Theoretical Concepts of Computer Science
Review Topics

Last Updated: August 26th, 2024

1 Notation Review

1.1 Logic

Boolean Operations.

Let a and b denote Boolean values (either 0 or 1).

And $a \wedge b$ evaluates to 1 iff $a = b = 1$

Inclusive Or $a \vee b$ evaluates to 1 iff either $a = 1$ or $b = 1$ (or both)

Exclusive Or $a \oplus b$ evaluates to 1 iff either $a = 1$ or $b = 1$ (but not both)

Not $\overline{a}$ evaluates to 1 iff $a = 0$.

Conditional $a \rightarrow b$ evaluates to 1 iff either $a = 0$ or $b = 1$ (or both)

Equivalence $a \leftrightarrow b$ evaluates to 1 iff $a = b$.

Definition 1.1. A **Boolean expression** (equivalently, **Boolean formula**) is any expression whose operations are all Boolean, and whose terminals are either 0, 1, or any Boolean variable.

Example 1.2. Assuming a, b, c are Boolean variables, then

$$(\bar{b} \vee 1) \rightarrow (a \wedge c)$$

is a Boolean expression.

Definition 1.3. Two Boolean expressions p and q are said to be **logically equivalent**, denoted $p \Leftrightarrow q$, iff they depend on the same set of variables and always evaluate to the same Boolean value, regardless of how the variables are assigned.

Common Logical Equivalences

Assume that p, q, and r are Boolean expressions.

Equivalence	**Name**
$p \wedge 1 \Leftrightarrow p$ $p \vee 0 \Leftrightarrow p$	Identity
$p \vee 1 \Leftrightarrow 1$ $p \wedge 0 \Leftrightarrow 0$	Domination
$p \vee p \Leftrightarrow p$ $p \wedge p \Leftrightarrow p$	Idempotency
$\overline{\overline{p}} \Leftrightarrow p$	Double negation
$p \vee q \Leftrightarrow q \vee p$ $p \wedge q \Leftrightarrow q \wedge p$	Commutativity
$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$ $(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	Associativity
$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$	Distributivity
$\overline{(p \vee q)} \Leftrightarrow \overline{p} \wedge \overline{q}$ $\overline{(p \wedge q)} \Leftrightarrow \overline{p} \vee \overline{q}$	De Morgan

1.2 Sets

Definition 1.4. A **set** represents a collection of items, where each item is called a **member** or **element** of the set.

List Notation the most common way to represent a set where the set members are listed one-by-one, and the list is delimited by braces. For example,

$$\{2, 3, 5, 7, 11\}$$

uses list notation to describe the set consisting of all prime numbers that do not exceed 11. Note that the order in which the members are listed does not matter. Indeed the sets $\{2, 3, 5, 7, 11\}$ and $\{3, 11, 5, 2, 7\}$ are identical. Also, each member occurs only *once* in the set, meaning that, e.g. $\{1, 2, 2, 3, 3, 3\} = \{1, 2, 3\}$.

Informal List Notation uses **ellipsis** ... to indicate that a pattern is to be continued in the list, either indefinitely or up to some value.

Common Numerical Sets 1. $\mathbb{N} = \{0, 1, 2, \ldots\}$

2. $\mathbb{I} = \{0, \pm 1, \pm 2, \ldots\} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$
3. $\mathbb{Q} = \{p/q \mid p, q \in \mathbb{I} \wedge q \neq 0\}$

Empty Set the set having no members and denoted by $\emptyset$.

Membership Symbol $x \in A$ indicates that item x is a member of set A.

Containment Symbol $A \subseteq B$ means that A is a **subset** of B. In other words, every member of A is also a member of B. Note: trivially, $\emptyset \subseteq B$ for every set B.

Proper Containment Symbol $A \subset B$ means that A is a **subset** of B but that there is some member of B that is not a member of A. In this case we say that A is a **proper subset** of B.

Set Equality $A = B$ iff $A \subseteq B$ and $B \subseteq A$ are both true statements.

Example 1.5. The following are some examples of informal list notation.

1. The set of prime numbers less than 100 may be written as

$$\{2, 3, 5, 7, 11, \ldots, 97\}.$$

2. $\mathbb{N}$ denotes the set of **natural numbers** $\{0, 1, 2, \ldots\}$.

