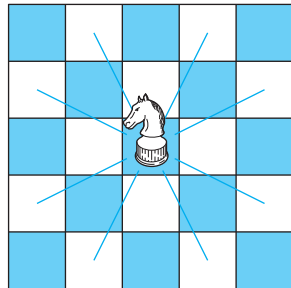


A **knight** is a chess piece that can move either two spaces horizontally and one space vertically or one space horizontally and two spaces vertically. That is, a knight on square (x, y) can move to any of the eight squares $(x \pm 2, y \pm 1)$, $(x \pm 1, y \pm 2)$, if these squares are on the chessboard, as illustrated here.



A **knight's tour** is a sequence of legal moves by a knight starting at some square and visiting each square exactly once. A knight's tour is called **reentrant** if there is a legal move that takes the knight from the last square of the tour back to where the tour began. We can model knight's tours using the graph that has a vertex for each square on the board, with an edge connecting two vertices if a knight can legally move between the squares represented by these vertices.

56. Draw the graph that represents the legal moves of a knight on a 3×3 chessboard.
57. Draw the graph that represents the legal moves of a knight on a 3×4 chessboard.
58. a) Show that finding a knight's tour on an $m \times n$ chessboard is equivalent to finding a Hamilton path on the graph representing the legal moves of a knight on that board.
b) Show that finding a reentrant knight's tour on an $m \times n$ chessboard is equivalent to finding a Hamilton circuit on the corresponding graph.
- *59. Show that there is a knight's tour on a 3×4 chessboard.
- *60. Show that there is no knight's tour on a 3×3 chessboard.
- *61. Show that there is no knight's tour on a 4×4 chessboard.
62. Show that the graph representing the legal moves of a knight on an $m \times n$ chessboard, whenever m and n are positive integers, is bipartite.
63. Show that there is no reentrant knight's tour on an $m \times n$ chessboard when m and n are both odd. [Hint: Use Exercises 55, 58b, and 62.]
- *64. Show that there is a knight's tour on an 8×8 chessboard. [Hint: You can construct a knight's tour using a method invented by H. C. Warnsdorff in 1823: Start in any square, and then always move to a square connected to the fewest number of unused squares. Although this method may not always produce a knight's tour, it often does.]
65. The parts of this exercise outline a proof of Ore's theorem. Suppose that G is a simple graph with n vertices, $n \geq 3$, and $\deg(x) + \deg(y) \geq n$ whenever x and y are nonadjacent vertices in G . Ore's theorem states that under these conditions, G has a Hamilton circuit.
 - a) Show that if G does not have a Hamilton circuit, then there exists another graph H with the same vertices as G , which can be constructed by adding edges to G such that the addition of a single edge would produce a Hamilton circuit in H . [Hint: Add as many edges as possible at each successive vertex of G without producing a Hamilton circuit.]
 - b) Show that there is a Hamilton path in H .
 - c) Let v_1, v_2, \dots, v_n be a Hamilton path in H . Show that $\deg(v_1) + \deg(v_n) \geq n$ and that there are at most $\deg(v_1)$ vertices not adjacent to v_n (including v_n itself).
 - d) Let S be the set of vertices preceding each vertex adjacent to v_1 in the Hamilton path. Show that S contains $\deg(v_1)$ vertices and $v_n \notin S$.
 - e) Show that S contains a vertex v_k , which is adjacent to v_n , implying that there are edges connecting v_1 and v_{k+1} and v_k and v_n .
 - f) Show that part (e) implies that $v_1, v_2, \dots, v_{k-1}, v_k, v_n, v_{n-1}, \dots, v_{k+1}, v_1$ is a Hamilton circuit in G . Conclude from this contradiction that Ore's theorem holds.
- *66. Show that the worst case computational complexity of Algorithm 1 for finding Euler circuits in a connected graph with all vertices of even degree is $O(m)$, where m is the number of edges of G .

10.6 Shortest-Path Problems

Introduction

Many problems can be modeled using graphs with weights assigned to their edges. As an illustration, consider how an airline system can be modeled. We set up the basic graph model by representing cities by vertices and flights by edges. Problems involving distances can be modeled by assigning distances between cities to the edges. Problems involving flight time can be modeled by assigning flight times to edges. Problems involving fares can be modeled by

