⋆**17.** Notice that Algorithm 7.1.1 is optimal for the denominations

$$d_1 = 50, \quad d_2 = 10, \quad d_3 = 5, \quad d_4 = 1,$$

and that

$$d_i \text{ divides } d_{i-2} - d_{i-1}, \qquad 3 \le i \le n.$$

Show, by giving counterexamples that, nevertheless, the preceding condition is neither necessary nor sufficient for Algorithm 7.1.1 to be optimal.

**18.** Given coins of denominations

$$1 = d[1] < d[2] < \cdots < d[n],$$

prove or disprove whether the following sets $c[i][j]$ equal to the minimum number of coins needed to make change for an amount $j$, $1 \le j \le m$, using only the denominations

$$d[1], \ldots, d[i].$$

```
for i = 0 to n
   c[i][0] = 0
for j = 1 to m
   c[0][j] = 0
for i = 1 to n
   for j = 1 to m
      if (d[i] divides j)
         c[i][j] = j/d[i]
      else
         c[i][j] = min(c[i][j - 1] + 1, c[i - 1][j])
```

# 7.2 Kruskal's Algorithm

Figure 7.2.1 shows six cities and the costs (in hundreds of thousands of dollars) of rebuilding roads between them. The road commission has decided

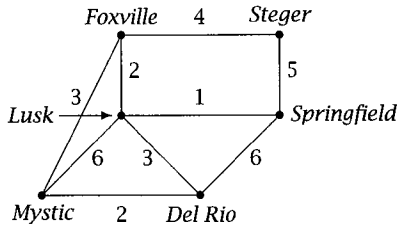



**Figure 7.2.1** Cities and the costs (in hundreds of thousands of dollars) of rebuilding roads between them.

to rebuild enough roads so that each pair of cities will be connected, either directly or by going through other cities, by rebuilt roads. Figure 7.2.2 shows one possibility that costs

$$2 + 6 + 1 + 5 + 4 = 18.$$

The road commission needs an algorithm to find a minimum-cost set of roads meeting its criterion.

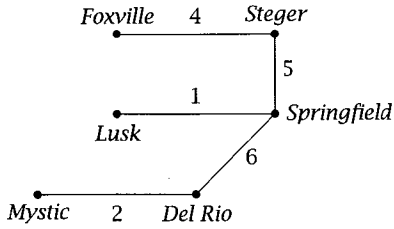



**Figure 7.2.2** A subset of the roads of Figure 7.2.1. If these roads are rebuilt, each pair of cities will be connected, either directly or by going through other cities, by rebuilt roads. The cost of rebuilding these roads is 18.

Roads, cities, and costs can be modeled as a weighted graph where the vertices represent the cities, the edges represent the roads, and the weights represent the costs (see Figure 7.2.3). A **spanning tree** for a graph $G$ is a subgraph of $G$ that is a tree containing all of $G$'s vertices. A **minimal spanning tree** is a spanning tree of minimum weight. Every connected graph has a spanning tree and, therefore, a minimal spanning tree. Thus, the problem of finding a minimum-cost set of roads directly or indirectly connecting all of the cities is the problem of finding a minimal spanning tree.

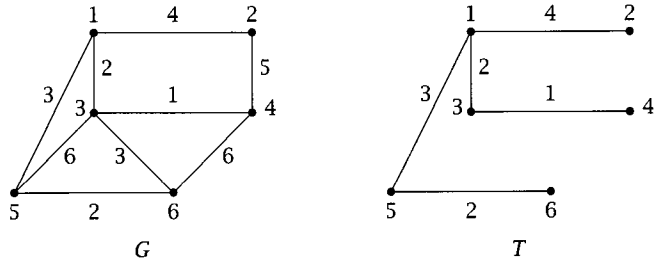



**Figure 7.2.3** The graph $G$ of Figure 7.2.1 with integers replacing the city names and a spanning tree $T$. No other spanning tree has a smaller weight, so $T$ is a *minimal* spanning tree.

