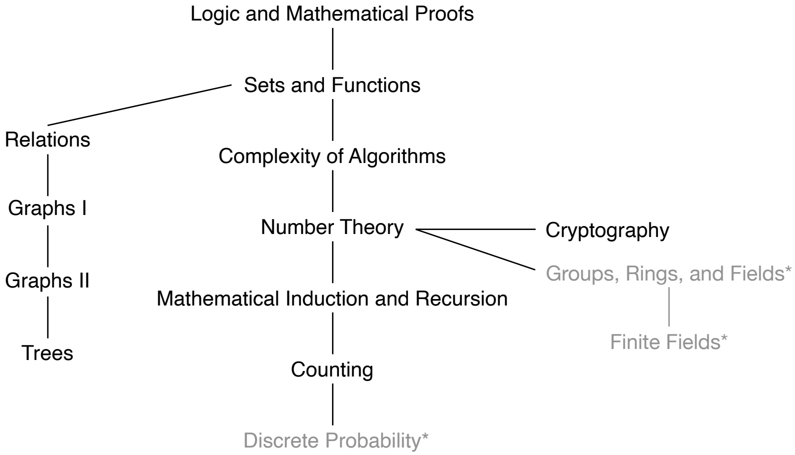
Discrete Mathematics for Computer Science

Lecture 21: Tree

Dr. Ming Tang

Department of Computer Science and Engineering
Southern University of Science and Technology (SUSTech)
Email: tangm3@sustech.edu.cn

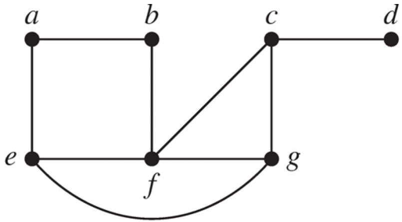
This Lecture



Tree, Tree Traversal, Spanning Trees ...

Spanning Trees

Definition: 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.



Remove edges to avoid circuits.

Spanning Trees

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

Proof:

- only if: The spanning tree can be obtained by removing edges from simple circuits.
- if: The spanning tree T contains every vertex of G. Furthermore, there is a path in T between any two of its vertices. Because T is a subgraph of G, there is a path in G between any two of its vertices. Hence, G is connected.

Depth-First Search

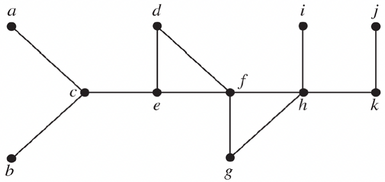
We can find spanning trees by removing edges from simple circuits.

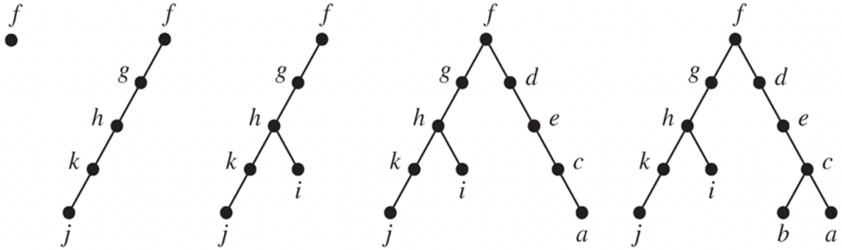
But, this is inefficient, since simple circuits should be identified first.

Instead, we build up spanning trees by successively adding edges.

- First, arbitrarily choose a vertex of the graph as the root.
- Form a path by successively adding vertices and edges. Continue adding to this path as long as possible.
- If the path goes through all vertices of the graph, the tree is a spanning tree.
- Otherwise, move back to some vertex to repeat this procedure (backtracking).