3. $\mathbb{I}$ denotes the set of integers

$$\{0, \pm 1, \pm 2, \ldots\} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}.$$

Example 1.6. The following are all true statements.

1. $61 \in \{2, 3, 5, 7, 11, \ldots, 97\}$.
2. $39 \notin \{2, 3, 5, 7, 11, \ldots, 97\}$.
3. $\{3\} \in \{\emptyset, \{1\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$
4. $3 \notin \{\emptyset, \{1\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$

$\square$

Example 1.7. Sets $\{3\}$ and $\{1, 3\}$ are proper subsets of $\{1, 2, 3\}$, while $\{1, 2, 3\}$ is a subset of $\{1, 2, 3\}$, but not a proper subset. $\square$

Example 1.8. Neither set $A = \{2, 3, 6, 7\}$ nor set $B = \{2, 3, 5, 7, 11, 13, 17\}$ is a subset of $C = \{2, 3, 5, 7, 11, 13\}$ since $6 \in A$ but $6 \notin C$, and $17 \in B$ but $17 \notin C$.

Example 1.9. $A = \{\{1,2\},\{3,4\},\{3,5\}\}$ is *not* a subset of $B = \{1,2,3,4,5\}$ since, e.g., $\{1,2\} \in A$ but $\{1,2\} \notin B$. This is true since $\{1,2\}$ is a set, and B's members are the natural numbers 1 through 5 which are not sets.

Example 1.10. Given $B = \{2,\{3,7\},3,4,\{5\},\{6,8,9\}\}$, all of the following statements are true.

a. $\{2,4\} \subseteq B$.

b. $\{3,7\} \not\subseteq B$.

c. $9 \notin B$.

d. $\{5\} \in B$.

Set Operations.

Union $x \in A \cup B$ iff either $x \in A$ or $x \in B$ (or both)

Intersection $x \in A \cap B$ iff $x \in A$ and $x \in B$

Complement $x \in \overline{A}$ iff $x \notin A$

Difference $x \in A - B$ iff $x \in A$ and $x \notin B$

Symmetric Difference $x \in A \oplus B$ iff $x \in A$ or $x \in B$, but not both; i.e. $A \oplus B = (A - B) \cup (B - A)$

Cartesian Product $(a, b) \in A \times B$ iff $a \in A$ and $b \in B$

Power Set The **power set** of a set S, denoted $\mathcal{P}(S)$ is the set of all subsets of S. Note that if $|S| = n < \infty$, then $|\mathcal{P}(S)| = 2^n$, since, for each of the n members of S, one has a binary choice as to whether or not to add the member to the subset. This makes a total of

$$\underbrace{2 \times 2 \times \cdots \times 2}_{n \text{ times}} = 2^n$$

different possible subsets.

Example 1.11. Given sets $A = \{1, 3, 5, 6, 7, 9\}$ and $B = \{0, 2, 4, 5, 6, 7, 8, 10\}$, we have $A \cup B = \{0, 1, 2, \ldots, 10\}$, $A \cap B = \{5, 6, 7\}$, $A - B = \{1, 3, 9\}$, and $A \oplus B = \{0, 1, 2, 3, 4, 8, 9, 10\}$.

Example 1.12. Let $\mathbb{E}$ denote the set of all even integers. Compute $\mathbb{N} \cup \mathbb{E}$, $\mathbb{N} \cap \mathbb{E}$, $\mathbb{N} - \mathbb{E}$, $\mathbb{E} - \mathbb{N}$, and $\mathbb{N} \oplus \mathbb{E}$.

Example 1.13. Given sets $A = \{a, b\}$, $B = \{1, 2, 3\}$, then $A \times B = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}$.

Example 1.14. For $S = \{1, 2, 3\}$ we have

$$\mathcal{P}(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

1.3 Functions

Definition 1.15. A **function** $f : A \to B$ is a set A, a set B, and a relationship between the two sets such that, for each $a \in A$, there is a unique $b \in B$ that is associated with a. In this case we write $f(a) = b$.

The following is some terminology and notation that is used to describe functions.

