Exercises

2. a. What is the time efficiency of the algorithm for computing a^n as a function of n? As a function of the number of bits in the binary representation of n?

b. If you are to compute a^n mod m where $a > 1$ and n is a large positive integer, how would you circumvent the problem of a very large magnitude of a^n?

4. a. Design an algorithm (or two) for computing the value of a polynomial

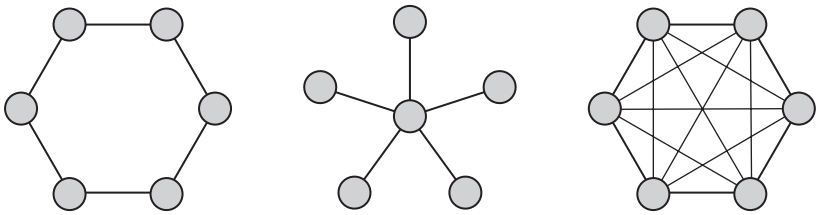
$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

at a given point x_0 and determine its worst-case efficiency class.

b. If the algorithm you designed is in $\Theta(n^2)$, design a linear algorithm for this problem.

c. Is it possible to design an algorithm with a better-than-linear efficiency for this problem?

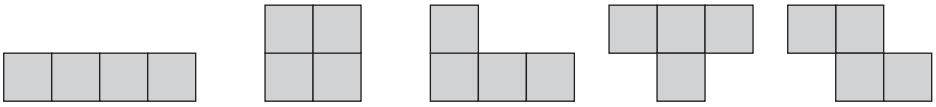
5. A network topology specifies how computers, printers, and other devices are connected over a network. The figure below illustrates three common topologies of networks: the ring, the star, and the fully connected mesh.



ring star fully connected mesh

You are given a boolean matrix $A[0..n-1, 0..n-1]$, where $n > 3$, which is supposed to be the adjacency matrix of a graph modeling a network with one of these topologies. Your task is to determine which of these three topologies, if any, the matrix represents. Design a brute-force algorithm for this task and indicate its time efficiency class.

6. *Tetromino tilings* Tetrominoes are tiles made of four 1×1 squares. There are five types of tetrominoes shown below:



Is it possible to tile—i.e., cover exactly without overlaps—an 8×8 chessboard with

a. straight tetrominoes? **b.** square tetrominoes?

c. L-tetrominoes? **d.** T-tetrominoes?

e. Z-tetrominoes?

7. *A stack of fake coins* There are n stacks of n identical-looking coins. All of the coins in one of these stacks are counterfeit, while all the coins in the other stacks are genuine. Every genuine coin weighs 10 grams; every fake weighs 11 grams. You have an analytical scale that can determine the exact weight of any number of coins.

a. Devise a brute-force algorithm to identify the stack with the fake coins and determine its worst-case efficiency class.

b. What is the minimum number of weighings needed to identify the stack with the fake coins?

8. Sort the list E, X, A, M, P, L, E in alphabetical order by selection sort.

9. Is selection sort stable? (The definition of a stable sorting algorithm was given in Section 1.3.)

10. Is it possible to implement selection sort for linked lists with the same $\Theta(n^2)$ efficiency as the array version?

11. Sort the list E, X, A, M, P, L, E in alphabetical order by bubble sort.

12. **a.** Prove that if bubble sort makes no exchanges on its pass through a list, the list is sorted and the algorithm can be stopped.

b. Write pseudocode of the method that incorporates this improvement.

c. Prove that the worst-case efficiency of the improved version is quadratic.

13. Is bubble sort stable?

14. *Alternating disks* You have a row of $2n$ disks of two colors, n dark and n light. They alternate: dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those that interchange the positions of two neighboring disks.

Design an algorithm for solving this puzzle and determine the number of moves it takes. [Gar99]

```
N  O  B  O  D  Y  _  N  O  T  I  C  E  D  _  H  I  M
N  O  T
   N  O  T
      N  O  T
         N  O  T
            N  O  T
               N  O  T
                  N  O  T
                     N  O  T
```

FIGURE N.3 Example of string matching. The pattern's characters that are compared with their text counterparts are in bold type.

$m(n - m + 1)$ character comparisons, which puts it in the $O(nm)$ class. For a typical word search in a natural language text, however, we should expect that most shifts would happen after very few comparisons (check the example again). Therefore, the average-case efficiency should be considerably better than the worst-case efficiency. Indeed it is: for searching in random texts, it has been shown to be linear, i.e., $\Theta(n)$. There are several more sophisticated and more efficient algorithms for string searching.

