SECTION 10.5 Minimum Spanning Trees

2. We start with the minimum weight edge $\{u, b\}$. The least weight edge incident to the tree constructed so far is edge $\{a, e\}$, with weight 2, so we add it to the tree. Next we add edge $\{d, e\}$, and then edge $\{c, d\}$. This completes the tree, whose total weight is 6.

4. The edges are added in the order $\{a, b\}$, $\{a, e\}$, $\{a, d\}$, $\{c, d\}$, $\{d, h\}$, $\{a, m\}$, $\{d, p\}$, $\{e, f\}$, $\{e, i\}$, $\{g, h\}$, $\{l, p\}$, $\{m, n\}$, $\{n, o\}$, $\{f, j\}$, and $\{k, l\}$, for a total weight of 28.

6. With Kruskal's algorithm, we add at each step the shortest edge that will not complete a simple circuit. Thus we pick edge $\{a, b\}$ first, and then edge $\{c, d\}$ (alphabetical order breaks ties), followed by $\{a, e\}$ and $\{d, e\}$.The total weight is 6.

8. The edges are added in the order $\{a, b\}$, $\{a, e\}$, $\{c, d\}$, $\{d, h\}$, $\{a, d\}$, $\{a, m\}$, $\{d, p\}$, $\{e, f\}$, $\{e, i\}$, $\{g, h\}$, $\{l, p\}$, $\{m, n\}$, $\{n, o\}$, $\{f, j\}$, and $\{k, l\}$, for a total weight of 28.

10. One way to do this is simply to apply the algorithm of choice to each component. In practice it is not clear what that means, since we would have to determine the components first. More to the point, we can implement the procedures as follows. For Prim's algorithm, start with the first vertex and repeatedly add to the tree the shortest edge adjacent to it that does not complete a simple circuit. When no such edges remain, we find a vertex that is not yet in the spanning forest and grow a new tree from this vertex. We repeat this process until no new vertices remain. Kruskal's algorithm is even simpler to implement. We keep choosing the shortest edge that does not complete a simple circuit, until no such edges remain. The result is a spanning forest of minimum weight.

12. If we simply replace the word "smallest" with the word "largest" (and replace the word "minimum" in the comment with the word "maximum") in Algorithm 2, then the resulting algorithm will find a maximum spanning tree.

14. The answer is unique. It uses edges $\{d,h\}$, $\{d,e\}$, $\{b,f\}$, $\{d,g\}$, $\{a,b\}$, $\{b,e\}$, $\{b,c\}$, and $\{f,i\}$.

16. We follow the procedure outlined in the solution to Exercise 17. Recall that the minimum spanning tree uses the edges Atlanta–Chicago, Atlanta–New York, Denver–San Francisco, and Chicago–San Francisco. First we delete the edge from Atlanta to Chicago. The minimum spanning tree for the remaining graph has cost \$3900. Next we delete the edge from Atlanta to New York (and put the previously deleted edge back). The minimum spanning tree now has cost \$3800. Next we look at the graph with the edge from Denver to San Francisco deleted. The minimum spanning tree has cost \$4000. Finally we look at the graph with the edge from Chicago to San Francisco deleted. The minimum spanning tree has cost \$3700. This last tree is our answer, then; it consists of the links Atlanta–Chicago, Atlanta–New York, Denver–San Francisco, and Chicago–Denver.

18. Suppose that an edge e with smallest weight is not included in some minimum spanning tree; in other words, suppose that the minimum spanning tree T contains only edges with weights larger than that of e. If we add e to T, then we will obtain a graph with exactly one simple circuit, which contains e. We can then delete some other edge in this circuit, resulting in a spanning tree with weight strictly less than that of T (since all the other edges have larger weight than e has). This is a contradiction to the fact that T is a minimum spanning tree. Therefore an edge with smallest weight must be included in T.

20. We start with the New York to Denver link and then form a spanning tree by successively adding the cheapest edges that do not form a simple circuit. In fact the three cheapest edges will do: Atlanta–Chicago, Atlanta–New York, and Denver–San Francisco. This gives a cost of \$4000.

22. The algorithm is the same as Kruskal's, except that instead of starting with the empty tree, we start with the given set of edges. (If there is already a simple circuit among these edges, then there is no solution.)

24. We prove this by contradiction. Suppose that there is a simple circuit formed after the addition of edges at some stage in the algorithm. The circuit will contain some edges that were added at that stage and perhaps some edges that were already present. Let e_1, e_2, ..., e_r be the edges that are new, in the order they are traversed in the circuit. Thus the circuit can be thought of as the sequence e_1, T_1, e_2, T_2, ..., e_r, T_r, e_1, where each T_i is a tree that existed before the addition of new edges. Each edge in this sequence was the edge picked by the tree containing one of its two endpoints, so since there are the same number of trees as there are edges in this sequence, each tree must have picked a different edge. However, let e be the shortest edge (after tie-breaking) among $\{e_1, e_2, \ldots, e_r\}$. Then the tree at both of its ends necessarily picked e to add to the tree, a contradiction. Therefore there are no simple circuits.

26. The actual implementation of this algorithm is more difficult than this pseudocode shows, of course.

```
procedure Sollin(G : simple graph)
initialize the set of trees to be the set of vertices
while |set of trees| > 1 do
begin
        for each tree T_i in the set of trees
                e_i := the shortest edge from a vertex in T_i to a vertex not in T_i
        add all the e_i's to the trees already present and
            reorganize the resulting graph into a set of trees
end
```

28. This is a special case of Exercise 29, with r equal to the number of vertices in the graph (each vertex is a tree by itself at the beginning of the algorithm); see the solution to that exercise.

30. As argued in the solution to Exercise 29, each stage in the algorithm reduces the number of trees by a factor of at least 2. Therefore after k stages at most $n/2^k$ trees remain. Since the number of trees is an integer, the number must be less than or equal to $\lfloor n/2^k \rfloor$.

32. Let G be a connected weighted graph. Suppose that the successive edges chosen by Kruskal's algorithm are $e_1, e_2, \ldots, e_{n-1}$, in that order, so that the tree S containing these edges is the tree constructed by the algorithm. Let T be a minimum spanning tree of G containing $e_1, e_2, \ldots, e_k$, with k chosen as large as possible (possibly 0). If $k = n-1$, then we are done, since $S = T$. Otherwise $k < n-1$, and in this case we will derive a contradiction by finding a minimum spanning tree T' which gives us a larger value of k. Consider $T \cup \{e_{k+1}\}$. Since T is a tree, this graph has a simple circuit which must contain e_{k+1}. Some edge e in this simple circuit is not in S, since S is a tree. Furthermore, e was available to be chosen by Kruskal's algorithm at the point at which e_{k+1} was chosen, since there is no simple circuit among $\{e_1, e_2, \ldots, e_k, e\}$ (these edges are all in T). Therefore the weight of e_{k+1} is less than or equal to the weight of e (otherwise the algorithm would have chosen e instead of e_{k+1}). Now add e_{k+1} to T and delete e; call the resulting tree T'. The weight of T' cannot be any greater than the weight of T. Therefore T' is also a minimum spanning tree, which contains the edges $e_1, e_2, \ldots, e_k, e_{k+1}$. This contradicts the choice of T, and our proof is complete.