- A is called the **domain** of f.
- B is called the **codomain** of f.
- $f : A \to B$ is a notation that indicates f is a function that has domain A and codomain B.
- $f(x)$ is a common notation that is used when both the domain of x (and hence f) and the codomain of f are already understood.
- $f(a) = b$ indicates that $b \in B$ is the unique member of the codomain that is associated with domain member $a \in A$. Also, a is called a **preimage** of b under f. It is also common to call a the **input** and $f(a) = b$ its assigned **output**.

Like variables, every function has a name, and it is common to use generic names, such as f, g, and h, when speaking of some arbitrary (i.e. no one in particular) function, just as it is common to use names like x, y, and z to represent an arbitrary variable.

Example 1.16. Consider the function grade : Student $\to G$ that assigns student $s \in$ Student a letter grade from the set $G = \{a, b, c, d, f\}$, where

$$\text{Student} = \{\text{Ann}, \text{Ethan}, \text{Jaspinder}, \text{Pam}\}.$$

Thus, Student is the domain of `grade`, and G is its codomain. Finally the student grades assigned by `grade` are grade(Ann) $= a$, grade(Ethan) $= c$, grade(Jaspinder) $= c$, and grade(Pam) $= b$.

Example 1.17. Let S denote the set of students attending a university, I the set of instructors at the university, and s a variable whose domain is S. For a student s attending a university, let ins(s) denote the instructor of the first calculus course that s enrolled in at the university. Then ins : $S \to I$ is not a function because not every student (e.g. a student majoring in music) will elect to take calculus at the university. A function must assign *every* member of the domain to some member of the codomain. □

1.4 Graphs

A **Graph** is a pair of sets V, E, where

Graph $G = (V, E)$ V is a set of **vertices**, also called **nodes**, while E is a set whose members are pairs of vertices and are called **edges**. Each edge may be written as a tuple of the form (u, v), where $u, v \in V$.

Adjacency and Incidence If $e = (u, v)$ is an edge, then we say that u is **adjacent** to v, and that e is **incident** with u and v.

Order $|V| = n$ is called the **order** of G.

Size $|E| = m$ is called the **size** of G.

Path A **path** P of length k in a graph is a sequence of vertices $P = v_0, v_1, \ldots, v_k$, such that $(v_i, v_{i+1}) \in E$ for every $0 \le i \le k-1$.

Simple Path $P = v_0, v_1, \ldots, v_k$ and the vertices $v_0, v_1, \ldots, v_k$ are all distinct.

Cycle A **cycle** is a path that begins and ends at the same vertex.

Geometrical Representation obtained by representing each vertex as a figure (usually a circle) on a two-dimensional plane, and each edge $e = (u, v)$ as a smooth arcs that connects vertex u with vertex v.

Degree The **degree** of a vertex v, denoted as $\deg(v)$, equals the number of edges that are incident with v. Note: loop edges are counted twice.

Example 1.18. Let $G = (V, E)$, where

$$V = \{SD, SB, SF, LA, SJ, OAK\}$$

are cities in California, and

$$E = \{(SD, LA), (SD, SF), (LA, SB), (LA, SF), (LA, SJ), (LA, OAK), (SB, SJ)\}$$

are edges, each of which represents the existence of one or more flights between two cities. Figure 1 shows a graphical representation of G. G has order 6 and size 7.

Figure 2 shows a simple path of length 4. Figure 3 shows a cycle of length 3. Let's verify the Handshaking theorem.

$$\deg(\text{SF}) + \deg(\text{LA}) + \deg(\text{SD}) + \deg(\text{OAK}) + \deg(\text{SJ}) + \deg(\text{SB}) =$$

