and b of a poset (S,\preceq) are called comparable if either a\preceq{b} or b\preceq{a}. When a and b are elements of S such that neither a\preceq{b} nor b\preceq{a}, a and b are called incomparable.

Totally/linearly ordered set: If (S,\preceq) is a poset and every two elements of S are comparable, S is called a totally or linearly ordered set, and \preceq is called a total or linear order. A totally ordered set is also called a chain.

Well-ordered set: (S,\preceq) is a well-ordered set if it is a poset such that \preceq is a total ordering and every nonempty subset of S has a least element.

The principle of well-ordered induction: Suppose that S is a well-ordered set. Then P(x) is true for all x\in{S}, if (inductive step:) For every y\in{S}, if

P(x) is true for all x\in{S} with x\prec{y}, then P(y) is true.

Lexicographic ordering: The lexical ordering \prec on A_1\times{A}_2 is defined by specifying that one pair is less than a second pair if the first entry of the first pair is less than (in A_1) the first entry of the second pair, or if the first entries are equal, but the second entry of this pair is less than (in A_2) the second entry of the second pair.

Hasse diagram: Start with the directed graph for this relation. First, Remove these loops because of reflexivity. Next, remove all edges that must be in the partial ordering because of transitivity. Finally, arrange each edge so that its initial vertex is below its terminal vertex and remove all the arrows on edges.

Covers: An element y\in{S} covers an element x\in{S} if x\prec{y} and there is no element z\in{S} such that x\prec{z}\prec{y}.

Covering relation: The set of pairs (x,y) such that y covers x is called the covering relation of (S,\preceq).

Maximal element: An element of a poset is called maximal if it is not less than any element of the poset. The top element of a Hasse diagram.

Minimal element: An element of a poset is called minimal if it is not greater than any element of the poset. The bottom element of a Hasse diagram.

Greatest element: An element in a poset that is greater than every other element.

Least element: An element in a poset that is less than every other element.

Upper bound: Element greater than or equal to all the elements in a

subset A of S.

Lower bound: Element less than or equal to all the elements in a subset A of S.

Least upper bound: Upper bound that is less than every other upper bound of a subset A of S.

Greatest lower bound: Lower bound that is greater than every other lower bound of a subset A of S.

Lattice: A partially ordered set in which every pair of elements has both a least upper bound and a greatest lower bound is called a lattice.

(P(S),\subseteq/\supseteq) is a lattice, with LUB and GLB being A\cup{B} and A\cap{B}.

Compatible: A total ordering \preceq said to be compatible with the partial ordering R if a\preceq{b} whenever aRb.

Topological sorting: Constructing a compatible total ordering from a partial ordering.

Lemma 1: Every finite nonempty poset (S,\preceq) has at least one minimal element.

Algorithm for topological sorting: Pick the least element and remove it from the poset. Can also be done with a Hasse diagram.

Graphs

(Undirected) graph: A graph G=(V,E) consists of V, a nonempty set of vertices (or nodes) and E, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to

connect its endpoints.

Simple graph: A graph in which each edge connects two different vertices and where no two edges connect the same pair of vertices is called a simple graph.

Infinite graph: A graph with an infinite vertex set or an infinite number of edges is called an infinite graph.

Finite graph: a graph with a finite vertex set and a finite edge set is called a finite graph.

Multigraph: Graphs that may have multiple edges connecting the same vertices are called multigraphs.

Loop: Edges that connect a vertex to itself.

Pseudographs: Graphs that may include loops, and possibly multiple edges connecting the same pair of vertices or a vertex to itself.

Directed graph (digraph): 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 associated with the ordered pair (u,v) is said to start at u and end at v.

Simple directed graph: A directed graph with no loops and no multiple directed edges that start and end at the same vertices.

Directed multigraphs: Directed graphs that may have multiple directed edges from a vertex to a second (possibly the same) vertex.

Multiplicity: When there are m directed edges, each associated to an ordered pair of vertices (u,v), we say that (u,v) is an edge of multiplicity m.

Mixed graph: A graph with both directed and undirected edges.