Depth-First Search: Example





Depth-First Search: Algorithm

When we add an edge connecting a vertex v to a vertex w, we finish exploring from w before we return to v to complete exploring from v.

procedure *DFS*(G: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of the vertex v_1
visit(v_1)

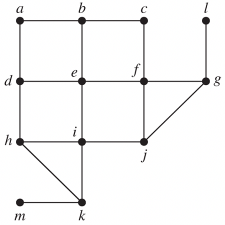
procedure *visit*(v: vertex of G)
for each vertex w adjacent to v and not yet in T
  add vertex w and edge $\{v, w\}$ to T
  visit(w)

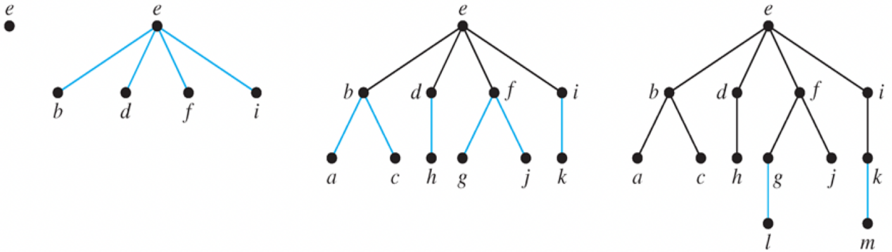
Breadth-First Search

This is the second algorithm that we build up spanning trees by successively adding edges.

- First arbitrarily choose a vertex of the graph as the root.
- Form a path by adding all edges incident to this vertex and the other endpoint of each of these edges
- For each vertex added at the previous level, add edge incident to this vertex, as long as it does not produce a simple circuit.
- Continue in this manner until all vertices have been added.

Breadth-First Search: Example





Breadth-First Search

```
procedure BFS(G: connected graph with vertices v1, v2, ..., vn)
T := tree consisting only of the vertex v1
L := empty list  visit(v1)
put v1 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
```

Backtracking Applications

There are problems that can be solved only by performing an exhaustive search of all possible solutions.

One way to search systematically for a solution is to use a decision tree, where each internal vertex represents a decision and each leaf a possible solution.

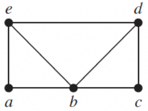
To find a solution via backtracking

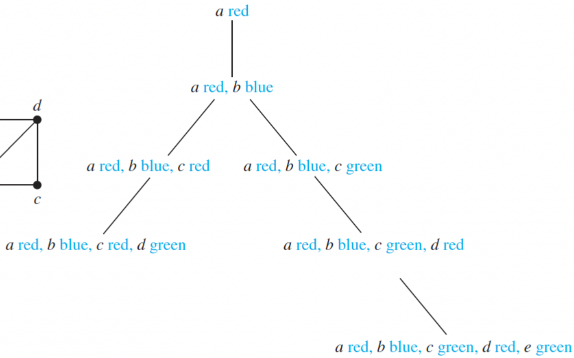
- first make a sequence of decisions in an attempt to reach a solution as long as this is possible.
- Once it is known that no solution can result from any further sequence of decisions, backtrack to the parent of the current vertex and work toward a solution with another series of decisions

The procedure continues until a solution is found, or it is established that no solution exists.

Backtracking Applications: Graph Colorings

How can backtracking be used to decide whether a graph can be colored using n colors?