$$2 + 5 + 2 + 1 + 2 + 2 = 14 = 2 \cdot 7 = 2|E|.$$

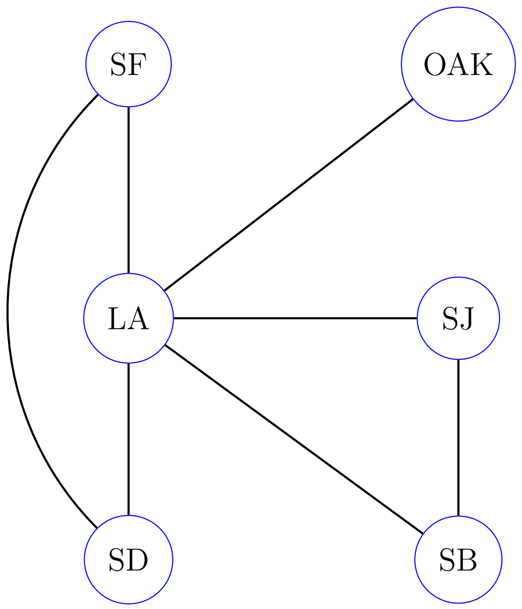

Figure 1: Graphical Representation of G

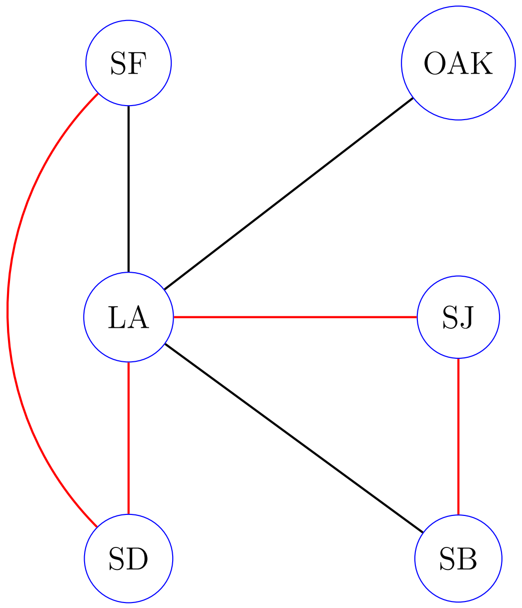

Figure 2: Simple path (in red) $P =$ SF,SD,LA,SJ,SB of length 4

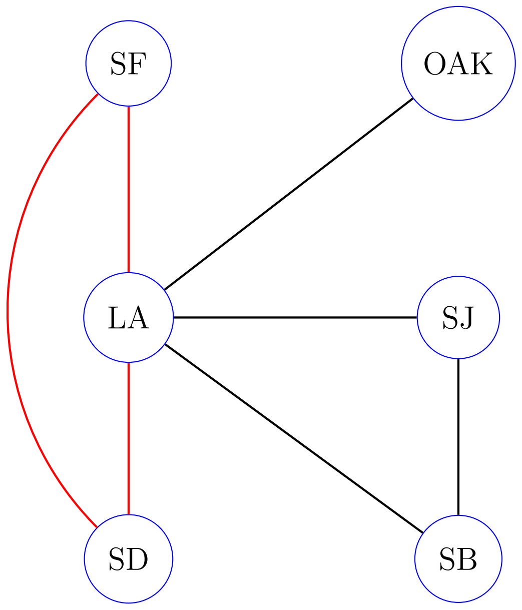

Figure 3: Cycle (in red) $C =$ SF,SD,LA,SF of length 3

1.5 Big-O Notation

Big-O notation is useful for making statements about the growth of a function $f(n)$, n a natural number, whose values may seem difficult or impossible to compute. The statements we care most about are those that state upper and/or lower bounds on f's growth. Although we may not know the exact bounds (since we may not know the exact values of f), we may be able to determine meaningful ones in case we know the following two things:

1. the rule, call it $g(n)$, for the fastest growing term (ignoring constants) of the bounding function, and
2. that there exists a constant $c > 0$ such that, $cg(n)$ provides a bound for f, for sufficiently large n.

Example 1.19. Carol has programmed the `Insertion Sort` algorithm to run on her laptop. What upper bound can she provide on the elapsed time $t(n)$ that will occur on her laptop clock after `Insertion Sort` has sorted an integer array of size n? She knows that the worst case occurs when the input array is sorted in reverse order, and in this case a total of

$$\frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

comparisons and swaps must be performed in order to sort such an array. In this case, the rule for the fastest growing term of the upper-bounding function is $g(n) = n^2$. Also, she knows that each comparison and swap requires at most two machine instructions, and that each machine instruction requires at most 10^{-8} seconds to execute. Therefore, there is a $c > 0$ such that $cg(n)$ is an upper bound for the elapsed time. $\square$

Let $f(n)$ and $g(n)$ be functions from the set of nonnegative integers to the set of nonnegative real numbers. Then

Big-O $f(n) = \mathrm{O}(g(n))$ iff there exist constants $c > 0$ and $k \geq 1$ such that $f(n) \leq cg(n)$ for every $n \geq k$.

