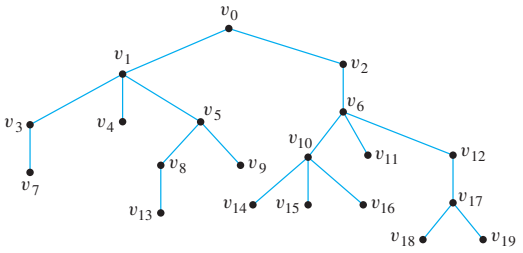
2. Consider the tree shown below with root v_0.
   a. What is the level of v_8?
   b. What is the level of v_0?
   c. What is the height of this rooted tree?
   d. What are the children of v_{10}?
   e. What is the parent of v_5?
   f. What are the siblings of v_1?
   g. What are the descendants of v_{12}?



3. Draw binary trees to represent the following expressions:
   a. $a \cdot b - (c/(d + e))$ b. $a/(b - c \cdot d)$

In each of 4–20 either draw a graph with the given specifications or explain why no such graph exists.

4. Full binary tree, five internal vertices

5. Full binary tree, five internal vertices, seven terminal vertices

6. Full binary tree, seven vertices, of which four are internal vertices

7. Full binary tree, twelve vertices

8. Full binary tree, nine vertices

9. Binary tree, height 3, seven terminal vertices

10. Full binary tree, height 3, six terminal vertices

11. Binary tree, height 3, nine terminal vertices

12. Full binary tree, eight internal vertices, seven terminal vertices.

13. Binary tree, height 4, eight terminal vertices

14. Full binary tree, seven vertices

15. Full binary tree, nine vertices, five internal vertices

16. Full binary tree, four internal vertices

17. Binary tree, height 4, eighteen terminal vertices

18. Full binary tree, sixteen vertices

19. Full binary tree, height 3, seven terminal vertices

20. What can you deduce about the height of a binary tree if you know that it has the following properties?
   a. Twenty-five terminal vertices
   b. Forty terminal vertices
   c. Sixty terminal vertices

Answers for Test Yourself

1. one vertex is distinguished from the others and is called the root; the number of edges along the unique path between it and the root; the maximum level of any vertex of the tree 2. every parent has at most two children 3. every parent has exactly two children 4. $2k + 1$; $k + 1$ 5. $t \leq 2^h$, or, equivalently, $\log_2 t \leq h$

10.7 Spanning Trees and Shortest Paths

I contend that each science is a real science insofar as it is mathematics.
— Immanuel Kant, 1724–1804

An East Coast airline company wants to expand service to the Midwest and has received permission from the Federal Aviation Authority to fly any of the routes shown in Figure 10.7.1.

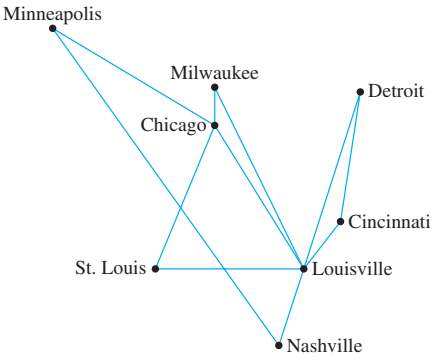


Figure 10.7.1

The company wishes to legitimately advertise service to all the cities shown but, for reasons of economy, wants to use the least possible number of individual routes to connect them. One possible route system is given in Figure 10.7.2.

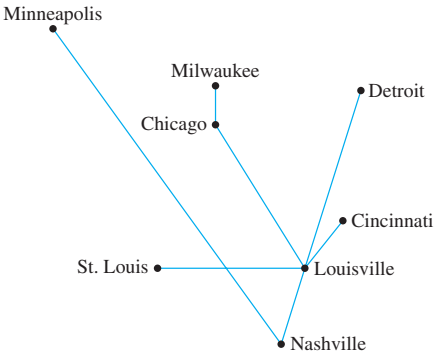


Figure 10.7.2

Clearly this system joins all the cities. Is the number of individual routes minimal? The answer is yes, and the reason may surprise you.

The fact is that the graph of any system of routes that satisfies the company's wishes is a tree, because if the graph were to contain a circuit, then one of the routes in the circuit could be removed without disconnecting the graph (by Lemma 10.5.3), and that would give a smaller total number of routes. But any tree with eight vertices has seven edges. Therefore, any system of routes that connects all eight vertices and yet minimizes the total number of routes consists of seven routes.