12/29 11:18
| read
12:06
| notes
12:34

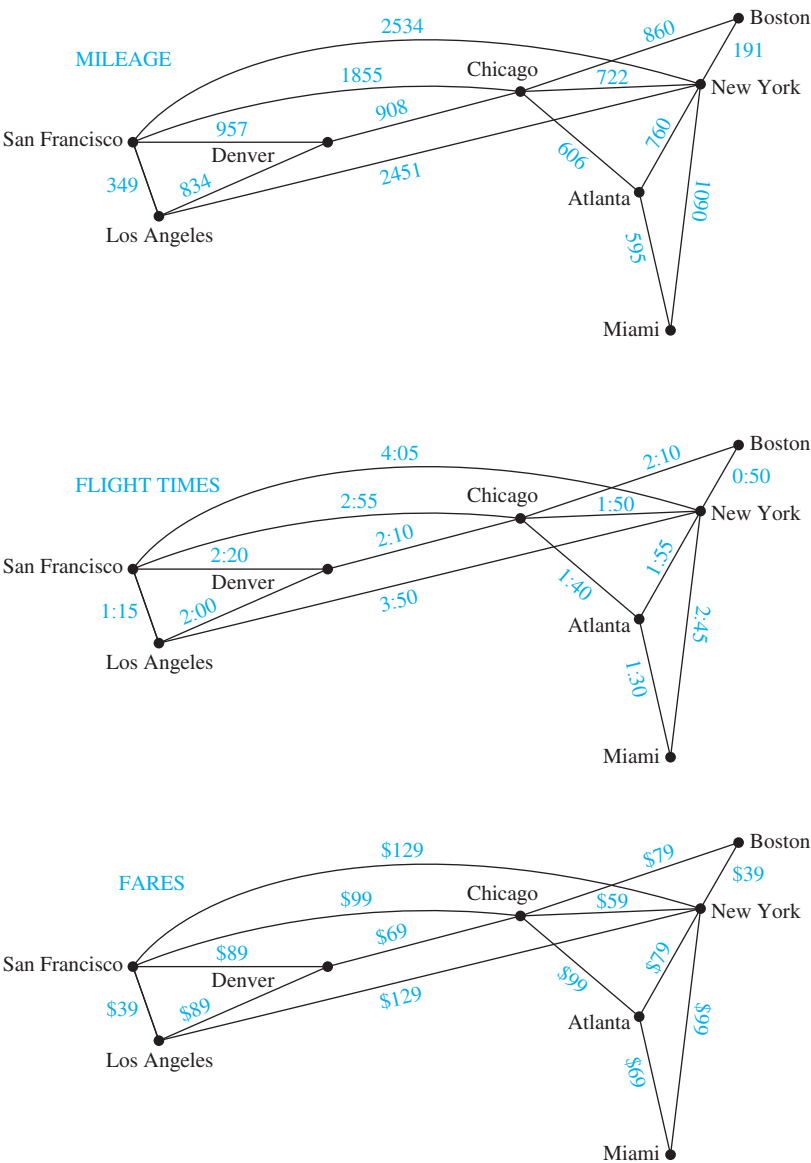


FIGURE 1 Weighted Graphs Modeling an Airline System.

assigning fares to the edges. Figure 1 displays three different assignments of weights to the edges of a graph representing distances, flight times, and fares, respectively.

Graphs that have a number assigned to each edge are called **weighted graphs**. Weighted graphs are used to model computer networks. Communications costs (such as the monthly cost of leasing a telephone line), the response times of the computers over these lines, or the distance between computers, can all be studied using weighted graphs. Figure 2 displays weighted graphs that represent three ways to assign weights to the edges of a graph of a computer network, corresponding to distance, response time, and cost.

Several types of problems involving weighted graphs arise frequently. Determining a path of least length between two vertices in a network is one such problem. To be more specific, let the **length** of a path in a weighted graph be the sum of the weights of the edges of this path. (The reader should note that this use of the term *length* is different from the use of *length* to denote the number of edges in a path in a graph without weights.) The question is: What is a shortest path, that is, a path of least length, between two given vertices? For instance, in the airline system

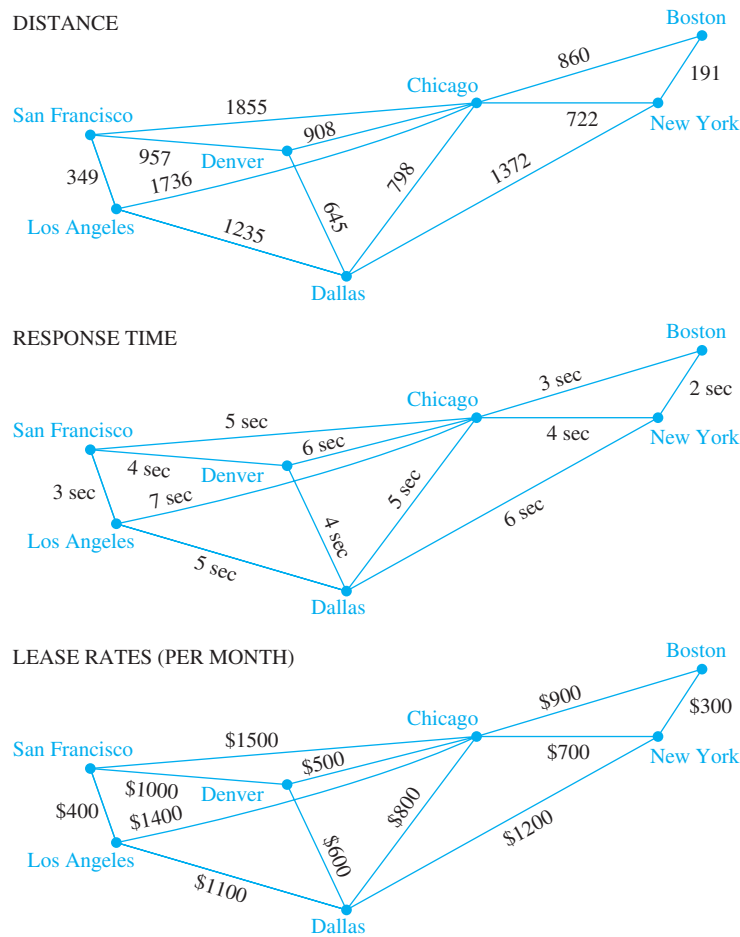


FIGURE 2 Weighted Graphs Modeling a Computer Network.

represented by the weighted graph shown in Figure 1, what is a shortest path in air distance between Boston and Los Angeles? What combinations of flights has the smallest total flight time (that is, total time in the air, not including time between flights) between Boston and Los Angeles? What is the cheapest fare between these two cities? In the computer network shown in Figure 2, what is a least expensive set of telephone lines needed to connect the computers in San Francisco with those in New York? Which set of telephone lines gives a fastest response time for communications between San Francisco and New York? Which set of lines has a shortest overall distance?