Exercises

1. Find the number of comparisons made by the sentinel version of sequential search
   a. in the worst case.
   b. in the average case if the probability of a successful search is p $(0 \le p \le 1)$.
2. As shown in Section 2.1, the average number of key comparisons made by sequential search (without a sentinel, under standard assumptions about its inputs) is given by the formula

$$C_{avg}(n) = \frac{p(n+1)}{2} + n(1-p),$$

   where p is the probability of a successful search. Determine, for a fixed n, the values of p $(0 \le p \le 1)$ for which this formula yields the maximum value of $C_{avg}(n)$ and the minimum value of $C_{avg}(n)$.

3. *Gadget testing* A firm wants to determine the highest floor of its n-story headquarters from which a gadget can fall without breaking. The firm has two identical gadgets to experiment with. If one of them gets broken, it cannot be repaired, and the experiment will have to be completed with the remaining gadget. Design an algorithm in the best efficiency class you can to solve this problem.

4. Determine the number of character comparisons made in searching for the pattern `GANDHI` in the text

```
THERE_IS_MORE_TO_LIFE_THAN_INCREASING_ITS_SPEED
```

Assume that the length of the text—it is 47 characters long—is known before the search starts.

7. In solving the string-matching problem, would there be any advantage in comparing pattern and text characters right-to-left instead of left-to-right?

10. *Word Find* A popular diversion in the United States, "word find" (or "word search") puzzles ask the player to find each of a given set of words in a square table filled with single letters. A word can read horizontally (left or right), vertically (up or down), or along a 45 degree diagonal (in any of the four directions) formed by consecutively adjacent cells of the table; it may wrap around the table's boundaries, but it must read in the same direction with no zigzagging. The same cell of the table may be used in different words, but, in a given word, the same cell may be used no more than once. Write a computer program for solving this puzzle.

Exercises

1. Assuming that *sqrt* takes about 10 times longer than each of the other operations in the innermost loop of *BFClosestPoints*, which are assumed to take the same amount of time, estimate how much faster the algorithm will run after the improvement discussed in Section 3.3.

2. Can you design an efficient algorithm to solve the closest-pair problem for n points $x1$, $x2$, ,,, xn on *the real line?*

3. Let $x_1 < x_2 < \cdots < x_n$ be real numbers representing coordinates of n villages located along a straight road. A post office needs to be built in one of these villages.

a. Design an efficient algorithm to find the post-office location minimizing the average distance between the villages and the post office.

b. Design an efficient algorithm to find the post-office location minimizing the maximum distance from a village to the post office.

---

2. For the sake of simplicity, we assume here that no three points of a given set lie on the same line. A modification needed for the general case is left for the exercises.

4. **a.** There are several alternative ways to define a distance between two points $p_1(x_1, y_1)$ and $p_2(x_2, y_2)$ in the Cartesian plane. In particular, the ***Manhattan distance*** is defined as

$$d_M(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|.$$

Prove that d_M satisfies the following axioms, which every distance function must satisfy:

i. $d_M(p_1, p_2) \geq 0$ for any two points p_1 and p_2, and $d_M(p_1, p_2) = 0$ if and only if $p_1 = p_2$

ii. $d_M(p_1, p_2) = d_M(p_2, p_1)$

iii. $d_M(p_1, p_2) \leq d_M(p_1, p_3) + d_M(p_3, p_2)$ for any p_1, p_2, and p_3

b. Sketch all the points in the Cartesian plane whose Manhattan distance to the origin (0, 0) is equal to 1. Do the same for the Euclidean distance.

c. True or false: A solution to the closest-pair problem does not depend on which of the two metrics—d_E (Euclidean) or d_M (Manhattan)—is used?

5. The ***Hamming distance*** between two strings of equal length is defined as the number of positions at which the corresponding symbols are different. It is named after Richard Hamming (1915–1998), a prominent American scientist and engineer, who introduced it in his seminal paper on error-detecting and error-correcting codes.

a. Does the Hamming distance satisfy the three axioms of a distance metric listed in Problem 4?

b. What is the time efficiency class of the brute-force algorithm for the closest-pair problem if the points in question are strings of m symbols long and the distance between two of them is measured by the Hamming distance?