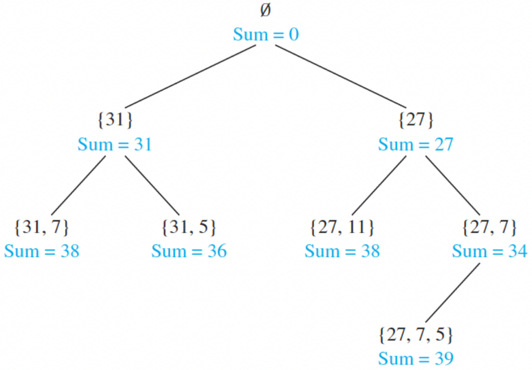




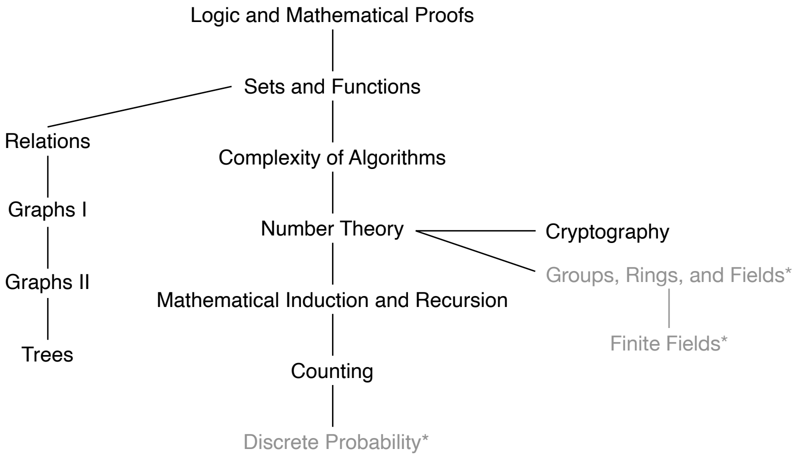
Backtracking Applications: Sums of Subsets

Consider this problem. Given a set of positive integers $x_1, x_2, \ldots, x_n$, find a subset of this set of integers that has M as its sum. How can backtracking be used to solve this problem?

Finding a subset of $\{31, 27, 15, 11, 7, 5\}$ with the sum equal to 39.



This Lecture



Tree, Tree Traversal, Spanning Trees, Minimum Spanning Trees, ...

Minimum Spanning Trees

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

Two greedy algorithms: Prim's Algorithm, Kruscal's Algorithm.

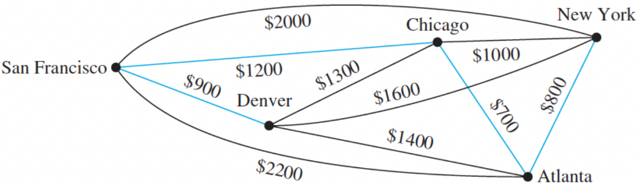
Both algorithms do produce optimal solutions.

Prim's Algorithm

ALGORITHM 1 Prim's Algorithm.

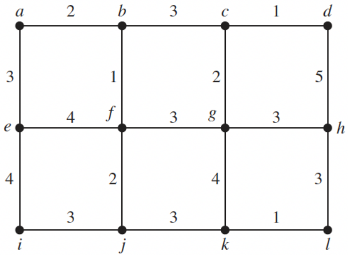
procedure *Prim*(G: weighted connected undirected graph with n vertices)
T := a minimum-weight edge
for $i := 1$ **to** $n - 2$
  e := an edge of minimum weight incident to a vertex in T and not forming a
    simple circuit in T if added to T
  $T := T$ with e added
return T {T is a minimum spanning tree of G}

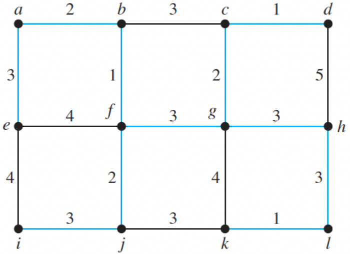
Prim's Algorithm: Example



Choice	Edge	Cost
1	{Chicago, Atlanta}	\$ 700
2	{Atlanta, New York}	\$ 800
3	{Chicago, San Francisco}	\$1200
4	{San Francisco, Denver}	\$ 900
	Total:	\$3600