Graph Terminology and Special Types of Graphs

Adjacent (Neighbor): Two vertices u and v in an undirected graph G are called adjacent (or neighbors) in G if u and v are endpoints of an edge e of G. Such an edge e is called incident with the vertices u and v and e is said to connect u and v.

Neighborhood: The set of all neighbors of a vertex v of $G=(V,E)$, denoted by $N(v)$, is called the neighborhood of v. If A is a subset of V , we denote by $N(A)$ the set of all vertices in G that are adjacent to at least one vertex in A. So, $N(A)=\bigcup_{v\in{A}}N(v)$.

Degree: The degree of a vertex in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes *twice* to the degree of that vertex. The degree of the vertex v is denoted by $deg(v)$.

Theorem 1, The handshaking theorem: Let $G=(V,E)$ be an undirected graph with m edges. Then $2m=\sum_{v\in{V}}deg(v)$. (Note that this applies even if multiple edges and loops are present.)

Theorem 2: An undirected graph has an even number of vertices of odd degree.

Adjacent to/from, initial/terminal vertex: When (u,v) is an edge of the graph G with directed edges, u is said to be adjacent to v and v is said to be adjacent from u. The vertex u is called the initial vertex of (u,v), and v is called the terminal or end vertex of (u,v). The initial vertex and

terminal vertex of a loop are the same.

In/out degree: In a graph with directed edges the in-degree of a vertex v, denoted by deg^−(v), is the number of edges with v as their terminal vertex. The out-degree of v, denoted by deg^+(v), is the number of edges with v as their initial vertex. (Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of this vertex.)

Theorem 3: Let G=(V,E) be a graph with directed edges. Then \sum_{v\in{V}}deg^−(v)=\sum_{v\in{V}}deg^+(v)=|E|.

Underlying undirected graph: The undirected graph that results from ignoring directions of edges is called the underlying undirected graph.

Complete graph: 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.

Noncomplete graph: A simple graph for which there is at least one pair of distinct vertex not connected by an edge.

Cycle: A cycle C_n, n\geq3, consists of n vertices v_1,v_2,...,v_n and edges {v_1,v_2},{v_2,v_3},...,{v_{n−1},v_n},{v_n,v_1}.

Wheel: We obtain a wheel W_n when we add an additional vertex to a cycle C_n, for n\geq3, and connect this new vertex to each of the n vertices in C_n, by new edges.

n-Cube: An n-dimensional hypercube, or n-cube, denoted by Q_n, is a graph that has vertices representing the 2^n bit strings of length n.

Bipartite and bipartition: A simple graph G is called bipartite if its vertex set V can be partitioned into two disjoint sets V_1 and V_2 such that every edge in the graph connects a vertex in V_1 and a vertex in V_2 (so

that no edge in G connects either two vertices in V_1 or two vertices in V_2). When this condition holds, we call the pair (V_1,V_2) a bipartition of the vertex set V of G.

Theorem 4: A simple graph is bipartite if and only if it is possible to assign one of two different colors to each vertex of the graph so that no two adjacent vertices are assigned the same color.

Complete Bipartite Graph: A complete bipartite graph K_{m,n} is a graph that has its vertex set partitioned into two subsets of m and n vertices, respectively with an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset.

Bipartite graphs can be used to model many types of applications that involve matching the elements of one set to elements of another.

Regular graph: A simple graph is called regular if every vertex of this graph has the same degree. A regular graph is called n-regular if every vertex in this graph has degree n.

Subgraph: 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=G.

Subgraph induced by vertex set: Let G=(V,E) be a simple graph. The subgraph induced by a subset W of the vertex set V is the graph (W,F), where the edge set F contains an edge in E if and only if both endpoints of this edge are in W.

Spanning subgraph: H is a spanning subgraph of G if W=V, F\subseteq{E}.

Union of graph: The union of two simple graphs G_1=(V_1,E_1) and G_2=(V_2,E_2) is the simple graph with vertex set V_1\cup{V}_2 and edge set

E_1\cup{E}_2. The union of G_1 and G_2 is denoted by G_1\cup{G}_2.

Representing Graphs and Graph Isomorphism

Adjacency list: Vertex and Adjacent vertices for simple graph, Initial vertex and terminal vertices for directed graph.