• Definition

A **spanning tree** for a graph G is a subgraph of G that contains every vertex of G and is a tree.

The preceding discussion contains the essence of the proof of the following proposition:

Proposition 10.7.1

1. Every connected graph has a spanning tree.
2. Any two spanning trees for a graph have the same number of edges.

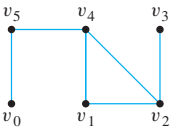
Proof of (1):

Suppose G is a connected graph. If G is circuit-free, then G is its own spanning tree and we are done. If not, then G has at least one circuit C_1. By Lemma 10.5.3, the subgraph of G obtained by removing an edge from C_1 is connected. If this subgraph is circuit-free, then it is a spanning tree and we are done. If not, then it has at least one circuit C_2, and, as above, an edge can be removed from C_2 to obtain a connected subgraph. Continuing in this way, we can remove successive edges from circuits, until eventually we obtain a connected, circuit-free subgraph T of G. *[This must happen at some point because the number of edges of G is finite, and at no stage does removal of an edge disconnect the subgraph.]* Also, T contains every vertex of G because no vertices of G were removed in constructing it. Thus T is a spanning tree for G.

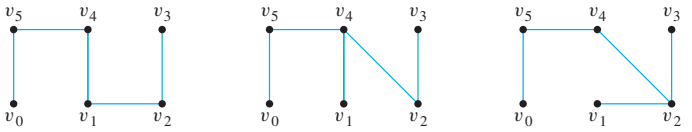
The proof of part (2) is left as an exercise.

Example 10.7.1 Spanning Trees

Find all spanning trees for the graph G pictured below.



Solution The graph G has one circuit $v_2v_1v_4v_2$, and removal of any edge of the circuit gives a tree. Thus, as shown below, there are three spanning trees for G.



Minimum Spanning Trees

The graph of the routes allowed by the Federal Aviation Authority shown in Figure 10.7.1 can be annotated by adding the distances (in miles) between each pair of cities. This is done in Figure 10.7.3.

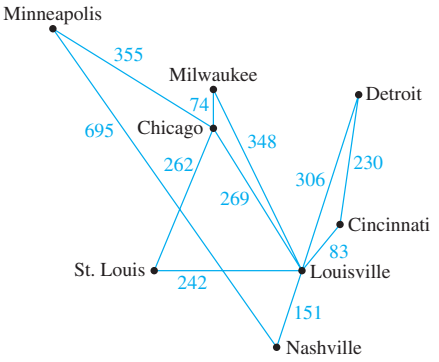


Figure 10.7.3

Now suppose the airline company wants to serve all the cities shown, but with a route system that minimizes the total mileage. Note that such a system is a tree, because if the system contained a circuit, removal of an edge from the circuit would not affect a person's ability to reach every city in the system from every other (again, by Lemma 10.5.3), but it would reduce the total mileage of the system.

More generally, a graph whose edges are labeled with numbers (known as *weights*) is called a *weighed graph*. A *minimum-weight spanning tree,* or simply a *minimum spanning tree,* is a spanning tree for which the sum of the weights of all the edges is as small as possible.

• Definition

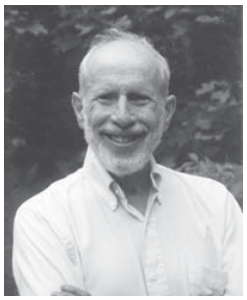
A **weighted graph** is a graph for which each edge has an associated positive real number **weight.** The sum of the weights of all the edges is the **total weight** of the graph. A **minimum spanning tree** for a connected weighted graph is a spanning tree that has the least possible total weight compared to all other spanning trees for the graph.

If G is a weighed graph and e is an edge of G, then $\boldsymbol{w(e)}$ denotes the weight of e and $\boldsymbol{w(G)}$ denotes the total weight of G.

The problem of finding a minimum spanning tree for a graph is certainly solvable. One solution is to list all spanning trees for the graph, compute the total weight of each, and choose one for which this total is a minimum. (Note that the well-ordering principle for the integers guarantees the existence of such a minimum total.) This solution, however, is inefficient in its use of computing time because the number of distinct spanning trees is so large. For instance, a complete graph with n vertices has n^{n-2} spanning trees. Even using the fastest computers available today, examining all such trees in a graph with approximately 100 vertices would require more time than is estimated to remain in the life of the universe.