**Example 7.2.1.** In Figure 7.2.3, the weight of the spanning tree $T$ for the graph $G$ is 12. No other spanning tree for $G$ has weight less than 12. [The only set of five edges whose weights sum to less than 12 is

$$\{(3,4),(5,6),(1,3),(3,6),(1,5)\},$$

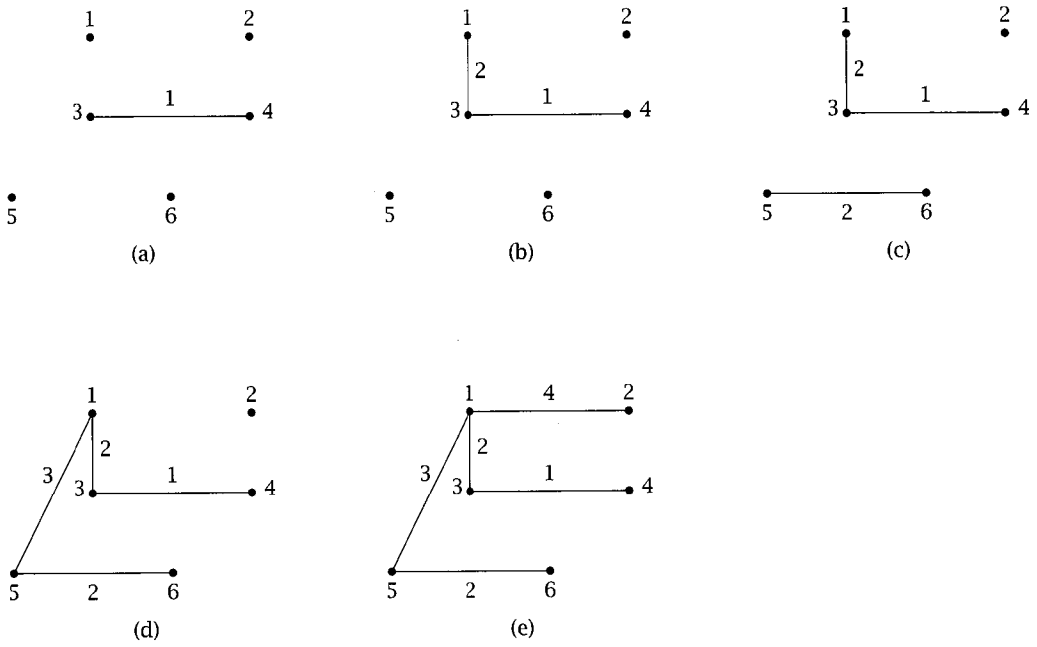



**Figure 7.2.4** Kruskal's algorithm with input the graph $G$ of Figure 7.2.3. The algorithm begins with all of the vertices and no edges. It then repeatedly adds an edge of minimum weight that does not make a cycle. In this case, it first selects edge (3,4) since it has minimum weight [graph (a)]. It next selects an edge of minimum weight 2; we assume that it selects edge $(1,3)$ [graph (b)]. It next selects edge (5,6) since it has minimum weight [graph (c)]. It next selects an edge of minimum weight 3; we assume that it selects edge $(1,5)$ [graph (d)]. Finally, it selects edge $(1,2)$ since it has minimum weight [graph(e)] to complete the minimal spanning tree.

but these edges do not form a tree.] Thus, $T$ is a minimal spanning tree.

Since $G$ models the road system of Figure 7.2.1, the minimal spanning tree $T$ is a solution to the problem of finding a minimum-cost subset of roads directly or indirectly connecting all of the cities. □

*WWW* In this section and the next, we discuss the problem of finding a minimal spanning tree in a connected, weighted graph. Unless specified otherwise, all of the weights are assumed to be positive. **Kruskal's algorithm** is a greedy algorithm for finding a minimal spanning tree in a graph $G$. The algorithm begins with all of the vertices of $G$ and no edges. It then applies the greedy rule: Add an edge of minimum weight that does not make a cycle.

**Example 7.2.2.** We show how Kruskal's algorithm finds a minimal spanning tree for the graph $G$ in Figure 7.2.3. Kruskal's algorithm first selects edge $(3,4)$ since it has minimum weight [see Figure 7.2.4(a)].

Kruskal's algorithm next selects edge $(1,3)$ or $(5,6)$; either edge has minimum weight 2 and neither makes a cycle when added to $\{(3,4)\}$. When

more than one edge has the same minimum weight, any can be selected. Different spanning trees may result, but all will be minimal. Suppose that we arbitrarily select edge $(1,3)$ [see Figure 7.2.4(b)].

Kruskal's algorithm next selects edge $(5,6)$ since it has minimum weight 2 and does not make a cycle when added to $\{(3,4),(1,3)\}$ [see Figure 7.2.4(c)].

Kruskal's algorithm next selects edge $(1,5)$ or $(3,6)$; both have minimum weight 3 and neither makes a cycle when added to

$$\{(3,4),(1,3),(5,6)\}.$$

Suppose that we arbitrarily select edge $(1,5)$ [see Figure 7.2.4(d)].