Adjacency matrix: A (or A_G).

Incidence matrix: 1 when edge j is incident with vertex i.

Isomorphism: 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 f is called an isomorphism. Two simple graphs that are not isomorphic are called nonisomorphic.

Graph invariant: A property preserved by isomorphism of graphs is called a graph invariant.

Graph invariants include: The number of vertices, the number of edges, *the number of vertices of each degree* (useful), bipartite, complete, wheel.

Can also check isomorphism by making a function that maps vertices and checking whether it is preserving edges using adjacent matrix.

Connectivity

Path: A sequence of edges that begins at a vertex of a graph and travels

from vertex to vertex along edges of the graph. When there are no multiple edges, the path can be denoted by its vertex sequence.

Circuit: The path is a circuit if it begins and ends at the same vertex, and has length greater than zero.

Pass through and traverse: The path or circuit is said to pass through the vertices in between or traverse the edges.

Simple A path or circuit is simple if it does not contain the same edge more than once.

Connected: An undirected graph is called connected if there is a path between every pair of distinct vertices of the graph. An undirected graph that is not connected is called disconnected.

Theorem 1: There is a simple path between every pair of distinct vertices of a connected undirected graph.

Connected component: A maximal connected subgraph of a graph.

Cut vertex: A vertex is a cut vertex (or articulation point), if removing it and all edges incident with it results in more connected components than in the original graph.

Cut edge: If removal of an edge creates more components, the edge is called a cut edge or bridge.

Strongly connected: A directed graph is strongly connected if there is a path from a to b and from b to a whenever a and b are vertices in the graph.

Weakly connected: A directed graph is weakly connected if there is a path between every two vertices in the underlying undirected graph.

Any strongly connected directed graph is also weakly connected.

Strongly connected component: A maximal strongly connected subgraph, is called a strongly connected component or strong component.

Two graphs are isomorphic only if they have simple circuits of the same length.

Two graphs are isomorphic only if they contain paths that go through vertices so that the corresponding vertices in the two graphs have the same degree.

Theorem 2: Let G be a graph with adjacency matrix A with respect to the ordering v_1,v_2,...,v_n of the vertices of the graph (with directed or undirected edges, with multiple edges and loops allowed). The number of different paths of length r from v_i to v_j, where r is a positive integer, equals the (i,j)th entry of A^r.

The graph G is connected if and only if every off-diagonal entry of A+A^2+A^3+...+A^{n−1} is positive. The check can end earlier if an A^i is found to be so.

Euler and Hamilton Paths

Euler circuit: A simple circuit containing every edge of graph G.

Euler path: A simple path containing every edge of graph G.

Theorem 1: A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has even degree.

Algorithm 1: Constructing Euler Circuits.

```
procedure Euler(G: connected multigraph with all vert
    even degree)
```

```
circuit := a circuit in G beginning at an arbitrarily
    vertex with edges successively added to form a pa
    returns to this vertex
H := G with the edges of this circuit removed
while H has edges
    subcircuit := a circuit in H beginning at a verte
        also is an endpoint of an edge of circuit
    H := H with edges of subcircuit and all isolated
        removed
    circuit := circuit with subcircuit inserted at th
        vertex
return circuit {circuit is an Euler circuit}
```

Theorem 2: A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.

Hamilton path: A simple path in a graph G that passes through every vertex exactly once.

Hamilton circuit: A simple circuit in a graph G that passes through every vertex exactly once.

- A graph with a vertex of degree one cannot have a Hamilton circuit.
- If a vertex in the graph has degree two, then both edges that are incident with this vertex must be part of any Hamilton circuit.
- When a Hamilton circuit is being constructed and this circuit has passed through a vertex, then all remaining edges incident with this vertex, other than the two used in the circuit, can be removed from consideration.
- A Hamilton circuit cannot contain a smaller circuit within it.

Dirac’ s theorem: If G is a simple graph with n vertices with n\geq3 such that the degree of every vertex in G is at least \frac{n}2, then G has a Hamilton circuit.