Prim's Algorithm: Example





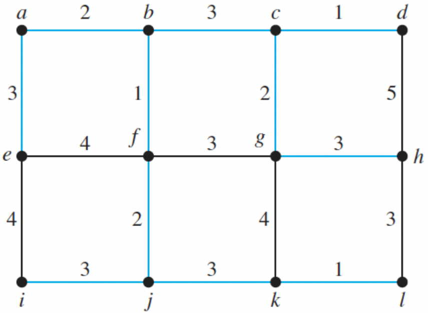
Choice	Edge	Weight
1	$\{b, f\}$	1
2	$\{a, b\}$	2
3	$\{f, j\}$	2
4	$\{a, e\}$	3
5	$\{i, j\}$	3
6	$\{f, g\}$	3
7	$\{c, g\}$	2
8	$\{c, d\}$	1
9	$\{g, h\}$	3
10	$\{h, l\}$	3
11	$\{k, l\}$	1
	Total:	24

Kruskal's Algorithm

ALGORITHM 2 Kruskal's Algorithm.

procedure *Kruskal*(G: weighted connected undirected graph with n vertices)
T := empty graph
for $i := 1$ **to** $n - 1$
  e := any edge in G with smallest weight that does not form a simple circuit
    when added to T
  $T := T$ with e added
return T {T is a minimum spanning tree of G}

Kruskal's Algorithm: Example



Choice	Edge	Weight
1	$\{c, d\}$	1
2	$\{k, l\}$	1
3	$\{b, f\}$	1
4	$\{c, g\}$	2
5	$\{a, b\}$	2
6	$\{f, j\}$	2
7	$\{b, c\}$	3
8	$\{j, k\}$	3
9	$\{g, h\}$	3
10	$\{i, j\}$	3
11	$\{a, e\}$	3
	Total:	24

Topics of This Course

- Logic and Mathematical Proofs
  - Sets and Functions
    - Relations
      - Graphs I
        - Graphs II
          - Trees
    - Complexity of Algorithms
      - Number Theory
        - Cryptography
        - Groups, Rings, and Fields*
          - Finite Fields*
        - Mathematical Induction and Recursion
          - Counting
            - Discrete Probability*

Lecture Schedule

1. Logic and Mathematical Proofs
2. Sets and Functions
3. Complexity of Algorithms
4. Number Theory and Cryptography
5. Mathematical Induction
6. Recursion
7. Counting
8. Relations
9. Graph
10. Trees

SUSTech Southern University of Science and Technology

Lecture Schedule

Propositional Logic

Proposition: a declarative sentence that is either true or false (not both).

- Conventional letters used for propositional variables are p, q, r, s, ...
- **Truth value** of a proposition: true, denoted by T; false, denoted by F.

Compound propositions are build using logical connectives:

- Negation $\neg$
- Conjunction $\wedge$
- Disjunction $\vee$
- Exclusive or $\oplus$
- Implication $\rightarrow$
- Biconditional $\leftrightarrow$

SUSTech Southern University of Science and Technology

Tautology and Logical Equivalences

- Tautology: A compound proposition that is always true, no matter what the truth values of the propositional variables that occur in it.
  - E.g., $p \vee \neg p$
- Contradiction: A compound proposition that is always false.

The compound propositions p and q are called logically equivalent, denoted by $p \equiv q$, if $p \leftrightarrow q$ is a tautology.

- E.g., $\neg(p \vee q)$ and $\neg p \wedge \neg q$

That is, two compound propositions are equivalent if they always have the same truth value.

Determine logically equivalent propositions using:

- Truth table
- Logical Equivalences

Important Logical Equivalences

Equivalence	***Name***
$p \wedge \mathbf{T} \equiv p$ $p \vee \mathbf{F} \equiv p$	Identity laws
$p \vee \mathbf{T} \equiv \mathbf{T}$ $p \wedge \mathbf{F} \equiv \mathbf{F}$	Domination laws
$p \vee p \equiv p$ $p \wedge p \equiv p$	Idempotent laws
$\neg(\neg p) \equiv p$	Double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	Commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive laws
$\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan's laws
$p \vee (p \wedge q) \equiv p$	Absorption laws

Predicate Logic and Quantified Statements

Predicate Logic: make statements with variables: $P(x)$.