Another important problem involving weighted graphs asks for a circuit of shortest total length that visits every vertex of a complete graph exactly once. This is the famous *traveling salesperson problem*, which asks for an order in which a salesperson should visit each of the cities on his route exactly once so that he travels the minimum total distance. We will discuss the traveling salesperson problem later in this section.

start



A Shortest-Path Algorithm

There are several different algorithms that find a shortest path between two vertices in a weighted graph. We will present a greedy algorithm discovered by the Dutch mathematician Edsger Di-

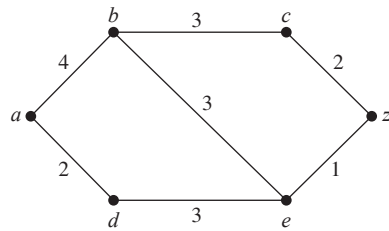


FIGURE 3 A Weighted Simple Graph.

jkstra in 1959. The version we will describe solves this problem in undirected weighted graphs where all the weights are positive. It is easy to adapt it to solve shortest-path problems in directed graphs.

Before giving a formal presentation of the algorithm, we will give an illustrative example.

EXAMPLE 1 What is the length of a shortest path between a and z in the weighted graph shown in Figure 3?

Solution: Although a shortest path is easily found by inspection, we will develop some ideas useful in understanding Dijkstra's algorithm. We will solve this problem by finding the length of a shortest path from a to successive vertices, until z is reached.

The only paths starting at a that contain no vertex other than a are formed by adding an edge that has a as one endpoint. These paths have only one edge. They are a, b of length 4 and a, d of length 2. It follows that d is the closest vertex to a , and the shortest path from a to d has length 2.

We can find the second closest vertex by examining all paths that begin with the shortest path from a to a vertex in the set $\{a, d\}$, followed by an edge that has one endpoint in $\{a, d\}$ and its other endpoint not in this set. There are two such paths to consider, a, d, e of length 7 and a, b of length 4. Hence, the second closest vertex to a is b and the shortest path from a to b has length 4.


To find the third closest vertex to a , we need examine only the paths that begin with the shortest path from a to a vertex in the set $\{a, d, b\}$, followed by an edge that has one endpoint in the set $\{a, d, b\}$ and its other endpoint not in this set. There are three such paths, a, b, c of length 7, a, b, e of length 7, and a, d, e of length 5. Because the shortest of these paths is a, d, e , the third closest vertex to a is e and the length of the shortest path from a to e is 5.



EDSGER WYBE DIJKSTRA (1930–2002) Edsger Dijkstra, born in the Netherlands, began programming computers in the early 1950s while studying theoretical physics at the University of Leiden. In 1952, realizing that he was more interested in programming than in physics, he quickly completed the requirements for his physics degree and began his career as a programmer, even though programming was not a recognized profession. (In 1957, the authorities in Amsterdam refused to accept “programming” as his profession on his marriage license. However, they did accept “theoretical physicist” when he changed his entry to this.)

Dijkstra was one of the most forceful proponents of programming as a scientific discipline. He has made fundamental contributions to the areas of operating systems, including deadlock avoidance; programming languages, including the notion of structured programming; and algorithms. In 1972 Dijkstra received the Turing

Award from the Association for Computing Machinery, one of the most prestigious awards in computer science. Dijkstra became a Burroughs Research Fellow in 1973, and in 1984 he was appointed to a chair in Computer Science at the University of Texas, Austin.

To find the fourth closest vertex to a , we need examine only the paths that begin with the shortest path from a to a vertex in the set $\{a, d, b, e\}$, followed by an edge that has one endpoint in the set $\{a, d, b, e\}$ and its other endpoint not in this set. There are two such paths, a, b, c of length 7 and a, d, e, z of length 6. Because the shorter of these paths is a, d, e, z , the fourth closest vertex to a is z and the length of the shortest path from a to z is 6. 

Example 1 illustrates the general principles used in Dijkstra's algorithm. Note that a shortest path from a to z could have been found by a brute force approach by examining the length of every path from a to z . However, this brute force approach is impractical for humans and even for computers for graphs with a large number of edges.

We will now consider the general problem of finding the length of a shortest path between a and z in an undirected connected simple weighted graph. Dijkstra's algorithm proceeds by finding the length of a shortest path from a to a first vertex, the length of a shortest path from a to a second vertex, and so on, until the length of a shortest path from a to z is found. As a side benefit, this algorithm is easily extended to find the length of the shortest path from a to all other vertices of the graph, and not just to z .