Ore’s theorem: If G is a simple graph with n vertices with n\geq3 such that deg(u)+deg(v)\geq{n} for every pair of nonadjacent vertices u and v in G, then G has a Hamilton circuit.

Finding Gray code is equivalent to finding a Hamilton circuit for n-cube.

Shortest-Path Problems

Algorithm 1: Dijkstra’s Algorithm

```
procedure Dijkstra(G: weighted connected simple graph
    all weights positive)
{G has vertices a = v_0, v_1, ..., v_n = z and length
    where w(v i , v j ) = ∞ if {v i , v j } is not an
for i := 1 to n
    L(v_i) := ∞
L(a) := 0
S := ∅
{the labels are now initialized so that the label of
    other labels are ∞, and S is the empty set}
while z ∉ S
    u := a vertex not in S with L(u) minimal
    S := S ∪ {u}
    for all vertices v not in S
        if L(u) + w(u, v) < L(v) then L(v) := L(u) +
        {this adds a vertex to S with minimal label a
        labels of vertices not in S}
return L(z) {L(z) = length of a shortest path from a
```

Theorem 1: Dijkstra’s algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph.

Theorem 2: Dijkstra’s algorithm uses O(n^2) operations (additions and comparisons) to find the length of a shortest path between two vertices in a connected simple undirected weighted graph with n vertices.

Traveling salesperson problem: The circuit of minimum total weight in a

weighted, complete, undirected graph that visits each vertex exactly once and returns to its starting point. This is equivalent to asking for a Hamilton circuit with minimum total weight in the complete graph, because each vertex is visited exactly once in the circuit.

Planar Graphs

Planar: A graph is called planar if it can be drawn in the plane without any edges crossing (where a crossing of edges is the intersection of the lines or arcs representing them at a point other than their common endpoint). Such a drawing is called a planar representation of the graph.

Proving no planar representation: Find a loop, divide the plane into regions, divide and conquer.

$K_{3,3}$ and K_5 are non-planar.

Euler’ s formula: Let G be a connected planar simple graph with e edges and v vertices. Let r be the number of regions in a planar representation of G. Then $r=e-v+2$.

Proved by mathematical induction.

Corollary 1: If G is a connected planar simple graph with e edges and v vertices, where $v\geq3$, then $e\leq3v-6$.

Can be used to show that a graph is non-planar.

Degree of a region: the number of edges on the boundary of this region.

Proved by $2e\geq3r$ and Euler’ s formula.

Corollary 2: If G is a connected planar simple graph, then G has a vertex of degree not exceeding five.

Corollary 3: If a connected planar simple graph has e edges and v vertices with $v\geq3$ and no circuits of length three, then $e\leq2v-4$.

Proved like corollary 1, where $2e\geq4r$.

Can be used to show that a graph is non-planar.

Elementary subdivision: If a graph is planar, so will be any graph obtained by removing an edge $\{u,v\}$ and adding a new vertex w together with edges $\{u,w\}$ and $\{w,v\}$. Such an operation is called an elementary subdivision.

Homeomorphic: The graphs $G_1=(V_1,E_1)$ and $G_2=(V_2,E_2)$ are called homeomorphic if they can be obtained from the same graph by a sequence of elementary subdivisions.

Kuratowski’s Theorem: A graph is nonplanar if and only if it contains a subgraph (deleting vertices and incident edges) homeomorphic to $K_{3,3}$ or K_5.

$K_{3,3}$ can also be a hexagon with opposing vertices connected, and the parts are the two sets of three unconnected vertices.

Graph Coloring

Dual graph: Each map in the plane can be represented by a graph. To set up this correspondence, each region of the map is represented by a vertex. Edges connect two vertices if the regions represented by these vertices have a common border. Two regions that touch at only one point are not considered adjacent. The resulting graph is called the dual graph of the map.

Any map in the plane has a planar dual graph.

Coloring: A coloring of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.

Chromatic number: The chromatic number of a graph is the least number of colors needed for a coloring of this graph, denoted by $\chi(G)$.

The four color theorem: The chromatic number of a planar graph is no greater than four.

Nonplanar graphs can have arbitrarily large chromatic numbers.