Propositional function $P(x) \overset{\text{specify } x}{\Longrightarrow}$ Proposition

Quantified Statements: Universal quantifier $\forall x P(x)$; Existential quantifier $\exists x P(x)$

Statement	When true?	When false?
∀x P(x)	P(x) true for all x	There is an x where P(x) is false.
∃x P(x)	There is some x for which P(x) is true.	P(x) is false for all x.

Propositional function $P(x) \overset{\text{for all/some } x \text{ in domain}}{\Longrightarrow}$ Proposition

Negation and Nest Quantifier

Negation	Equivalent Statement	When Is Negation True?	When False?
$\neg \exists x\ P(x)$	$\forall x\ \neg P(x)$	For every x, $P(x)$ is false.	There is an x for which $P(x)$ is true.
$\neg \forall x\ P(x)$	$\exists x\ \neg P(x)$	There is an x for which $P(x)$ is false.	$P(x)$ is true for every x.

Statement	***When True?***	***When False?***
$\forall x \forall y P(x, y)$ $\forall y \forall x P(x, y)$	$P(x, y)$ is true for every pair x, y.	There is a pair x, y for which $P(x, y)$ is false.
$\forall x \exists y P(x, y)$	For every x there is a y for which $P(x, y)$ is true.	There is an x such that $P(x, y)$ is false for every y.
$\exists x \forall y P(x, y)$	There is an x for which $P(x, y)$ is true for every y.	For every x there is a y for which $P(x, y)$ is false.
$\exists x \exists y P(x, y)$ $\exists y \exists x P(x, y)$	There is a pair x, y for which $P(x, y)$ is true.	$P(x, y)$ is false for every pair x, y.

Southern University of Science and Technology

Validity of Argument Form:

The argument form with premises $p_1, p_2, ..., p_n$ and conclusion q is valid, if

$$(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow q \text{ is a tautology.}$$

Note: According to the definition of $p \rightarrow q$, we do not worry about the case where $p_1 \wedge p_2 \wedge \cdots \wedge p_n$ is false.

Rules of Inference for Propositional Logic

Rule of Inference	***Tautology***	***Name***
p $p \to q$ $\therefore q$	$(p \wedge (p \to q)) \to q$	Modus ponens
$\neg q$ $p \to q$ $\therefore \neg p$	$(\neg q \wedge (p \to q)) \to \neg p$	Modus tollens
$p \to q$ $q \to r$ $\therefore p \to r$	$((p \to q) \wedge (q \to r)) \to (p \to r)$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore q$	$((p \vee q) \wedge \neg p) \to q$	Disjunctive syllogism
p $\therefore p \vee q$	$p \to (p \vee q)$	Addition
$p \wedge q$ $\therefore p$	$(p \wedge q) \to p$	Simplification
p q $\therefore p \wedge q$	$((p) \wedge (q)) \to (p \wedge q)$	Conjunction

Methods of Proving Theorems

A proof is a valid argument that establishes the truth of a mathematical statement.

- **Direct proof**

  $p \to q$ is proved by showing that if p is true then q follows
- **Proof by contrapositive**

  show the contrapositive $\neg q \to \neg p$
- **Proof by contradiction**

  show that $(p \wedge \neg q)$ contradicts the assumptions
- **Proof by cases**

  give proofs for all possible cases
- **Proof of equivalence**

  $p \leftrightarrow q$ is replaced with $(p \to q) \wedge (q \leftarrow p)$

Proof Exercises

Prove that $\sqrt{2}$ is irrational. (Rational numbers are those of the form $\frac{m}{n}$, where m and n are integers.)

Prove that there are infinitely many prime numbers.

Show that there exist irrational numbers x and y such that x^y is rational.

Lecture Schedule

1 Logic and Mathematical Proofs

2 Sets and Functions

3 Complexity of Algorithms

4 Number Theory and Cryptography

5 Mathematical Induction

6 Recursion

7 Counting

8 Relations

9 Graph

10 Trees

Sets

A set is an unordered collection of objects.