The algorithm relies on a series of iterations. A distinguished set of vertices is constructed by adding one vertex at each iteration. A labeling procedure is carried out at each iteration. In this labeling procedure, a vertex w is labeled with the length of a shortest path from a to w that contains only vertices already in the distinguished set. The vertex added to the distinguished set is one with a minimal label among those vertices not already in the set.

We now give the details of Dijkstra's algorithm. It begins by labeling a with 0 and the other vertices with ∞ . We use the notation $L_0(a) = 0$ and $L_0(v) = \infty$ for these labels before any iterations have taken place (the subscript 0 stands for the "0th" iteration). These labels are the lengths of shortest paths from a to the vertices, where the paths contain only the vertex a . (Because no path from a to a vertex different from a exists, ∞ is the length of a shortest path between a and this vertex.)

Dijkstra's algorithm proceeds by forming a distinguished set of vertices. Let S_k denote this set after k iterations of the labeling procedure. We begin with $S_0 = \emptyset$. The set S_k is formed from S_{k-1} by adding a vertex u not in S_{k-1} with the smallest label.

Once u is added to S_k , we update the labels of all vertices not in S_k , so that $L_k(v)$, the label of the vertex v at the k th stage, is the length of a shortest path from a to v that contains vertices only in S_k (that is, vertices that were already in the distinguished set together with u). Note that the way we choose the vertex u to add to S_k at each step is an optimal choice at each step, making this a greedy algorithm. (We will prove shortly that this greedy algorithm always produces an optimal solution.)

Let v be a vertex not in S_k . To update the label of v , note that $L_k(v)$ is the length of a shortest path from a to v containing only vertices in S_k . The updating can be carried out efficiently when this observation is used: A shortest path from a to v containing only elements of S_k is either a shortest path from a to v that contains only elements of S_{k-1} (that is, the distinguished vertices not including u), or it is a shortest path from a to u at the $(k-1)$ st stage with the edge $\{u, v\}$ added. In other words,

$$L_k(a, v) = \min\{L_{k-1}(a, v), L_{k-1}(a, u) + w(u, v)\},$$

where $w(u, v)$ is the length of the edge with u and v as endpoints. This procedure is iterated by successively adding vertices to the distinguished set until z is added. When z is added to the distinguished set, its label is the length of a shortest path from a to z .

Dijkstra's algorithm is given in Algorithm 1. Later we will give a proof that this algorithm is correct. Note that we can find the length of the shortest path from a to all other vertices of the graph if we continue this procedure until all vertices are added to the distinguished set.

ALGORITHM 1 Dijkstra's Algorithm.


```

procedure Dijkstra( $G$ : weighted connected simple graph, with
    all weights positive)
    { $G$  has vertices  $a = v_0, v_1, \dots, v_n = z$  and lengths  $w(v_i, v_j)$ 
    where  $w(v_i, v_j) = \infty$  if  $\{v_i, v_j\}$  is not an edge in  $G$ }
    for  $i := 1$  to  $n$ 
         $L(v_i) := \infty$ 
     $L(a) := 0$ 
     $S := \emptyset$ 
    {the labels are now initialized so that the label of  $a$  is 0 and all
    other labels are  $\infty$ , and  $S$  is the empty set}
    while  $z \notin S$ 
         $u :=$  a vertex not in  $S$  with  $L(u)$  minimal
         $S := S \cup \{u\}$ 
        for all vertices  $v$  not in  $S$ 
            if  $L(u) + w(u, v) < L(v)$  then  $L(v) := L(u) + w(u, v)$ 
            {this adds a vertex to  $S$  with minimal label and updates the
            labels of vertices not in  $S$ }
    return  $L(z)$  { $L(z)$  = length of a shortest path from  $a$  to  $z$ }

```

Example 2 illustrates how Dijkstra's algorithm works. Afterward, we will show that this algorithm always produces the length of a shortest path between two vertices in a weighted graph.

EXAMPLE 2 Use Dijkstra's algorithm to find the length of a shortest path between the vertices a and z in the weighted graph displayed in Figure 4(a).

Solution: The steps used by Dijkstra's algorithm to find a shortest path between a and z are shown in Figure 4. At each iteration of the algorithm the vertices of the set S_k are circled. A shortest path from a to each vertex containing only vertices in S_k is indicated for each iteration. The algorithm terminates when z is circled. We find that a shortest path from a to z is a, c, b, d, e, z , with length 13. 

Remark: In performing Dijkstra's algorithm it is sometimes more convenient to keep track of labels of vertices in each step using a table instead of redrawing the graph for each step.

Next, we use an inductive argument to show that Dijkstra's algorithm produces the length of a shortest path between two vertices a and z in an undirected connected weighted graph. Take as the inductive hypothesis the following assertion: At the k th iteration

- (i) the label of every vertex v in S is the length of a shortest path from a to this vertex, and
- (ii) the label of every vertex not in S is the length of a shortest path from a to this vertex that contains only (besides the vertex itself) vertices in S .

When $k = 0$, before any iterations are carried out, $S = \emptyset$, so the length of a shortest path from a to a vertex other than a is ∞ . Hence, the basis case is true.