6. *Odd pie fight* There are $n \geq 3$ people positioned on a field (Euclidean plane) so that each has a unique nearest neighbor. Each person has a cream pie. At a signal, everybody hurls his or her pie at the nearest neighbor. Assuming that n is odd and that nobody can miss his or her target, true or false: There always remains at least one person not hit by a pie. [Car79]

7. The closest-pair problem can be posed in the k-dimensional space, in which the Euclidean distance between two points $p'(x'_1, \ldots, x'_k)$ and $p''(x''_1, \ldots, x''_k)$ is defined as

$$d(p', p'') = \sqrt{\sum_{s=1}^{k} (x'_s - x''_s)^2}.$$

What is the time-efficiency class of the brute-force algorithm for the k-dimensional closest-pair problem?

8. Find the convex hulls of the following sets and identify their extreme points (if they have any):

a. a line segment

b. a square

c. the boundary of a square

d. a straight line

12. Consider the following small instance of the linear programming problem:

$$\begin{aligned} \text{maximize} \quad & 3x + 5y \\ \text{subject to} \quad & x + \ y \le 4 \\ & x + 3y \le 6 \\ & x \ge 0,\ y \ge 0. \end{aligned}$$

a. Sketch, in the Cartesian plane, the problem's ***feasible region***, defined as the set of points satisfying all the problem's constraints.

b. Identify the region's extreme points.

c. Solve this optimization problem by using the following theorem: A linear programming problem with a nonempty bounded feasible region always has a solution, which can be found at one of the extreme points of its feasible region.

Exhaustive Search

Many important problems require finding an element with a special property in a domain that grows exponentially (or faster) with an instance size. Typically, such problems arise in situations that involve—explicitly or implicitly—combinatorial objects such as permutations, combinations, and subsets of a given set. Many such problems are optimization problems: they ask to find an element that maximizes or minimizes some desired characteristic such as a path length or an assignment cost.

Exhaustive search is simply approach to combinatorial problems.It suggests generating each and every element of the problem domain, selecting those of them that satisfy all the constraints, and then finding a desired element (e.g., the one that optimizes some objective function). Note that although the idea of exhaustive search is quite straightforward, its implementation typically requires an algorithm for generating certain combinatorial objects. We delay a discussion of such algorithms until the next chapter and assume here that they exist.

$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

<1, 2, 3, 4> cost = 9 + 4 + 1 + 4 = 18
<1, 2, 4, 3> cost = 9 + 4 + 8 + 9 = 30
<1, 3, 2, 4> cost = 9 + 3 + 8 + 4 = 24
<1, 3, 4, 2> cost = 9 + 3 + 8 + 6 = 26
<1, 4, 2, 3> cost = 9 + 7 + 8 + 9 = 33
<1, 4, 3, 2> cost = 9 + 7 + 1 + 6 = 23
etc.

FIGURE First few iterations of solving a small instance of the assignment problem by exhaustive search.

Since the number of permutations to be considered for the general case of the assignment problem is $n!$, exhaustive search is impractical for all but very small instances of the problem. Fortunately, there is a much more efficient algorithm for this problem called the ***Hungarian method*** after the Hungarian mathematicians König and Egerváry, whose work underlies the method (see, e.g., [Kol95]).

This is good news: the fact that a problem domain grows exponentially or faster does not necessarily imply that there can be no efficient algorithm for solving it. In fact, we present several other examples of such problems later in the book. However, such examples are more of an exception to the rule. More often than not, there are no known polynomial-time algorithms for problems whose domain grows exponentially with instance size, provided we want to solve them exactly. And, as we mentioned above, such algorithms quite possibly do not exist.

Exercises

1. a. Assuming that each tour can be generated in constant time, what will be the efficiency class of the exhaustive-search algorithm outlined in the text for the traveling salesman problem?

b. If this algorithm is programmed on a computer that makes ten billion additions per second, estimate the maximum number of cities for which the problem can be solved in

i. 1 hour. **ii.** 24 hours. **iii.** 1 year. **iv.** 1 century.

2. Outline an exhaustive-search algorithm for the Hamiltonian circuit problem.

3. Outline an algorithm to determine whether a connected graph represented by its adjacency matrix has an Eulerian circuit. What is the efficiency class of your algorithm?

4. Complete the application of exhaustive search to the instance of the assignment problem started in the text.

5. Give an example of the assignment problem whose optimal solution does not include the smallest element of its cost matrix.

6. Consider the ***partition problem***: given n positive integers, partition them into two disjoint subsets with the same sum of their elements. (Of course, the problem does not always have a solution.) Design an exhaustive-search algorithm for this problem. Try to minimize the number of subsets the algorithm needs to generate.