- listing (enumerating) the elements
- if enumeration is hard, use ellipses (...)
- definition by property, using the set builder

$$\{x \mid x \text{ has property } P \text{ or property } P(x))\}$$

Proof of Subset:

- Showing $A \subseteq B$: if x belongs to A, then x also belongs to B.
- Showing $A \nsubseteq B$: find a single $x \in A$ such that $x \notin B$.

Prove $A = B$?

Cardinality, Power Set, Tuples, and Cartesian Product

Cardinality: If there are exactly n distinct elements in S, where n is a nonnegative integer, we say that S is a finite set and n is the cardinality of S, denoted by $|S|$.

Power Set: Given a set S, the power set of S is the set of all subsets of the set S, denoted by $\mathcal{P}(S)$.

Tuples: The ordered n-tuple $(a_1, a_2, ..., a_n)$ is the ordered collection that has a_1 as its first element and a_2 as its second element and so on.

Cartesian Product: Let A and B be sets. The Cartesian product of A and B, denoted by $A \times B$, is the set of all ordered pairs (a, b), where $a \in A$ and $b \in B$:

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

Set Operations

Union: Let A and B be sets. The union of the sets A and B, denoted by $A \cup B$, is the set $\{x \mid x \in A \vee x \in B\}$.

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

Complement: If A is a set, then the complement of the set A (with respect to U), denoted by $\bar{A}$ is the set $U - A$, $\bar{A} = \{x \in U \mid x \notin A\}$

Difference: Let A and B be sets. The difference of A and B, denoted by $A - B$, is the set containing the elements of A that are not in B.
$A - B = \{x \mid x \in A \wedge x \notin B\} = A \cap \bar{B}$.

Principle of inclusion–exclusion: $|A \cup B| = |A| + |B| - |A \cap B|$

Set Identities

$A \cap U = A$ $A \cup \emptyset = A$	Identity laws
$A \cup U = U$ $A \cap \emptyset = \emptyset$	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
$A \cup (A \cap B) = A$	Absorption laws

Proof of Set Identities

Prove that $\overline{A \cap B} = \bar{A} \cup \bar{B}$

Proof 1: using membership tables.
Proof 2: by showing that $\overline{A \cap B} \subseteq \bar{A} \cup \bar{B}$ and $\bar{A} \cup \bar{B} \subseteq \overline{A \cap B}$
Proof 3: Using set builder and logical equivalences

$\overline{A \cap B}$	$= \{x \mid x \notin A \cap B\}$	by definition of complement
	$= \{x \mid \neg(x \in (A \cap B))\}$	by definition of does not belong symbol
	$= \{x \mid \neg(x \in A \wedge x \in B)\}$	by definition of intersection
	$= \{x \mid \neg(x \in A) \vee \neg(x \in B)\}$	by the first De Morgan law for logical equivalences
	$= \{x \mid x \notin A \vee x \notin B\}$	by definition of does not belong symbol
	$= \{x \mid x \in \overline{A} \vee x \in \overline{B}\}$	by definition of complement
	$= \{x \mid x \in \overline{A} \cup \overline{B}\}$	by definition of union
	$= \overline{A} \cup \overline{B}$	by meaning of set builder notation

Function

Let A and B be two sets. A function from A to B, denoted by $f : A \to B$, is an assignment of exactly one element of B to each element of A.

- **One-to-one (injective) function**:
  - A function f is called one-to-one or injective if and only if $f(x) = f(y)$ implies $x = y$ for all x, y in the domain of f.
- **Onto (surjective) function**:
  - A function f is called onto or surjective if and only if for every $b \in B$ there is an element $a \in A$ such that $f(a) = b$.
- **One-to-one (bijective) correspondence**
  - One-to-one and onto

Proof for One-to-One and Onto

Suppose that $f : A \rightarrow B$.