Kruskal's algorithm next considers selecting edge $(3,6)$, which has minimum weight 3. Since $(3,6)$ makes a cycle when added to

$$\{(3,4),(1,3),(5,6),(1,5)\},$$

it does *not* select $(3,6)$.

Finally, Kruskal's algorithm selects edge $(1,2)$ because it has minimum weight 4 and does not make a cycle when added to

$$\{(3,4),(1,3),(5,6),(1,5)\}$$

[see Figure 7.2.4(e)].

Since we now have a spanning tree, Kruskal's algorithm terminates with the minimal spanning tree shown in Figure 7.2.4(e). □

To implement Kruskal's algorithm, several issues need to be addressed. First, we must represent the graph. Since we are selecting edges by weight, we represent the graph as a list of edges and their weights. Second, we must select the edges in nondecreasing order of weight. We can sort the edges in nondecreasing order by weight and then examine them in sorted order. Third, we must be able to determine whether adding an edge would create a cycle. We observe that adding edge $(v,w)$ creates a cycle when there is a path between $v$ and $w$ formed by edges already selected, that is, when $v$ and $w$ are in the same *component* (see Definition 2.5.25) of the graph of edges already selected. We keep track of components by recording the set of vertices belonging to each component.

**Example 7.2.3.** Consider the graph $G$ of Figure 7.2.3. Its representation is

$$(1,2,4)\ (1,3,2)\ (1,5,3)\ (2,4,5)\ (3,4,1)\ (3,5,6)\ (3,6,3)\ (4,6,6)\ (5,6,2),$$

where $(v_1,v_2,w)$ is interpreted as edge $(v_1,v_2)$ of weight $w$.

We first sort the edges in nondecreasing order by weight:

$$(3,4,1)\ (1,3,2)\ (5,6,2)\ (1,5,3)\ (3,6,3)\ (1,2,4)\ (2,4,5)\ (3,5,6)\ (4,6,6).$$

When Kruskal's algorithm starts, no edges have been selected, so each vertex belongs to a component consisting of itself:

$$\{1\}\quad\{2\}\quad\{3\}\quad\{4\}\quad\{5\}\quad\{6\}.$$

The first edge $(3,4)$ is selected, and the components to which vertices 3 and 4 belong are merged; the components become

$$\{1\} \quad \{2\} \quad \{3,4\} \quad \{5\} \quad \{6\}.$$

Next edge $(1,3)$ is selected, and the components to which vertices 1 and 3 belong are merged; the components become

$$\{1,3,4\} \quad \{2\} \quad \{5\} \quad \{6\}.$$

Next edge $(5,6)$ is selected, and the components to which vertices 5 and 6 belong are merged; the components become

$$\{1,3,4\} \quad \{2\} \quad \{5,6\}.$$

Next edge $(1,5)$ is selected, and the components to which vertices 1 and 5 belong are merged; the components become

$$\{1,3,4,5,6\} \quad \{2\}.$$

Next edge $(3,6)$ is examined but rejected because its vertices belong to the same component $\{1,3,4,5,6\}$. Finally, edge $(1,2)$ is selected, and the components to which vertices 1 and 2 belong are merged; the components become

$$\{1,2,3,4,5,6\},$$

and Kruskal's algorithm terminates. □

The algorithms to manage disjoint sets (see Section 3.6) can be used to handle the components. Algorithm *makeset* can be used to initialize each vertex to its own component;

$$findset(v) == findset(w)$$

can be used to test whether vertices $v$ and $w$ belong to the same component; and $union(v,w)$ can be used to merge the components to which vertices $v$ and $w$ belong. Algorithm 7.2.4 formally states Kruskal's algorithm.

**Algorithm 7.2.4 Kruskal's Algorithm.** Kruskal's algorithm finds a minimal spanning tree in a connected, weighted graph with vertex set $\{1,\ldots,n\}$. The input to the algorithm is *edgelist*, an array of *edge*, and $n$. The members of *edge* are

- $v$ and $w$, the vertices on which the edge is incident.
- *weight*, the weight of the edge.

The output lists the edges in a minimal spanning tree. The function *sort* sorts the array *edgelist* in nondecreasing order of *weight*.

Input Parameters: *edgelist*, $n$
Output Parameters: None

```
kruskal(edgelist, n) {
  sort(edgelist)
  for i = 1 to n
    makeset(i)
  count = 0
  i = 1
  while (count < n - 1) {
    if (findset(edgelist[i].v) != findset(edgelist[i].w)) {
      println(edgelist[i].v + " " + edgelist[i].w)
      count = count + 1
      union(edgelist[i].v, edgelist[i].w)
    }
    i = i + 1
  }
}
```

After Kruskal's algorithm adds $n - 1$ edges, an acyclic, $(n - 1)$-edge subgraph of the original $n$-vertex graph is obtained. By Theorem 2.6.5, the subgraph is a tree and, therefore, a spanning tree.

