SECTION 7.3 REVIEW

TECHNIQUES

- Find a shortest path from x to y in a graph (using Dijkstra's algorithm).
- Find a minimal spanning tree for a graph (using Prim's algorithm).

MAIN IDEA

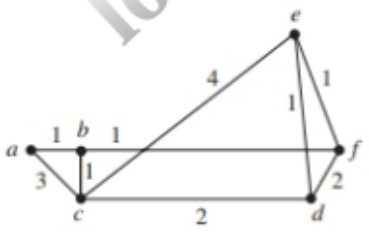
- Algorithms that are $\Theta(n^2)$ in the worst case can find a shortest path between two nodes or a minimal spanning tree in a simple, positively weighted, connected graph with n nodes.

EXERCISES 7.3

For Exercises 1–4, use the graph that follows. Apply Dijkstra's algorithm for the pairs of nodes given; show the values for p and IN and the d values and s values for each pass through the **while** loop. Write out the nodes in the shortest path and the distance of the path.

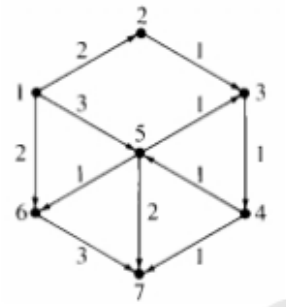
1. From 2 to 5
2. From 3 to 6
3. From 1 to 5
4. From 4 to 7

For Exercises 5 and 6, use the graph that follows. Apply Dijkstra's algorithm for the pairs of nodes given; show the values for p and IN and the d values and s values for each pass through the **while** loop. Write out the nodes in the shortest path and the distance of the path.



5. From a to e
6. From d to a

For Exercises 7 and 8, use the directed graph that follows. Apply Dijkstra's algorithm to the nodes given; show the values for p and IN and the d values and s values for each pass through the **while** loop. Write out the nodes in the shortest path and the distance of the path.



7. From 1 to 7
8. From 3 to 1
9. a. Modify Dijkstra's algorithm so that it finds the shortest paths from x to all other nodes in the graph.
   b. Does this change the worst-case order of magnitude of the algorithm?
10. Give an example to show that Dijkstra's algorithm does not work when negative weights are allowed.

Another algorithm for finding shortest paths from a single source node to all other nodes in the graph is the *Bellman–Ford algorithm*. In contrast to Dijkstra's algorithm, which keeps a set of nodes whose shortest path (minimum-weight path) of whatever length (that is, number of hops) has been determined, the Bellman–Ford algorithm performs a series of computations that seeks to find successively smaller-weight paths of length 1, then of length 2, then of length 3, and so on, up to a maximum of length $n - 1$ (if a path exists at all, then there is a path of length no greater than $n - 1$). A pseudocode description of the Bellman–Ford algorithm is given in the accompanying box; when using this algorithm, the adjacency matrix **A** must have $\mathbf{A}[i, i] = 0$ for all i.

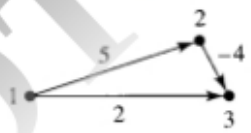
ALGORITHM ***BELLMAN–FORD ALGORITHM***

```
Bellman–Ford(n × n matrix A; node x; array of integers d; array of nodes s)
//Computes the shortest path between a source node x and all other nodes in a simple,
//weighted, connected graph. A is a modified adjacency matrix with A[i, i] = 0.
//When procedure terminates, the nodes in the shortest path from x to a node y
//are y, s[y], s[s[y]], ... , x; the distance for that path is d[y].
Local variables:
nodes z, p                  //temporary nodes
array of integers t         //temporary distance array created at each iteration

   //initialize arrays d and s; this establishes the shortest 1-length paths from x
   d[x] = 0
   for all nodes z not equal to x do
      d[z] = A[x, z]
      s[z] = x
   end for

   //find shortest paths of length 2, 3, etc.
   for i = 2 to n − 1 do
      t = d                 //copy current array d into array t
```

```
        //modify t to hold shortest paths of length i
        for all nodes z not equal to x do
            //find the shortest path with one more link
            p = node in G for which (d[p] + A[p, z]) is minimum
            t[z] = d[p] + A[p, z]
            if p ≠ z then
                s[z] = p
            end if
        end for
        d = t ;     //copy array t back into d
    end for
end Bellman–Ford
```

For Exercises 11–14 use the Bellman–Ford algorithm to find the shortest path from the source node to any other node. Show the successive d values and s values.