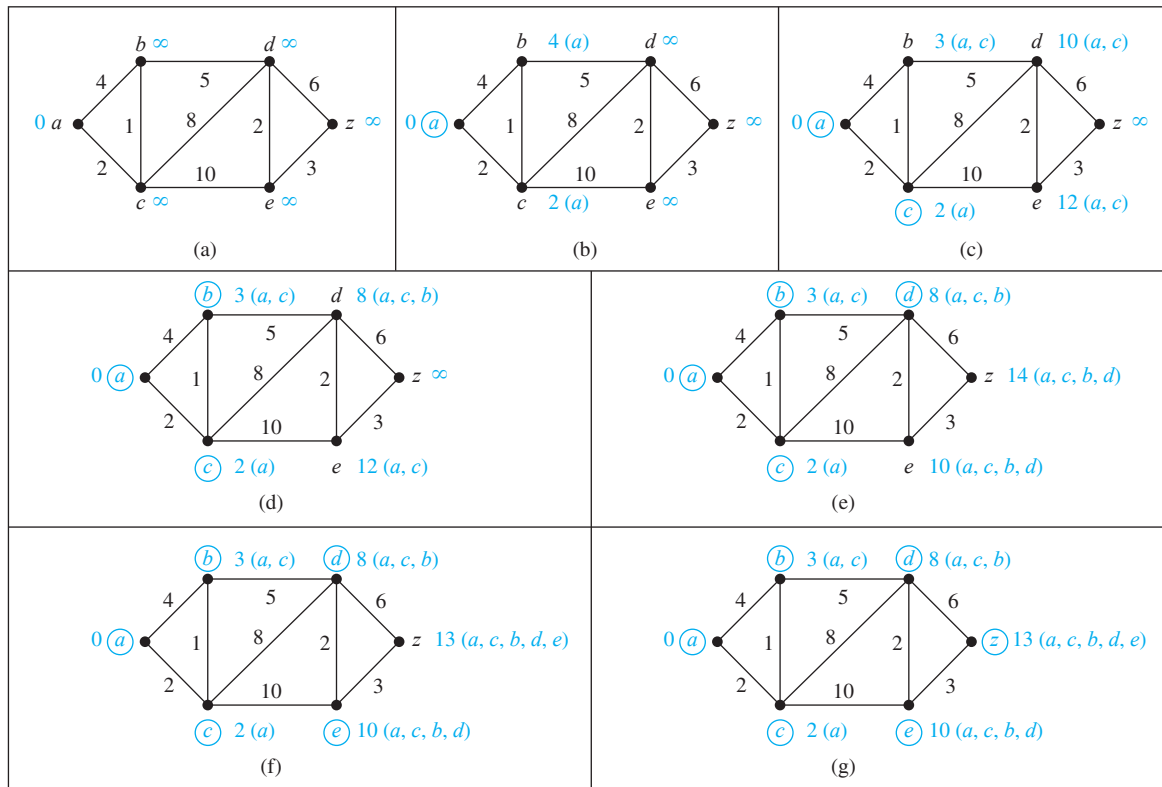


FIGURE 4 Using Dijkstra's Algorithm to Find a Shortest Path from a to z .



Assume that the inductive hypothesis holds for the k th iteration. Let v be the vertex added to S at the $(k + 1)$ st iteration, so v is a vertex not in S at the end of the k th iteration with the smallest label (in the case of ties, any vertex with smallest label may be used).

From the inductive hypothesis we see that the vertices in S before the $(k + 1)$ st iteration are labeled with the length of a shortest path from a . Also, v must be labeled with the length of a shortest path to it from a . If this were not the case, at the end of the k th iteration there would be a path of length less than $L_k(v)$ containing a vertex not in S [because $L_k(v)$ is the length of a shortest path from a to v containing only vertices in S after the k th iteration]. Let u be the first vertex not in S in such a path. There is a path with length less than $L_k(v)$ from a to u containing only vertices of S . This contradicts the choice of v . Hence, (i) holds at the end of the $(k + 1)$ st iteration.

Let u be a vertex not in S after $k + 1$ iterations. A shortest path from a to u containing only elements of S either contains v or it does not. If it does not contain v , then by the inductive hypothesis its length is $L_k(u)$. If it does contain v , then it must be made up of a path from a to v of shortest possible length containing elements of S other than v , followed by the edge from v to u . In this case, its length would be $L_k(v) + w(v, u)$. This shows that (ii) is true, because $L_{k+1}(u) = \min\{L_k(u), L_k(v) + w(v, u)\}$.

We now state the theorem that we have proved.

THEOREM 1

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

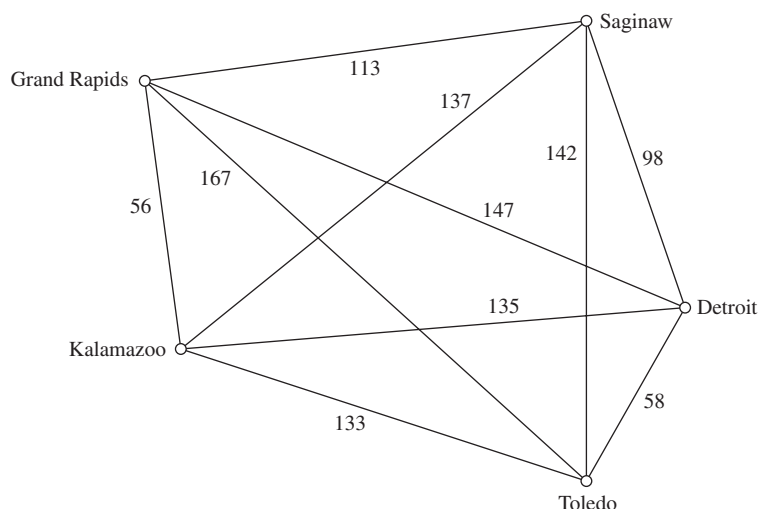


FIGURE 5 The Graph Showing the Distances between Five Cities.