There are $n$ *makeset* operations, at most $2m$ *findset* operations, and $n - 1$ *union* operations. Because the graph input to Kruskal's algorithm is connected, $m \geq n - 1$. Thus the number of union and findset operations is $O(m)$. Using union by rank alone, or union by rank and path compression, these operations take time $O(m \lg m)$ (see Section 3.6). In the worst case, comparison-based sorting takes time $\Theta(m \lg m)$. Thus the worst-case time of Algorithm 7.2.4 is $\Theta(m \lg m)$.

In Section 7.1, we noted that greedy algorithms may or may not be optimal. Fortunately, Kruskal's algorithm is optimal. We deduce this from a slightly stronger result.

**Theorem 7.2.5.** *Let $G$ be a connected, weighted graph, and let $G'$ be a subgraph of a minimal spanning tree of $G$. Let $C$ be a component of $G'$, and let $S$ be the set of all edges with one vertex in $C$ and the other not in $C$. If we add a minimum weight edge in $S$ to $G'$, the resulting graph is also contained in a minimal spanning tree of $G$.*

Before proving Theorem 7.2.5, we show how it implies the correctness of Kruskal's algorithm (Algorithm 7.2.4).

**Theorem 7.2.6 Correctness of Kruskal's Algorithm.** *Kruskal's algorithm (Algorithm 7.2.4) is correct; that is, it finds a minimal spanning tree.*

**Proof.** We use induction to show that at each iteration of Kruskal's algorithm, the subgraph constructed is contained in a minimal spanning tree. It then follows that, at the termination of Kruskal's algorithm, the subgraph constructed *is* a minimal spanning tree.

When we begin, the subgraph, which consists of no edges, is contained in every minimal spanning tree. Thus the basis step is true.

Turning to the inductive step, let $G'$ denote the subgraph constructed by Kruskal's algorithm prior to another iteration of the algorithm. The inductive assumption is that $G'$ is contained in a minimal spanning tree. Let $(v, w)$ be the next edge selected by Kruskal's algorithm, and let $C$ be the component of $G'$ to which $v$ belongs. Edge $(v, w)$ is a minimum weight edge with one vertex in $C$ and one not in $C$ because it is a minimum weight edge from *any* component to any other. Therefore, by Theorem 7.2.5, when $(v, w)$ is added to $G'$, the resulting graph is also contained in a minimal spanning tree. The inductive step is complete and the theorem is proved. ■

We conclude by proving Theorem 7.2.5.

**Proof of Theorem 7.2.5.** Let $G$ be a connected, weighted graph, and let $G'$ be a subgraph of $G$ that is contained in a minimal spanning tree $T$ of $G$. Let $C$ be a component of $G'$, and let $(v, w)$ be a minimum weight edge with $v$ in $C$ and $w$ not in $C$. We must show that the graph obtained by adding $(v, w)$ to $G'$ is contained in a minimal spanning tree of $G$.

If $T$ also contains $(v, w)$, the proof is complete; so, suppose that $T$ does not contain $(v, w)$. If we add the edge $(v, w)$ to $T$ and remove an edge from the cycle $S$ created by adding $(v, w)$, the resulting subgraph $T'$ is also a spanning tree. We choose the edge to remove as follows.

Let $w'$ be the first vertex on $S$, going from $v$ to $w$, that is not in $C$, and let $v'$ be the vertex on $S$ just before $w'$ ($v'$ is in $C$) (see Figure 7.2.5). Add $(v, w)$ to $T$ and remove $(v', w')$ from $T$ to obtain $T'$. Since $(v, w)$ is a minimum weight edge with one vertex in $C$ and the other not in $C$,

$$\text{weight}(v', w') \geq \text{weight}(v, w).$$

Therefore

$$\text{weight}(T) \geq \text{weight}(T').$$

Since $T$ is a *minimal* spanning tree, we must have

$$\text{weight}(T) = \text{weight}(T').$$

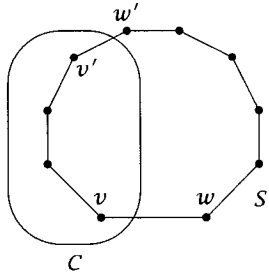


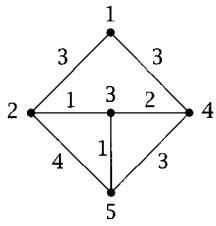

**Figure 7.2.5** The proof of Theorem 7.2.5. Vertex $w'$ is the first vertex on cycle $S$, going from $v$ to $w$, that is not in $C$. Vertex $v'$ is the vertex on $S$ just before $w'$. The spanning tree $T$ is modified by adding edge $(v, w)$ and removing edge $(v', w')$. The tree $T'$ obtained is also a minimal spanning tree.