In 1956 and 1957 Joseph B. Kruskal and Robert C. Prim each described much more efficient algorithms to construct minimum spanning trees. Even for large graphs, both algorithms can be implemented so as to take relatively short computing times.

Kruskal's Algorithm



Courtesy of Joseph Kruskal

Joseph Kruskal (born 1928)

In Kruskal's algorithm, the edges of a connected weighted graph are examined one by one in order of increasing weight. At each stage the edge being examined is added to what will become the minimum spanning tree, provided that this addition does not create a circuit. After $n - 1$ edges have been added (where n is the number of vertices of the graph), these edges, together with the vertices of the graph, form a minimum spanning tree for the graph.

Algorithm 10.7.1 Kruskal

Input: *G [a connected weighted graph with n vertices, where n is a positive integer]*

Algorithm Body:

[Build a subgraph T of G to consist of all the vertices of G with edges added in order of increasing weight. At each stage, let m be the number of edges of T.]

1. Initialize T to have all the vertices of G and no edges.
2. Let E be the set of all edges of G, and let $m := 0$.
3. **while** $(m < n - 1)$
   - 3a. Find an edge e in E of least weight.
   - 3b. Delete e from E.
   - 3c. **if** addition of e to the edge set of T does not produce a circuit
     then add e to the edge set of T and set $m := m + 1$

   end while

Output: *T [T is a minimum spanning tree for G.]*

The following example shows how Kruskal's algorithm works for the graph of the airline route system.

Example 10.7.2 Action of Kruskal's Algorithm

Describe the action of Kruskal's algorithm on the graph shown in Figure 10.7.4, where $n = 8$.

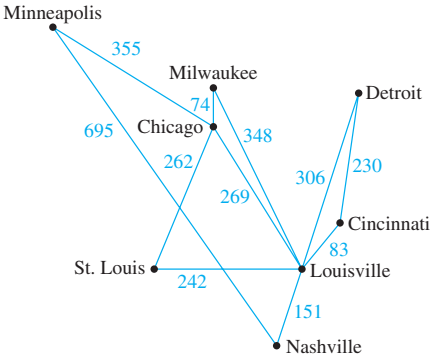


Figure 10.7.4

Solution

Iteration Number	Edge Considered	Weight	Action Taken
1	Chicago–Milwaukee	74	added
2	Louisville–Cincinnati	83	added
3	Louisville–Nashville	151	added
4	Cincinnati–Detroit	230	added
5	St. Louis–Louisville	242	added
6	St. Louis–Chicago	262	added
7	Chicago–Louisville	269	not added
8	Louisville–Detroit	306	not added
9	Louisville–Milwaukee	348	not added
10	Minneapolis–Chicago	355	added

The tree produced by Kruskal's algorithm is shown in Figure 10.7.5.

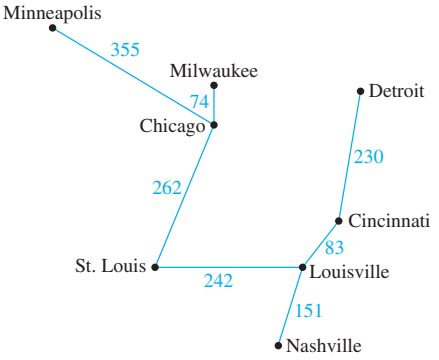


Figure 10.7.5

When Kruskal's algorithm is used on a graph in which some edges have the same weight as others, more than one minimum spanning tree can occur as output. To make

the output unique, the edges of the graph can be placed in an array and edges having the same weight can be added in the order they appear in the array.

It is not obvious from the description of Kruskal's algorithm that it does what it is supposed to do. To be specific, what guarantees that it is possible at each stage to find an edge of least weight whose addition does not produce a circuit? And if such edges can be found, what guarantees that they will all eventually connect? And if they do connect, what guarantees that the resulting tree has minimum weight? Of course, the mere fact that Kruskal's algorithm is printed in this book may lead you to believe that everything works out. But the questions above are real, and they deserve serious answers.

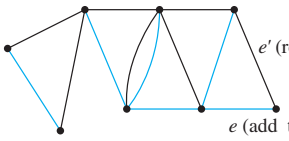
Theorem 10.7.2 Correctness of Kruskal's Algorithm

When a connected, weighted graph is input to Kruskal's algorithm, the output is a minimum spanning tree.