We can now estimate the computational complexity of Dijkstra's algorithm (in terms of additions and comparisons). The algorithm uses no more than $n - 1$ iterations where n is the number of vertices in the graph, because one vertex is added to the distinguished set at each iteration. We are done if we can estimate the number of operations used for each iteration. We can identify the vertex not in S_k with the smallest label using no more than $n - 1$ comparisons. Then we use an addition and a comparison to update the label of each vertex not in S_k . It follows that no more than $2(n - 1)$ operations are used at each iteration, because there are no more than $n - 1$ labels to update at each iteration. Because we use no more than $n - 1$ iterations, each using no more than $2(n - 1)$ operations, we have Theorem 2.

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.

The Traveling Salesperson Problem



We now discuss an important problem involving weighted graphs. Consider the following problem: A traveling salesperson wants to visit each of n cities exactly once and return to his starting point. For example, suppose that the salesperson wants to visit Detroit, Toledo, Saginaw, Grand Rapids, and Kalamazoo (see Figure 5). In which order should he visit these cities to travel the minimum total distance? To solve this problem we can assume the salesperson starts in Detroit (because this must be part of the circuit) and examine all possible ways for him to visit the other four cities and then return to Detroit (starting elsewhere will produce the same circuits). There are a total of 24 such circuits, but because we travel the same distance when we travel a circuit in reverse order, we need only consider 12 different circuits to find the minimum total distance he must travel. We list these 12 different circuits and the total distance traveled for each circuit. As can be seen from the list, the minimum total distance of 458 miles is traveled using the circuit Detroit–Toledo–Kalamazoo–Grand Rapids–Saginaw–Detroit (or its reverse).

<i>Route</i>	<i>Total Distance (miles)</i>
Detroit–Toledo–Grand Rapids–Saginaw–Kalamazoo–Detroit	610
Detroit–Toledo–Grand Rapids–Kalamazoo–Saginaw–Detroit	516
Detroit–Toledo–Kalamazoo–Saginaw–Grand Rapids–Detroit	588
Detroit–Toledo–Kalamazoo–Grand Rapids–Saginaw–Detroit	458
Detroit–Toledo–Saginaw–Kalamazoo–Grand Rapids–Detroit	540
Detroit–Toledo–Saginaw–Grand Rapids–Kalamazoo–Detroit	504
Detroit–Saginaw–Toledo–Grand Rapids–Kalamazoo–Detroit	598
Detroit–Saginaw–Toledo–Kalamazoo–Grand Rapids–Detroit	576
Detroit–Saginaw–Kalamazoo–Toledo–Grand Rapids–Detroit	682
Detroit–Saginaw–Grand Rapids–Toledo–Kalamazoo–Detroit	646
Detroit–Grand Rapids–Saginaw–Toledo–Kalamazoo–Detroit	670
Detroit–Grand Rapids–Toledo–Saginaw–Kalamazoo–Detroit	728

We just described an instance of the **traveling salesperson problem**. The traveling salesperson problem asks for 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.

The most straightforward way to solve an instance of the traveling salesperson problem is to examine all possible Hamilton circuits and select one of minimum total length. How many circuits do we have to examine to solve the problem if there are n vertices in the graph? Once a starting point is chosen, there are $(n - 1)!$ different Hamilton circuits to examine, because there are $n - 1$ choices for the second vertex, $n - 2$ choices for the third vertex, and so on. Because a Hamilton circuit can be traveled in reverse order, we need only examine $(n - 1)!/2$ circuits to find our answer. Note that $(n - 1)!/2$ grows extremely rapidly. Trying to solve a traveling salesperson problem in this way when there are only a few dozen vertices is impractical. For example, with 25 vertices, a total of $24!/2$ (approximately 3.1×10^{23}) different Hamilton circuits would have to be considered. If it took just one nanosecond (10^{-9} second) to examine each Hamilton circuit, a total of approximately ten million years would be required to find a minimum-length Hamilton circuit in this graph by exhaustive search techniques.

Because the traveling salesperson problem has both practical and theoretical importance, a great deal of effort has been devoted to devising efficient algorithms that solve it. However, no algorithm with polynomial worst-case time complexity is known for solving this problem. Furthermore, if a polynomial worst-case time complexity algorithm were discovered for the traveling salesperson problem, many other difficult problems would also be solvable using polynomial worst-case time complexity algorithms (such as determining whether a proposition in n variables is a tautology, discussed in Chapter 1). This follows from the theory of NP-completeness. (For more information about this, consult [GaJo79].)