7. Consider the ***clique problem***: given a graph G and a positive integer k, determine whether the graph contains a ***clique*** of size k, i.e., a complete subgraph of k vertices. Design an exhaustive-search algorithm for this problem.

8. Explain how exhaustive search can be applied to the sorting problem and determine the efficiency class of such an algorithm.

9. *Eight-queens problem* Consider the classic puzzle of placing eight queens on an 8×8 chessboard so that no two queens are in the same row or in the same column or on the same diagonal. How many different positions are there so that

a. no two queens are on the same square?

b. no two queens are in the same row?

c. no two queens are in the same row or in the same column?

Also estimate how long it would take to find all the solutions to the problem by exhaustive search based on each of these approaches on a computer capable of checking 10 billion positions per second.

10. *Magic squares* A magic square of order n is an arrangement of the integers from 1 to n^2 in an $n \times n$ matrix, with each number occurring exactly once, so that each row, each column, and each main diagonal has the same sum.

a. Prove that if a magic square of order n exists, the sum in question must be equal to $n(n^2 + 1)/2$.

b. Design an exhaustive-search algorithm for generating all magic squares of order n.

c. Go to the Internet or your library and find a better algorithm for generating magic squares.

d. Implement the two algorithms—the exhaustive search and the one you have found—and run an experiment to determine the largest value of n for which each of the algorithms is able to find a magic square of order n in less than 1 minute on your computer.

11. *Famous alphametic* A puzzle in which the digits in a correct mathematical expression, such as a sum, are replaced by letters is called ***cryptarithm***; if, in addition, the puzzle's words make sense, it is said to be an ***alphametic***. The most well-known alphametic was published by the renowned British puzzlist Henry E. Dudeney (1857–1930):

$$\begin{array}{r} \text{S E N D} \\ +\ \text{M O R E} \\ \hline \text{M O N E Y} \end{array}$$

Two conditions are assumed: first, the correspondence between letters and decimal digits is one-to-one, i.e., each letter represents one digit only and different letters represent different digits. Second, the digit zero does not appear as the left-most digit in any of the numbers. To solve an alphametic means to find which digit each letter represents. Note that a solution's uniqueness cannot be assumed and has to be verified by the solver.

a. Do not write a program for solving cryptarithms by exhaustive search. Assume that a given cryptarithm is a sum of two words.

b. Solve Dudeney's puzzle.

Depth-First Search and Breadth-First Search

The term "exhaustive search" can also be applied to two very important algorithms that systematically process all vertices and edges of a graph. These two traversal algorithms are ***depth-first search (DFS)*** and ***breadth-first search (BFS)***. These algorithms have proved to be very useful for many applications involving graphs in artificial intelligence and operations research. In addition, they are indispensable for efficient investigation of fundamental properties of graphs such as connectivity and cycle presence.

Depth-First Search

Depth-first search starts a graph's traversal at an arbitrary vertex by marking it as visited. On each iteration, the algorithm proceeds to an unvisited vertex that is adjacent to the one it is currently in. (If there are several such vertices, a tie can be resolved arbitrarily. As a practical matter, which of the adjacent unvisited candidates is chosen is dictated by the data structure representing the graph. In our examples, we always break ties by the alphabetical order of the vertices.) This process continues until a dead end—a vertex with no adjacent unvisited vertices—is encountered. At a dead end, the algorithm backs up one edge to the vertex it came from and tries to continue visiting unvisited vertices from there. The algorithm eventually halts after backing up to the starting vertex, with the latter being a dead end. By then, all the vertices in the same connected component as the starting vertex have been visited. If unvisited vertices still remain, the depth-first search must be restarted at any one of them.

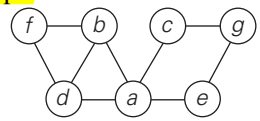
It is convenient to use a stack to trace the operation of depth-first search. We push a vertex onto the stack when the vertex is reached for the first time (i.e., the

TABLE N1 Main facts about depth-first search (DFS) and breadth-first search (BFS)

	DFS	BFS
Data structure	a stack	a queue
Number of vertex orderings	two orderings	one ordering
Edge types (undirected graphs)	tree and back edges	tree and cross edges
Applications	connectivity, acyclicity, articulation points	connectivity, acyclicity, minimum-edge paths
Efficiency for adjacency matrix	$\Theta(\|V^2\|)$	$\Theta(\|V^2\|)$
Efficiency for adjacency lists	$\Theta(\|V\| + \|E\|)$	$\Theta(\|V\| + \|E\|)$

Exercises

1. Consider the following graph.



a. Write down the adjacency matrix and adjacency lists specifying this graph. (Assume that the matrix rows and columns and vertices in the adjacency lists follow in the alphabetical order of the vertex labels.)

b. Starting at vertex a and resolving ties by the vertex alphabetical order, traverse the graph by depth-first search and construct the corresponding depth-first search tree. Give the order in which the vertices were reached for the first time (pushed onto the traversal stack) and the order in which the vertices became dead ends (popped off the stack).