Big-Ω $f(n) = \Omega(g(n))$ iff there exist constants $c > 0$ and $k \geq 1$ such that $f(n) \geq cg(n)$ for every $n \geq k$.

Big-Θ $f(n) = \Theta(g(n))$ iff $f(n) = \mathrm{O}(g(n))$ and $f(n) = \Omega(g(n))$.

little-o $f(n) = \mathrm{o}(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

little-ω $f(n) = \omega(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

Definition 1.20. The following table shows the most common kinds of rules for $g(n)$ that are used within big-O notation.

Function	**Type of Growth**
1	constant growth
$\log n$	logarithmic growth
$\log^k n$, for some integer $k \geq 1$	polylogarithmic growth
n^k for some positve $k < 1$	sublinear growth
n	linear growth
$n \log n$	log-linear growth
$n \log^k n$, for some integer $k \geq 1$	polylog-linear growth
$n^j \log^k n$, for some integers $j, k \geq 1$	polylog-polynomial growth
n^2	quadratic growth
n^3	cubic growth
n^k for some integer $k \geq 1$	polynomial growth
$2^{\log^c n}$, for some $c > 1$	quasi-polynomial growth
$\omega(n^k)$, for all integers $k \geq 1$	superpolynomial growth
a^n for some $a > 1$	exponential growth

Example 1.21. Returning to Example 1.19, using big-O notation Carol can say that, when running `Insertion Sort` on her laptop with an input of size n, the elapsed time equals $O(n^2)$ seconds. Also, since `Insertion Sort` requires at least n comparisons for any input, she may also say that its running time equals $\Omega(n)$ seconds. $\square$

Given a problem L, and an algorithm $\mathcal{A}$ that solves L, big-O notation finds its use in the study of algorithms as a means for describing bounds on the number of steps and the amount of memory required by $\mathcal{A}$ as a function of the **size parameters** of L, i.e. the parameters used to indicate the number of bits required to represent an instance of L.

Example 1.22. The following are examples of how big-O notation arises in the study of data structures and algorithms.

1. Inserting an item into a balanced tree of size n requires $O(\log n)$ comparisons.
2. It has been proven that, sorting n numbers using pairwise comparisons requires $\Omega(n \log n)$ comparisions.
3. The Fast Fourier Transform algorithm has a running time of $\Theta(m \log m)$, where m is the degree of the input polynomial.
4. Two b-bit integers may be recursively added/subtracted in $O(b)$ steps and recursively multiplied/divided in $O(b^2)$.

It is also common to see big-O notation used within an arithmetic expression, such as $n^2 + o(n)$, and $n^{\mathrm{O}(1)}$.

2 Algorithmic Techniques

2.1 Divide-and-Conquer

A **divide-and-conquer algorithm** $\mathcal{A}$ follows the following general steps.

Base Case If the problem instance is O(1) in size, then use a brute-force procedure that requires O(1) steps.

Divide Divide the problem instance into one or more subproblem instances, each having a size that is smaller than the original instance.

Conquer Each subproblem instance is solved by making a recursive call to $\mathcal{A}$.

Combine Combine the subproblem-instance solutions into a final solution to the original problem instance.

The following are some problems that can be efficiently solved using a divide-and-conquer algorithm.

Binary Search locating an integer in a sorted array of integers

Quicksort and Mergesort sorting an array of integers

Order Statistics finding the k th least or greatest integer of an array

Convex Hulls finding the convex hull of a set of points in $\mathcal{R}^n$

Minimum Distance Pair finding two points from a set of points in $\mathcal{R}^2$ that are closest

Matrix Operations matrix inversion, matrix multiplication, finding the largest submatrix of 1's in a Boolean matrix.

Fast Fourier Transform finding the product of two polynomials

Maximum Subsequence Sum finding the maximum sum of any subsequence in a sequence of integers.