Show that the chromatic number of a graph is k:

1. Show that the graph can be colored with k colors. This can be done by constructing such a coloring.
2. Show that the graph cannot be colored using fewer than k colors, when 3 it is often shown by a three vertices loop.

The chromatic number of a complete graph K_n is n because every vertex is connected with all others, and this does not contradict the four color theorem because K_n is not planar when $n>4$.

The chromatic number of a complete bipartite graph $K_{m,n}$ is 2, by coloring either set a color.

The chromatic number of a cycle graph C_n, is 1 when $n=1$, 2 when n is even, 3 when n is odd and $n>1$.

Equivalent to scheduling and required number of time slots.

Trees

Introduction to Trees

Tree: A tree is a connected undirected graph with no simple circuits.

Theorem 1: An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.

Rooted tree: A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

Suppose that T is a rooted tree, v is a vertex in T other than the root.

Parent: The parent of v is the unique vertex u such that there is a directed edge from u to v.

Child: When u is the parent of v, v is called a child of u.

Sibling: Vertices with the same parent are called siblings.

Ancestor: The ancestors of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root (that is, its parent, its parent’ s parent, and so on, until the root is reached).

Descendant: The descendants of a vertex v are those vertices that have v as an ancestor.

Leaf: A vertex of a rooted tree is called a leaf if it has no children.

Internal vertex: Vertices that have children are called internal vertices. The root is an internal vertex unless it is the only vertex in the graph, in which case it is a leaf.

Subtree: If a is a vertex in a tree, the subtree with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

m-ary tree: A rooted tree is called an m-ary tree if every internal vertex has no more than m children.

Full m-ary tree: An m-ary tree is called a full m-ary tree if every internal vertex has exactly m children.

Binary tree: An m-ary tree with $m=2$ is called a binary tree.

Ordered rooted trees: An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered. So in (ordered) binary tree defines left child, right child, left subtree, right subtree.

Theorem 2: A tree with n vertices has $n-1$ edges.

Theorem 3: A full m-ary tree with i internal vertices contains $n=mi+1$ vertices.

$n=l+i$

Theorem 4: A full m-ary tree with

1. n vertices has $i=(n-1)/m$ internal vertices and $l=[(m-1)n+1]/m$ leaves.
2. i internal vertices has $n=mi+1$ vertices and $l=(m-1)i+1$ leaves.
3. l leaves has $n=(ml-1)/(m-1)$ vertices and $i=(l-1)/(m-1)$ internal vertices.

Level: The level of a vertex v in a rooted tree is the length of the unique path from the root to this vertex. The level of the root is defined to be zero.

Height: The height of a rooted tree is the maximum of the levels of vertices. In other words, the height of a rooted tree is the length of the longest path from the root to any vertex.

Balanced: A rooted m-ary tree of height h is balanced if all leaves are at levels h or $h-1$.

Theorem 5: There are at most m^h leaves in an m-ary tree of height h.

Corollary 1: If an m-ary tree of height h has l leaves, then h≥ \lceil\log_ml\rceil. If the m-ary tree is full and balanced, then h=\lceil\log_ml\rceil.

Complete m-ary tree: A complete m-ary tree is a full m-ary tree in which every leaf is at the same level.

Applications of Trees

Binary Search Trees: A binary tree in which each child of a vertex is designated as a right or left child, no vertex has more than one right child or left child, and each vertex is labeled with a key, which is one of the items. Furthermore, vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.

Algorithm 1: Locating an Item in or Adding an Item to a Binary Search Tree.

```
procedure insertion(T : binary search tree, x: item)
v := root of T
{a vertex not present in T has the value null }
while v ≠ null and label(v) ≠ x
    if x < label(v) then
        if left child of v ≠ null then v := left chil
        else add new vertex as a left child of v and
    else
        if right child of v ≠ null then v := right ch
    else add new vertex as a right child of v and set
if root of T = null then add a vertex v to the tree a
else if v is null or label(v) ≠ x then label new vert
return v {v = location of x}
```

It is necessary to perform at least \lceil\log(n + 1)\rceil comparisons to add some item.

Decision tree: A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a decision tree.