11. Graph for Exercises 1–4, source node = 2 (compare your answer to Exercise 1)
12. Graph for Exercises 1–4, source node = 1 (compare your answer to Exercise 3)
13. Graph for Exercises 7–8, source node = 1 (compare your answer to Exercise 7)
14. a. Accompanying graph, source node = 1 (compare your answer to Exercise 10)
    b. What does this say about the Bellman–Ford algorithm as opposed to Dijkstra's algorithm?



To compute the distance for the shortest path between any two nodes in a graph, Dijkstra's algorithm could be used repeatedly, with each node in turn as the source node. A different algorithm, *Floyd's algorithm*, can also be used to solve this "all pairs" shortest-path problem, but while Floyd's algorithm produces the weight of all shortest paths, it does not calculate what the shortest paths actually are, that is, what nodes are on a given shortest path. Floyd's algorithm is very similar to Warshall's algorithm. A description follows, where **A** is the adjacency matrix of the graph with $\mathbf{A}[i, i] = 0$ for all i.

ALGORITHM ***FLOYD'S ALGORITHM***

```
Floyd (n × n matrix A)
//Computes the shortest path between any two nodes in a simple, weighted,
//connected graph; A is a modified adjacency matrix with A[i,i] = 0.
//Upon termination, A will contain all the shortest-path distances
    for k = 1 to n do
        for i = 1 to n do
            for j = 1 to n do
                if A[i, k] + A[k, j] < A[i, j] then
                    A[i, j] = A[i, k] + A[k, j]
                end if
            end for
        end for
    end for
end Floyd
```

For Exercises 15 and 16, use Floyd's algorithm to find the distances for all the shortest paths. Show the successive values of the **A** matrix for each pass through the outer loop.

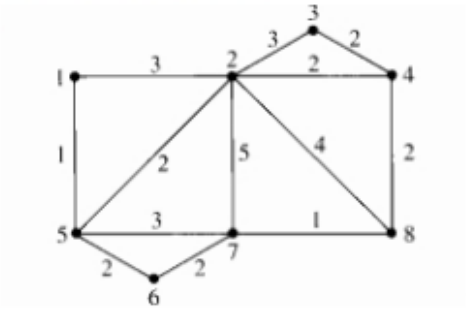
15. Figure 7.10

16. Graph for Exercises 1–4

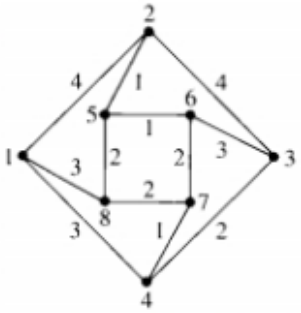
For Exercises 17–20, use Prim's algorithm to find a minimal spanning tree for the graph in the specified figure.

17. Graph for Exercises 1–4

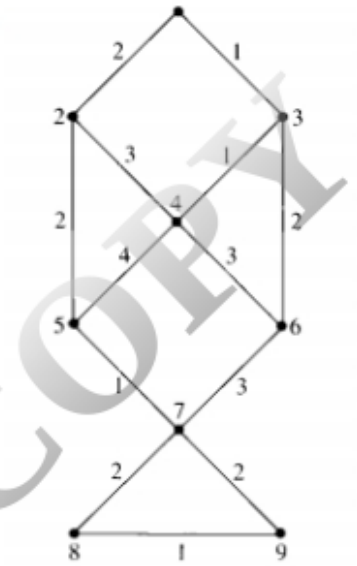
18.