2. If we define sparse graphs as graphs for which $|E| \in O(|V|)$, which implementation of DFS will have a better time efficiency for such graphs, the one that uses the adjacency matrix or the one that uses the adjacency lists?

3. Let G be a graph with n vertices and m edges.

a. True or false: All its DFS forests (for traversals starting at different vertices) will have the same number of trees?

b. True or false: All its DFS forests will have the same number of tree edges and the same number of back edges?

4. Traverse the graph of Problem 1 by breadth-first search and construct the corresponding breadth-first search tree. Start the traversal at vertex a and resolve ties by the vertex alphabetical order.

5. Prove that a cross edge in a BFS tree of an undirected graph can connect vertices only on either the same level or on two adjacent levels of a BFS tree.

6. **a.** Explain how one can check a graph's acyclicity by using breadth-first search.

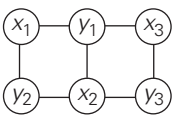
b. Does either of the two traversals—DFS or BFS—always find a cycle faster than the other? If you answer yes, indicate which of them is better and explain why it is the case; if you answer no, give two examples supporting your answer.

7. Explain how one can identify connected components of a graph by using

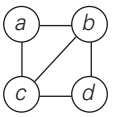
a. a depth-first search.

b. a breadth-first search.

8. A graph is said to be ***bipartite*** if all its vertices can be partitioned into two disjoint subsets X and Y so that every edge connects a vertex in X with a vertex in Y. (One can also say that a graph is bipartite if its vertices can be colored in two colors so that every edge has its vertices colored in different colors; such graphs are also called ***2-colorable***.) For example, graph (i) is bipartite while graph (ii) is not.



(i)



(ii)

a. Design a DFS-based algorithm for checking whether a graph is bipartite.

b. Design a BFS-based algorithm for checking whether a graph is bipartite.

9. Write a program that, for a given graph, outputs:

a. vertices of each connected component

b. its cycle or a message that the graph is acyclic

10. One can model a maze by having a vertex for a starting point, a finishing point, dead ends, and all the points in the maze where more than one path can be taken, and then connecting the vertices according to the paths in the maze.

a. Construct such a graph for the following maze.

b. Which traversal—DFS or BFS—would you use if you found yourself in a maze and why?

11. *Three Jugs* Siméon Denis Poisson (1781–1840), a famous French mathematician and physicist, is said to have become interested in mathematics after encountering some version of the following old puzzle. Given an 8-pint jug full of water and two empty jugs of 5- and 3-pint capacity, get exactly 4 pints of water in one of the jugs by completely filling up and/or emptying jugs into others. Solve this puzzle by using breadth-first search.

SUMMARY

- The principal strengths of the approach presented are wide applicability and simplicity; its principal weakness is the subpar efficiency.
- A first application of the approach presented, often results in an algorithm that can be improved with a modest amount of effort.
- The following noted algorithms can be considered good examples of this approach:
  - definition-based algorithm for matrix multiplication
  - *selection sort*
  - *sequential search*
  - straightforward string-matching algorithm
- *Exhaustive search* is an approach to combinatorial problems. It suggests generating each and every combinatorial object of the problem, selecting those of them that satisfy all the constraints, and then finding a desired object.
- The *traveling salesman problem,* the *knapsack problem,* and the *assignment problem* are typical examples of problems that can be solved, at least theoretically, by exhaustive-search algorithms.
- Exhaustive search is impractical for all but very small instances of problems it can be applied to.
- *Depth-first search* (*DFS*) and *breadth-first search* (*BFS*) are two principal graph-traversal algorithms. By representing a graph in a form of a depth-first or breadth-first search forest, they help in the investigation of many important properties of the graph. Both algorithms have the same time efficiency: $\Theta(|V|^2)$ for the adjacency matrix representation and $\Theta(|V| + |E|)$ for the adjacency list representation.