Therefore $T'$ is a minimal spanning tree. Since $T'$ contains all of the edges of $G'$ as well as $(v, w)$, the proof is complete. ■

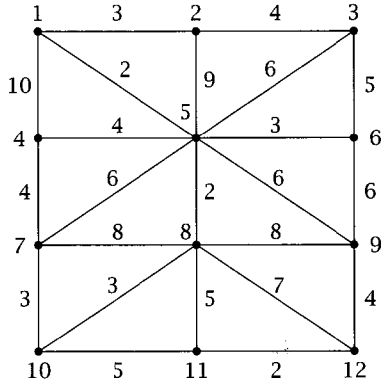
## Exercises

*Trace Kruskal's algorithm for each graph in Exercises 1–3.*

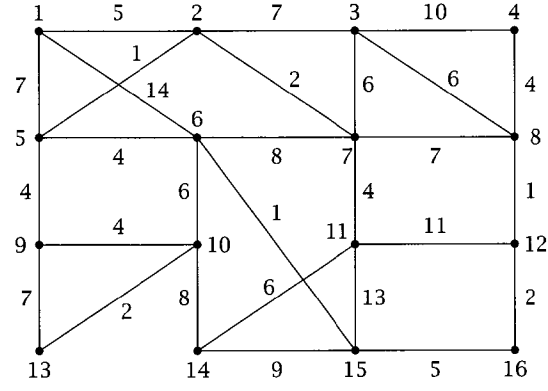
**1S.**




**2.**




**3.**




**4S.** What is the worst-case time (in terms of $n$) of Kruskal's algorithm when the input is the complete graph on $n$ vertices?

5. In Kruskal's algorithm, sorting all of the edges will be more work than necessary if not all of the edges are examined for possible inclusion in the minimal spanning tree; so, suppose that instead of sorting the edges, we place them in a binary minheap and remove them from the minheap as needed. Analyze the worst-case time of this implementation of Kruskal's algorithm in terms of the number of edges, the number of vertices, and the number of edges examined for possible inclusion in a minimal spanning tree.

6. Consider a possible divide-and-conquer approach to finding a minimal spanning tree in a connected, weighted graph $G$. Suppose that we divide the vertices of $G$ into two disjoint subsets $V_1$ and $V_2$. We then find a minimal spanning tree $T_1$ for $V_1$ and a minimal spanning tree $T_2$ for $V_2$. Finally, we find a minimum weight edge $e$ connecting $T_1$ and $T_2$. We then let $T$ be the graph obtained by combining $T_1$, $T_2$, and $e$.

   (a) Is $T$ always a spanning tree?

   (b) If $T$ is a spanning tree, is it always a minimal spanning tree?

**7S.** Let $T$ be a minimal spanning tree for a graph $G$, let $e$ be an edge in $T$, and let $T'$ be $T$ with $e$ removed. Show that $e$ is a minimum weight edge between components of $T'$.

8. Let $G$ be a connected, weighted graph, let $v$ be a vertex in $G$, and let $e$ be an edge of minimum weight incident on $v$. Show that $e$ is contained in some minimal spanning tree.

9. Let $G$ be a connected, weighted graph, and let $v$ be a vertex in $G$. Suppose that the weights of the edges incident on $v$ are distinct. Let $e$ be the edge of minimum weight incident on $v$. Show that $e$ is contained in every minimal spanning tree.

**10S.** Let $T$ and $T'$ be two spanning trees of a connected graph $G$. Suppose that an edge $e$ is in $T$ but not in $T'$. Show that there is an edge $e'$ in $T'$, but not in $T$, such that $(T - \{e\}) \cup \{e'\}$ and $(T' - \{e'\}) \cup \{e\}$ are spanning trees of $G$.

*In Exercises 11-13, tell whether the statement is true or false. If the statement is true, prove it; otherwise, give a counterexample. In each exercise, $G$ is a connected, weighted graph.*

**11S.** If all of the weights in $G$ are distinct, distinct spanning trees of $G$ have distinct weights.

12. If all of the weights in $G$ are distinct, $G$ has a unique minimal spanning tree.

13. If $e$ is an edge in $G$ whose weight is less than the weight of every other edge, $e$ is in every minimal spanning tree of $G$.