Theorem 1: A sorting algorithm based on binary comparisons requires at least \lceil\log{(n!)}\rceil comparisons.

Corollary 1: The number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is \Omega(n\log{n}).

\lceil\log{(n!)}\rceil is \Theta(n\log{n}).

Corollary 2: The average number of comparisons used by a sorting algorithm to sort n elements based on binary comparisons is \Omega(n\log{n}).

Prefix code: Code that the bit string for a letter never occurs as the first part of the bit string for another letter.

Algorithm 2: Huffman Coding

```
procedure Huffman(C: symbols a i with frequencies w_i
F := forest of n rooted trees, each consisting of the
while F is not a tree
    Replace the rooted trees T and T' of least weight
    Assign w(T) + w(T') as the weight of the new tree
{the Huffman coding for the symbol a i is the concate
```

// Game tree is not mentioned in courseware.

Game trees: The vertices of these trees represent the positions that a game can be in as it progresses; the edges represent legal moves between these positions. Because game trees are usually large, we

simplify game trees by representing all symmetric positions of a game by the same vertex. However, the same position of a game may be represented by different vertices if different sequences of moves lead to this position. The root represents the starting position. The usual convention is to represent vertices at even levels (starting by 0) by boxes and vertices at odd levels by circles. When the game is in a position represented by a vertex at an even level, it is the first player's move; when the game is in a position represented by a vertex at an odd level, it is the second player's move. Game trees may be infinite when the games they represent never end, such as games that can enter infinite loops, but for most games there are rules that lead to finite game trees. The leaves of a game tree represent the final positions of a game. We assign a value to each leaf indicating the payoff to the first player if the game terminates in the position represented by this leaf. For games that are win–lose, we label a terminal vertex represented by a circle with a 1 to indicate a win by the first player and we label a terminal vertex represented by a box with a −1 to indicate a win by the second player. For games where draws are allowed, we label a terminal vertex corresponding to a draw position with a 0. Note that for win–lose games, we have assigned values to terminal vertices so that the larger the value, the better the outcome for the first player.

The value of a vertex in a game tree is defined recursively as:

1. The value of a leaf is the payoff to the first player when the game terminates in the position represented by this leaf.
2. The value of an internal vertex at an even level is the maximum of the values of its children, and the value of an internal vertex at an odd level is the minimum of the values of its children.

Minmax strategy: The strategy where the first player moves to a position represented by a child with maximum value and the second player moves to a position of a child with minimum value is called the minmax strategy.

Theorem 3: The value of a vertex of a game tree tells us the payoff to the first player if both players follow the minmax strategy and play starts from the position represented by this vertex.

Tree Traversal

Universal address system:

1. Label the root with the integer 0. Then label its k children (at level 1) from left to right with 1,2,3,...,k.
2. For each vertex v at level n with label A, label its k_v children, as they are drawn from left to right, with A.1,A.2,...,A.k_v.

Traversal algorithm: A procedure for systematically visiting every vertex of an ordered rooted tree.

Preorder traversal: Let T be an ordered rooted tree with root r. If T consists only of r, then r is the preorder traversal of T . Otherwise, suppose that T_1, T_2, ⋯, T_n are the subtrees at r from left to right in T. The preorder traversal begins by visiting r. It continues by traversing T_1 in preorder, then T_2 in preorder, and so on, until T_n is traversed in preorder.

Algorithm 1: Preorder Traversal

```
procedure preorder(T : ordered rooted tree)
r := root of T
list r
```

```
    for each child c of r from left to right
        T (c) := subtree with c as its root
        preorder(T (c))
```

Inorder traversal: Let T be an ordered rooted tree with root r. If T consists only of r, then r is the inorder traversal of T. Otherwise, suppose that T_1, T_2, ⋯, T_n are the subtrees at r from left to right. The inorder traversal begins by traversing T_1 in inorder, then visiting r. It continues by traversing T_2 in inorder, then T_3 in inorder, ⋯ , and finally T_n in inorder.