To show that f is *injective*	Show that if $f(x) = f(y)$ for all $x, y \in A$, then $x = y$
To show that f is not *injective*	Find specific 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 specific element $y \in B$ such that $f(x) \neq y$ for all $x \in A$

SUSTech Southern University of Science and Technology

Inverse Function and Composition of Functions

Inverse function: Let f be a one-to-one correspondence (bijection) 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 $f(a) = b$.

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

The floor function assigns a real number x the largest integer that is $\leq x$, denoted by $\lfloor x \rfloor$. E.g., $\lfloor 3.5 \rfloor = 3$.

The ceiling function assigns a real number x the smallest integer that is $\geq x$, denoted by $\lceil x \rceil$. E.g., $\lceil 3.5 \rceil = 4$.

Sequences

A sequence is a function from a subset of the set of integers (typically the set $\{0, 1, 2, \ldots\}$ or $\{1, 2, 3, \ldots\}$) to a set S.

We use the notation a_n to denote the image of the integer n. $\{a_n\}$ represents the ordered list $\{a_1, a_2, a_3, \ldots\}$

Recursively Defined Sequences: provide

- One or more initial terms
- A rule for determining subsequent terms from those that precede them.

Cardinality of Sets

A set that is either finite or has the same cardinality as the set of positive integers $\mathbf{Z}^+$ is called countable.

If there is a one-to-one function from A to B, the cardinality of A is less than or equal to the cardinality of B, denoted by $|A| \leq |B|$.

Theorem: If there is a one-to-one correspondence between elements in A and B, then the sets A and B have the same cardinality.

Theorem: If A and B are sets with $|A| \leq |B|$ and $|B| \leq |A|$, then $|A| = |B|$.

Lecture Schedule

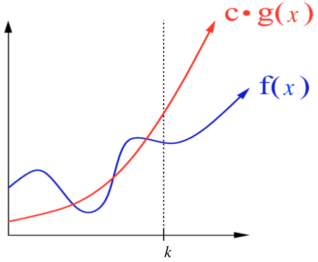
1. Logic and Mathematical Proofs
2. Sets and Functions
3. Complexity of Algorithms
4. Number Theory and Cryptography
5. Mathematical Induction
6. Recursion
7. Counting
8. Relations
9. Graph
10. Trees

Big-O Notation

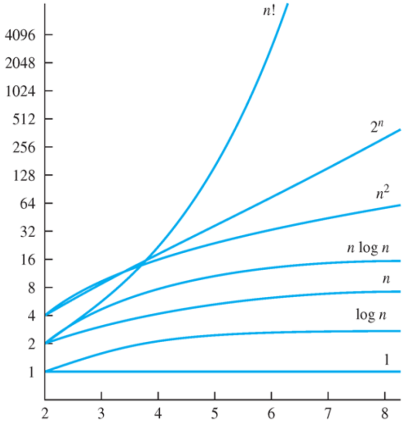
Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers.We say that $f(x)$ is $O(g(x))$ if there are constants C and k such that

$$|f(x)| \leq C|g(x)|,$$

whenever $x > k$. [This is read as "$f(x)$ is big-oh of $g(x)$."]



Big-O Estimates for Some Functions



Big-Omega 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 $\Omega(g(x))$ if there are positive constants C and k such that

$$|f(x)| \geq C|g(x)|$$

whenever $x > k$. [This is read as "$f(x)$ is big-Omega of $g(x)$."]

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 $\Theta(g(x))$ if

- $f(x)$ is $O(g(x))$ and
- $f(x)$ is $\Omega(g(x))$.

When $f(x)$ is $\Theta(g(x))$, we say that $f(x)$ 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.

We will NOT let you compute the complexity of an algorithm.

Lecture Schedule

1. Logic and Mathematical Proofs
2. Sets and Functions
3. Complexity of Algorithms
4. Number Theory and Cryptography
5. Mathematical Induction
6. Recursion
7. Counting
8. Relations
9. Graph
10. Trees

Southern University of Science and Technology