Proof:

Suppose that G is a connected, weighted graph with n vertices and that T is a subgraph of G produced when G is input to Kruskal's algorithm. Clearly T is circuit-free *[since no edge that completes a circuit is ever added to T].* Also T is connected. For as long as T has more than one connected component, the set of edges of G that can be added to T without creating a circuit is nonempty. *[The reason is that since G is connected, given any vertex v_1 in one connected component C_1 of T and any vertex v_2 in another connected component C_2, there is a path in G from v_1 to v_2. Since C_1 and C_2 are distinct, there is an edge e of this path that is not in T. Adding e to T does not create a circuit in T, because deletion of an edge from a circuit does not disconnect a graph and deletion of e would.]* The preceding arguments show that T is circuit-free and connected. Since by construction T contains every vertex of G, T is a spanning tree for G.

Next we show that T has minimum weight. Let T_1 be any minimum spanning tree for G such that the number of edges T_1 and T have in common is a maximum. Suppose that $T \neq T_1$. Then there is an edge e in T that is not an edge of T_1. *[Since trees T and T_1 both have the same vertex set, if they differ at all, they must have different, but same-size, edge sets.]* Now adding e to T_1 produces a graph with a unique circuit (see exercise 19 at the end of this section). Let e' be an edge of this circuit such that e' is not in T. *[Such an edge must exist because T is a tree and hence circuit-free.]* Let T_2 be the graph obtained from T_1 by removing e' and adding e. This situation is illustrated below.



The entire graph is G. T_1 has black edges. e is in T but not T_1. e' is in T_1 but not T.

Note that T_2 has $n - 1$ edges and n vertices and that T_2 is connected *[since by Lemma 10.5.3 the subgraph obtained by removing an edge from a circuit in a connected graph is connected].* Consequently, T_2 is a spanning tree for G. In addition,

$$w(T_2) = w(T_1) - w(e') + w(e).$$

Now $w(e) \leq w(e')$ because at the stage in Kruskal's algorithm when e was added to T, e' was available to be added *[since it was not already in T, and at that stage its*

addition could not produce a circuit since e was not in T], and e' *would* have been added had its weight been less than that of e. Thus

$$w(T_2) = w(T_1) - \underbrace{[w(e') - w(e)]}_{\geq 0}$$
$$\leq w(T_1).$$

But T_1 is a minimum spanning tree. Since T_2 is a spanning tree with weight less than or equal to the weight of T_1, T_2 is also a minimum spanning tree for G.

Finally, note that by construction, T_2 has one more edge in common with T than T_1 does, which contradicts the choice of T_1 as a minimum spanning tree for G with a maximum number of edges in common with T. Thus the supposition that $T \neq T_1$ is false, and hence T itself is a minimum spanning tree for G.

Prim's Algorithm

Courtesy of Alcatel-Lucent Technologies

Robert Prim (born 1921)

Prim's algorithm works differently from Kruskal's. It builds a minimum spanning tree T by expanding outward in connected links from some vertex. One edge and one vertex are added at each stage. The edge added is the one of least weight that connects the vertices already in T with those not in T, and the vertex is the endpoint of this edge that is not already in T.

Algorithm 10.7.2

Input: G *[a connected weighted graph with n vertices where n is a positive integer]*

Algorithm Body:

[Build a subgraph T of G by starting with any vertex v of G and attaching edges (with their endpoints) one by one to an as-yet-unconnected vertex of G, each time choosing an edge of least weight that is adjacent to a vertex of T.]

1. Pick a vertex v of G and let T be the graph with one vertex, v, and no edges.
2. Let V be the set of all vertices of G except v.
3. **for** $i := 1$ **to** $n - 1$
   - 3a. Find an edge e of G such that (1) e connects T to one of the vertices in V, and (2) e has the least weight of all edges connecting T to a vertex in V. Let w be the endpoint of e that is in V.
   - 3b. Add e and w to the edge and vertex sets of T, and delete w from V.

   next i

Output: T *[T is a minimum spanning tree for G.]*

The following example shows how Prim's algorithm works for the graph of the airline route system.

Example 10.7.3 Action of Prim's Algorithm

Describe the action of Prim's algorithm for the graph in Figure 10.7.6 using the Minneapolis vertex as a starting point.