19.



20.



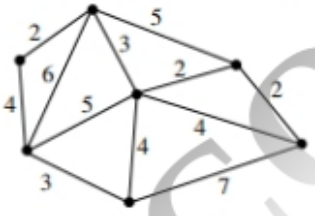
Kruskal's algorithm is another algorithm for finding a minimal spanning tree in a connected graph. Whereas Prim's algorithm "grows" the tree from an arbitrary starting point by attaching adjacent short arcs, Kruskal's algorithm adds arcs in order by increasing distance wherever they may be in the graph. Ties are resolved arbitrarily. The only restriction is that an arc is not added if adding it would create a cycle. The algorithm terminates when all nodes have been incorporated into a connected structure. A (very informal) pseudocode description follows:

ALGORITHM ***KRUSKAL'S ALGORITHM***

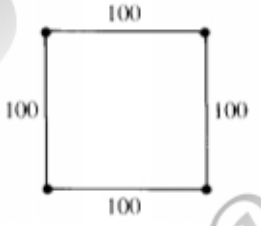
```
Kruskal (n × n matrix A; collection of arcs T)
//Finds a minimal spanning tree; T is initially empty;
//at termination, T = minimal spanning tree
   order arcs in G by increasing distance
   repeat
      if next arc in order does not complete a cycle then
         add that arc into T
      end if
   until T is connected and contains all nodes of G
end Kruskal
```

For Exercises 21–24 use Kruskal's algorithm to find the minimal spanning tree.

21. Graph for Exercises 1–4
22. Graph for Exercise 18
23. Graph for Exercise 19
24. Graph for Exercise 20
25. Give an example to show that adding the node closest to *IN* at each step, as is done in Prim's minimal spanning tree algorithm, will not guarantee a shortest path.
26. Let a be the arc of lowest weight in a weighted graph. Show that a must be an arc in any minimal spanning tree.
27. A city plans to lay out bike paths connecting various city parks. A map with the distances between the parks is shown in the figure. (Some direct connections would have to cross major highways, so these distances are not shown in the map.) Find which paths to pave so that all parks are connected but the cost is minimal.



28. Assume that arc weights represent distance. Then adding new nodes and arcs to a graph may result in a spanning tree for the new graph that has less weight than a spanning tree for the original graph. (The new spanning tree could represent a minimal-cost network for communications between a group of cities obtained by adding a switch in a location outside any of the cities.)
    a. Find a spanning tree of minimum weight for the following labeled graph. What is its weight?



    b. Put a node in the center of the square. Add new arcs from the center to the corners. Find a spanning tree for the new graph, and compute its (approximate) weight.
29. At the beginning of this chapter, you received the following assignment:

You are the network administrator for a wide-area backbone network that serves your company's many offices across the country. Messages travel through the network by being routed from point to point until they reach their destination. Each node in the network therefore acts as a switching station to forward messages to other nodes according to a routing table maintained at each node. Some connections in the network carry heavy traffic, while others are less used. Traffic may vary with the time of day; in addition, new nodes occasionally come on line and existing nodes may go off line. Therefore you must periodically provide each node with updated information so that it can forward messages along the most efficient (that is, the least heavily traveled) route.

How can you compute the routing table for each node?

You realize that you can represent the network as a weighted graph, where the arcs are the connections between nodes and the weights of the arcs represent traffic on the connections. The routing problem then becomes one of finding the shortest path in the graph from any node to any other node. Dijkstra's algorithm can be used to give the shortest path from any one node to all other nodes (see Exercise 9), so you could use the algorithm repeatedly with different start nodes. Or you could use Floyd's algorithm. Discuss the advantages and disadvantages of each approach, including an analysis of the order of magnitude of each approach.