Minimum Positive Subsequence Sum finding the minimum positive sum of any subsequence in a sequence of integers.

Multiplication of Binary Numbers finding the product of two binary numbers

2.2 Graph Traversals

Types of traversals: depth first, breadth first, and best first

The following are some problems that can be efficiently solved using one or more graph traversals.

Reachability determining if a vertex in a graph is reachable from another vertex

Distance finding the minimum-length path between two vertices

Connectivity finding the connected components of a graph

Topological Sort sorting the vertices of a DAG

2.3 Greedy Algorithms

A **greedy algorithm** is often considered the easiest of algorithms to describe and implement, and is characterized by the following two properties:

1. the algorithm works in successive stages, and during each stage a choice is made that is locally optimal
2. the sum totality of all the locally optimal choices produces a globally optimal solution

If a greedy algorithm does not always lead to a globally optimal solution, then we refer to it as a **heuristic**, or a **greedy heuristic**. Heuristics often provide a "short cut" (not necessarily optimal) solution.

The following are some computational problems that that can be efficiently solved using a greedy algorithm.

Huffman Coding finding a code for a set of items that minimizes the expected code-length

Minimum Spanning Tree finding a spanning tree for a graph whose weighted edges sum to a minimum value

Single source distances in a graph finding the distance from a source vertex in a weighted graph to every other vertex in the graph

Fractional Knapsack selecting a subset of items to load in a container in order to maximize profit

Task Selection finding a maximum set of timewise non-overlapping tasks (each with a fixed start and finish time) that can be completed by a single processor

Unit Task Scheduling with Deadlines finding a task-completion schedule for a single processor in order to maximize the total earned profit

2.4 Dynamic Programming

A **dynamic-programming algorithm** is similar to a divide-and-conquer algorithm in that it attempts to solve a problem instance by relating it to the solutions of sub-problem instances via a recurrence equation. For such an equation a subproblem-instance solution may need to be referenced several times. For this reason, each subproblem-instance solution is stored in a table for future reference.

The following are some problems that may be solved using a dynamic-programming algorithm. All except for the 0-1 Knapsack solution are efficient algorithms.

0-1 Knapsack Given items $x_1, \ldots, x_n$, where item x_i has weight w_i and profit p_i (if it gets placed in the knapsack), determine the subset of items to place in the knapsack in order to maximize profit, assuming that the sack has weight capacity M.

Longest Common Subsequence Given an alphabet Σ, and two words X and Y whose letters belong to Σ, find the longest word Z which is a (non-contiguous) subsequence of both X and Y.

Optimal Binary Search Tree Given a set of keys $k_1, \ldots, k_n$ and weights $w_1, \ldots w_n$, where w_i is proportional to how often k_i is accessed, design a binary search tree so that the weighted cost of accessing a key is minimized.

Matrix Chain Multiplication Given a sequence of matrices that must be multiplied, parenthesize the product so that the total multiplication complexity is minimized.

All-Pairs Minimum Distance Given a directed graph $G = (V, E)$, find the distance between all pairs of vertices in V.

Polygon Triangulation Given a convex polygon $P =< v_0, v_1, \ldots, v_{n-1} >$ and a weight function defined on both the chords and sides of P, find a triangulation of P that minimizes the sum of the weights of which forms the triangulation.

Bitonic Traveling Salesperson given n cities $c_1, \ldots, c_n$, where c_i has grid coordinates (x_i, y_i), and a cost matrix C, where entry C_{ij} denotes the cost of traveling from city i to city j, determine a left-to-right followed by right-to-left Hamilton-cycle tour of all the cities which minimizes the total traveling cost. In other words, the tour starts at the leftmost city, proceeds from left to right visiting a subset of the cities (including the rightmost city), and then concludes from right to left visiting the remaining cities.

Viterbi's algorithm for context-dependent classification Given a set of observations $\vec{x}_1, \ldots, \vec{x}_n$ find the sequence of classes $\omega_1, \ldots, \omega_n$ that are most likely to have produced the observation sequence.

Edit Distance Given two words u and v over some alphabet, determine the least number of edits (letter deletions, additions, and changes) that are needed to transform u into v.