A practical approach to the traveling salesperson problem when there are many vertices to visit is to use an **approximation algorithm**. These are algorithms that do not necessarily produce the exact solution to the problem but instead are guaranteed to produce a solution that is close to an exact solution. (Also, see the preamble to Exercise 46 in the Supplementary Exercises of Chapter 3.) That is, they may produce a Hamilton circuit with total weight W' such that $W \leq W' \leq cW$, where W is the total length of an exact solution and c is a constant. For example, there is an algorithm with polynomial worst-case time complexity that works if the weighted graph satisfies the triangle inequality such that $c = 3/2$. For general weighted graphs for every positive real number k no algorithm is known that will always produce a solution at most k times a best solution. If such an algorithm existed, this would show that the class P would be the same as the class NP, perhaps the most famous open question about the complexity of algorithms (see Section 3.3).

An 1832 handbook *Der Handlungsreisende* (The Traveling Salesman) mentions the traveling salesman problem, with sample tours through Germany and Switzerland.

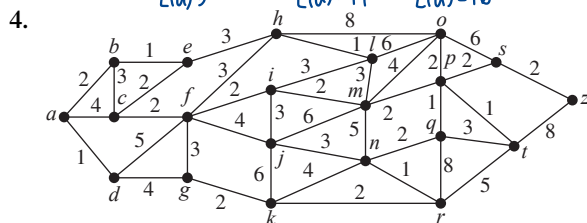
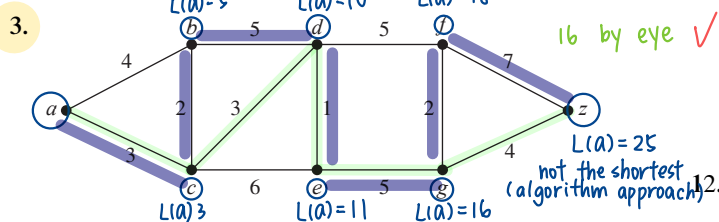
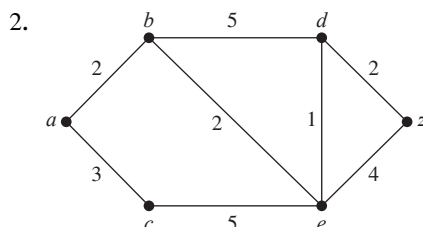
In practice, algorithms have been developed that can solve traveling salesperson problems with as many as 1000 vertices within 2% of an exact solution using only a few minutes of computer time. For more information about the traveling salesperson problem, including history, applications, and algorithms, see the chapter on this topic in *Applications of Discrete Mathematics* [MiRo91] also available on the website for this book.

$12/29$ $12:37$
 $13:12$ $13:29$ | correct

Exercises

- For each of these problems about a subway system, describe a weighted graph model that can be used to solve the problem.
 - What is the least amount of time required to travel between two stops?
 - What is the minimum distance that can be traveled to reach a stop from another stop?
 - What is the least fare required to travel between two stops if fares between stops are added to give the total fare?

In Exercises 2–4 find the length of a shortest path between a and z in the given weighted graph.

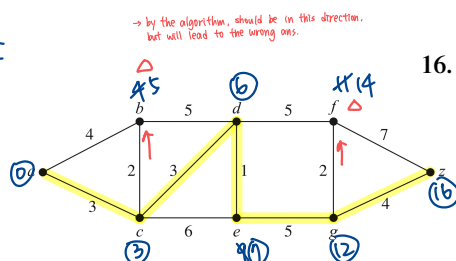


- Find a shortest path between a and z in each of the weighted graphs in Exercises 2–4.

$a \rightarrow b \rightarrow g \rightarrow z$ $a \rightarrow c \rightarrow f \rightarrow z$
 $a \rightarrow b \rightarrow f \rightarrow z$ $a \rightarrow c \rightarrow e \rightarrow g \rightarrow z$

- Find the length of a shortest path between these pairs of vertices in the weighted graph in Exercise 3.

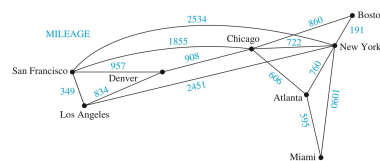
- a and d 6 $a \rightarrow c \rightarrow d$
- a and f 11 $a \rightarrow c \rightarrow d \rightarrow f$
- c and f 8 $c \rightarrow d \rightarrow f$
- b and z 15



- Find shortest paths in the weighted graph in Exercise 3 between the pairs of vertices in Exercise 6.

- Find a shortest path (in mileage) between each of the following pairs of cities in the airline system shown in Figure 1.

- New York and Los Angeles *direct flight*
- Boston and San Francisco *visit Chicago*
- Miami and Denver *visit Atlanta, Chicago*
- Miami and Los Angeles *visit Atlanta, Chicago, Denver*



- Find a combination of flights with the least total air time between the pairs of cities in Exercise 8, using the flight times shown in Figure 1.

- Find a least expensive combination of flights connecting the pairs of cities in Exercise 8, using the fares shown in Figure 1.

- Find a shortest route (in distance) between computer centers in each of these pairs of cities in the communications network shown in Figure 2.

- Boston and Los Angeles
- New York and San Francisco
- Dallas and San Francisco
- Denver and New York

- Find a route with the shortest response time between the pairs of computer centers in Exercise 11 using the response times given in Figure 2.