Algorithm 2: Inorder Traversal

```
procedure inorder(T : ordered rooted tree)
r := root of T
if r is a leaf then list r
else
    l := first child of r from left to right
    T (l) := subtree with l as its root
    inorder(T (l))
    list r
    for each child c of r except for l from left to r
        T (c) := subtree with c as its root
        inorder(T (c))
```

Postorder traversal: Let T be an ordered rooted tree with root r. If T consists only of r, then r is the postorder traversal of T. Otherwise, suppose that T_1, T_2, ⋯, T_n are the subtrees at r from left to right. The postorder traversal begins by traversing T_1 in postorder, then T_2 in postorder, ⋯, then T_n in postorder, and ends by visiting r.

Algorithm 3: Postorder Traversal

```
procedure postorder(T : ordered rooted tree)
r := root of T
for each child c of r from left to right
```

```
T (c) := subtree with c as its root
    postorder(T (c))
list r
```

Infix form: The fully parenthesized expression obtained by inorder traversal is said to be in infix form.

By saying fully parenthesized, all the possible parentheses should be added, including the out most one covering the whole expression.

Prefix form:We obtain the prefix form of an expression when we traverse its rooted tree in preorder. Expressions written in prefix form are said to be in Polish notation.

Evaluating prefix form: Work from right to left. When we encounter an operator, we perform the corresponding operation with the two operands immediately to the right of this operand. Also, whenever an operation is performed, we consider the result a new operand.

Postfix form: We obtain the postfix form of an expression by traversing its binary tree in postorder. Expressions written in postfix form are said to be in reverse Polish notation.

Evaluating postfix form: Start at the left and carry out operations when two operands are followed by an operator.

Expressions in prefix or postfix form is unambiguous, so parentheses are not needed.

Spanning Trees

Spanning tree: Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.

Theorem 1: A simple graph is connected if and only if it has a spanning tree.

We can build a spanning tree for a connected simple graph using depth-first search.

Algorithm 1: Depth-First Search

```
procedure DFS(G: connected graph with vertices v_1, v
T := tree consisting only of the vertex v 1
visit(v_1)

procedure visit(v: vertex of G)
for each vertex w adjacent to v and not yet in T
    add vertex w and edge {v, w} to T
    visit(w)
```

Backtracking: Depth-first search is also called backtracking, because the algorithm returns to vertices previously visited to add paths.

Tree edge: The edges selected by depth-first search of a graph are called tree edges.

Back edge: All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree. These edges are called back edges.

DFS constructs a spanning tree using O(e), or O(n^2), steps where e and n are the number of edges and vertices in G, respectively.

Algorithm 2: Breadth-First Search

```
procedure BFS (G: connected graph with vertices v_1,
T := tree consisting only of vertex v 1
L := empty list
put v_1 in the list L of unprocessed vertices
while L is not empty
    remove the first vertex, v, from L
```

```
        for each neighbor w of v
            if w is not in L and not in T then
            add w to the end of the list L
            add w and edge {v, w} to T
```

Applications of backtracking scheme:

- Graph coloring
- The n-queens problem
- Sum of subsets

Depth-first search in directed graphs: Form as spanning forest.

Minimum Spanning Trees

Minimum spanning tree: A minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

Algorithm 1: Prim's Algorithm

```
procedure Prim(G: weighted connected undirected graph
T := a minimum-weight edge
for i := 1 to n - 2
    e := an edge of minimum weight incident to a vert
        simple circuit in T if added to T
    T := T with e added
return T {T is a minimum spanning tree of G}
```

Algorithm 2: Kruskal's Algorithm

```
procedure Kruskal(G: weighted connected undirected gr
T := empty graph
for i := 1 to n - 1
    e := any edge in G with smallest weight that does
        when added to T
```

```
    T := T with e added
return T {T is a minimum spanning tree of G}
```

In Kruskal’s algorithm edges added don’t need to be incident to a vertex already in the tree.

To find a minimum spanning tree of a graph with m edges and n vertices, Kruskal’s algorithm can be carried out using $O(m\log{m})$ operations and Prim’s algorithm can be carried out using $O(m\log{n})$ operations. Consequently, it is preferable to use Kruskal’s algorithm for graphs that are sparse (where m is very small compared to $C(n,2)=n(n-1)/2$).