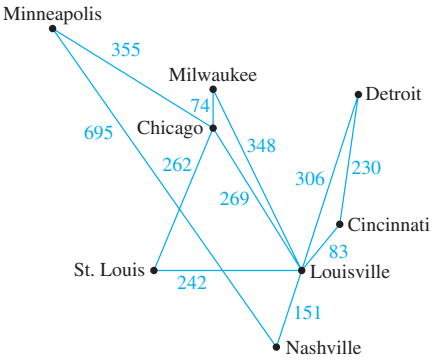


Figure 10.7.6

Solution

Iteration Number	Vertex Added	Edge Added	Weight
0	Minneapolis		
1	Chicago	Minneapolis–Chicago	355
2	Milwaukee	Chicago–Milwaukee	74
3	St. Louis	Chicago–St. Louis	262
4	Louisville	St. Louis–Louisville	242
5	Cincinnati	Louisville–Cincinnati	83
6	Nashville	Louisville–Nashville	151
7	Detroit	Cincinnati–Detroit	230

Note that the tree obtained is the same as that obtained by Kruskal's algorithm, but the edges are added in a different order.

As with Kruskal's algorithm, in order to ensure a unique output, the edges of the graph could be placed in an array and those with the same weight could be added in the order they appear in the array. It is not hard to see that when a connected graph is input to Prim's algorithm, the result is a spanning tree. What is not so clear is that this spanning tree is a minimum. The proof of the following theorem establishes that it is.

Theorem 10.7.3 Correctness of Prim's Algorithm

When a connected, weighted graph G is input to Prim's algorithm, the output is a minimum spanning tree for G.

Proof:

Let G be a connected, weighted graph, and suppose G is input to Prim's algorithm. At each stage of execution of the algorithm, an edge must be found that connects a vertex in a subgraph to a vertex outside the subgraph. As long as there are vertices outside the subgraph, the connectedness of G ensures that such an edge can always be found. *[For if one vertex in the subgraph and one vertex outside it are chosen, then by the connectedness of G there is a walk in G linking the two. As one travels along this walk, at some point one moves along an edge from a vertex inside the subgraph to a vertex outside the subgraph.]*

Now it is clear that the output T of Prim's algorithm is a tree because the edge and vertex added to T at each stage are connected to other edges and vertices of T

and because at no stage is a circuit created since each edge added connects vertices in two disconnected sets. *[Consequently, removal of a newly added edge produces a disconnected graph, whereas by Lemma 10.5.3, removal of an edge from a circuit produces a connected graph.]* Also, T includes every vertex of G because T, being a tree with $n - 1$ edges, has n vertices *[and that is all G has]*. Thus T is a spanning tree for G.

Next we show that T has minimum weight. Let T_1 be a minimum spanning tree for G such that the number of edges T_1 and T have in common is a maximum. Suppose that $T \neq T_1$. Then there is an edge e in T that is not an edge of T_1. *[Since trees T and T_1 both have the same vertex set if they differ at all, they must have different, same-size edge sets.]* Of all such edges, let e be the last that was added when T was constructed using Prim's algorithm. Let S be the set of vertices of T just before the addition of e. Then one endpoint, say v of e, is in S and the other, say w, is not. Since T_1 is a spanning tree, there is a path in T_1 joining v to w. And since $v \in S$ and $w \notin S$, as one travels along this path, one must encounter an edge e' that joins a vertex in S to one that is not in S and that therefore is not in T because e was the last edge added to T. Now at the stage when e was added to T, e' could also have been added and it *would* have been added instead of e had its weight been less than that of e. Since e' was not added at that stage, we conclude that

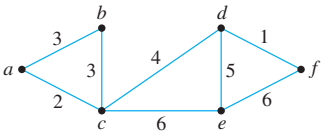
$$w(e') \geq w(e).$$

Let T_2 be the graph obtained from T_1 by removing e' and adding e. *[Thus T_2 has one more edge in common with T than T_1 does.]* Note that T_2 is a tree. The reason is that since e' is part of a path in T_1 from v to w, and e connects v and w, adding e to T_1 creates a circuit. When e' is removed from this circuit, the resulting subgraph remains connected. In fact, T_2 is a spanning tree for G since no vertices were removed in forming T_2 from T_1. The argument showing that $w(T_2) \leq w(T_1)$ is left as an exercise. *[It is virtually identical to part of the proof of Theorem 10.7.2.]* It follows that T_2 is a minimum spanning tree for G.