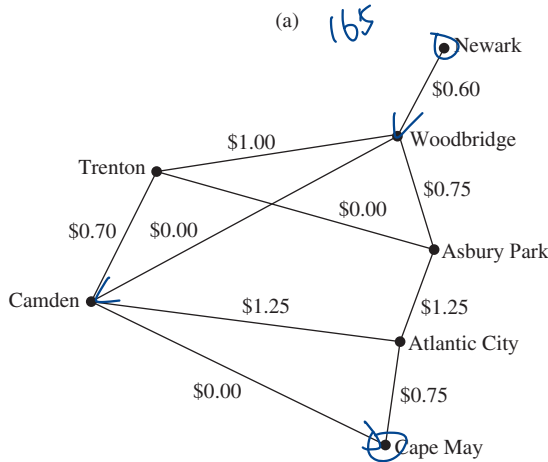
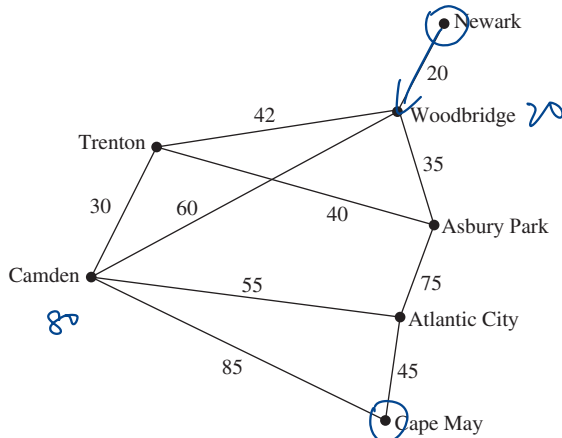
- Find a least expensive route, in monthly lease charges, between the pairs of computer centers in Exercise 11 using the lease charges given in Figure 2.

- Explain how to find a path with the least number of edges between two vertices in an undirected graph by considering it as a shortest path problem in a weighted graph.

- Extend Dijkstra's algorithm for finding the length of a shortest path between two vertices in a weighted simple connected graph so that the length of a shortest path between the vertex a and every other vertex of the graph is found.

- Extend Dijkstra's algorithm for finding the length of a shortest path between two vertices in a weighted simple connected graph so that a shortest path between these vertices is constructed.

17. The weighted graphs in the figures here show some major roads in New Jersey. Part (a) shows the distances between cities on these roads; part (b) shows the tolls.



- a) Find a shortest route in distance between Newark and Camden, and between Newark and Cape May, using these roads.
 Visit Woodbridge, Camden ✓ $d = 80$
 Visit Woodbridge, Cape May ✓ $d = 165$
- b) Find a least expensive route in terms of total tolls using the roads in the graph between the pairs of cities in part (a) of this exercise.
 Visit Woodbridge, Cape May ✓ $\$0.60$
 Visit Woodbridge, Camden ✓ $\$0.60$
18. Is a shortest path between two vertices in a weighted graph unique if the weights of edges are distinct? **no**
 cuz it can be unique # but same sum
 ex. $4 + 9 = 6 + 7$
19. What are some applications where it is necessary to find the length of a longest simple path between two vertices in a weighted graph?
20. What is the length of a longest simple path in the weighted graph in Figure 4 between a and z ? Between c and z ?



Floyd's algorithm, displayed as Algorithm 2, can be used to find the length of a shortest path between all pairs of vertices in a weighted connected simple graph. However, this algorithm cannot be used to construct shortest paths. (We assign an infinite weight to any pair of vertices not connected by an edge in the graph.)

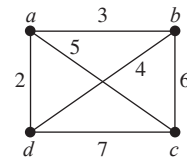
21. Use Floyd's algorithm to find the distance between all pairs of vertices in the weighted graph in Figure 4(a).
- *22. Prove that Floyd's algorithm determines the shortest distance between all pairs of vertices in a weighted simple graph.
- *23. Give a big- O estimate of the number of operations (comparisons and additions) used by Floyd's algorithm to determine the shortest distance between every pair of vertices in a weighted simple graph with n vertices.
- *24. Show that Dijkstra's algorithm may not work if edges can have negative weights.

ALGORITHM 2 Floyd's Algorithm.

```

procedure Floyd( $G$ : weighted simple graph)
  {  $G$  has vertices  $v_1, v_2, \dots, v_n$  and weights  $w(v_i, v_j)$ 
    with  $w(v_i, v_j) = \infty$  if  $\{v_i, v_j\}$  is not an edge }
  for  $i := 1$  to  $n$ 
    for  $j := 1$  to  $n$ 
       $d(v_i, v_j) := w(v_i, v_j)$ 
  for  $i := 1$  to  $n$ 
    for  $j := 1$  to  $n$ 
      for  $k := 1$  to  $n$ 
        if  $d(v_j, v_i) + d(v_i, v_k) < d(v_j, v_k)$ 
          then  $d(v_j, v_k) := d(v_j, v_i) + d(v_i, v_k)$ 
  return  $[d(v_i, v_j)]$  {  $d(v_i, v_j)$  is the length of a shortest
    path between  $v_i$  and  $v_j$  for  $1 \leq i \leq n, 1 \leq j \leq n$  }
  
```

25. Solve the traveling salesperson problem for this graph by finding the total weight of all Hamilton circuits and determining a circuit with minimum total weight.



26. Solve the traveling salesperson problem for this graph by finding the total weight of all Hamilton circuits and determining a circuit with minimum total weight.