By construction, T_2 has one more edge in common with T than T_1, does which contradicts the choice of T_1 as a minimum spanning tree for G with a maximum number of edges in common with T. It follows that $T = T_1$, and hence T itself is a minimum spanning tree for G.

Example 10.7.4 Finding Minimum Spanning Trees

Find all minimum spanning trees for the following graph. Use Kruskal's algorithm and Prim's algorithm starting at vertex a. Indicate the order in which edges are added to form each tree.



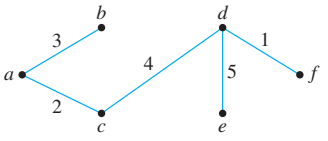
Solution When Kruskal's algorithm is applied, edges are added in one of the following two orders:

1. $\{d, f\}, \{a, c\}, \{a, b\}, \{c, d\}, \{d, e\}$
2. $\{d, f\}, \{a, c\}, \{b, c\}, \{c, d\}, \{d, e\}$

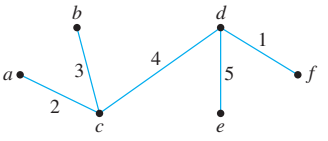
When Prim's algorithm is applied starting at a, edges are added in one of the following two orders:

1. $\{a, c\}, \{a, b\}, \{c, d\}, \{d, f\}, \{d, e\}$
2. $\{a, c\}, \{b, c\}, \{c, d\}, \{d, f\}, \{d, e\}$

Thus, as shown below, there are two distinct minimum spanning trees for this graph.



(a)



(b)

Dijkstra's Shortest Path Algorithm

Although the trees produced by Kruskal's and Prim's algorithms have the least possible total weight compared to all other spanning trees for the given graph, they do not always reveal the shortest distance between any two points on the graph. For instance, according to the complete route system shown in Figure 10.7.3, one can fly directly from Nashville to Minneapolis for a distance of 695 miles, whereas if you use the minimum spanning tree shown in Figure 10.7.5 the only way to fly from Nashville to Minneapolis is by going through Louisville, St. Louis, and Chicago, which gives a total distance of $151 + 242 + 262 + 355 = 1,010$ miles and the unpleasantness of three changes of plane.

In 1959 the computing pioneer, Edsgar Dijkstra (see Section 5.5), developed an algorithm to find the shortest path between a starting vertex and an ending vertex in a weighted graph in which all the weights are positive. It is somewhat similar to Prim's algorithm in that it works outward from a starting vertex a, adding vertices and edges one by one to construct a tree T. However, it differs from Prim's algorithm in the way it chooses the next vertex to add, ensuring that for each added vertex v, the length of the shortest path from a to v has been identified.

At the start of execution of the algorithm, each vertex u of G is given a label $L(u)$, which indicates the current best estimate of the length of the shortest path from a to u. $L(a)$ is initially set equal to 0 because the shortest path from a to a has length zero, but, because there is no previous information about the lengths of the shortest paths from a to any other vertices of G, the label $L(u)$ of each vertex u other than a is initially set equal to a number, denoted ∞, that is greater than the sum of the weights of all the edges of G. As execution of the algorithm progresses, the values of $L(u)$ are changed, eventually becoming the actual lengths of the shortest paths from a to u in G.

Because T is built up outward from a, at each stage of execution of the algorithm the only vertices that are candidates to join T are those that are adjacent to at least one vertex of T. Thus at each stage of Dijkstra's algorithm, the graph G can be thought of as divided into three parts: the tree T that is being built up, the set of "fringe" vertices that are adjacent to at least one vertex of the tree, and the rest of the vertices of G. Each fringe vertex is a candidate to be the next vertex added to T. The one that is chosen is the one for which the length of the shortest path to it from a through T is a minimum among all the vertices in the fringe.

An essential observation underlying Dijkstra's algorithm is that after each addition of a vertex v to T, the only fringe vertices for which a shorter path from a might be found are those that are adjacent to v *[because the length of the path from a to v was a minimum among all the paths from a to vertices in what was then the fringe].* So after each addition of a vertex v to T, each fringe vertex u adjacent to v is examined and two numbers are

compared: the current value of $L(u)$ and the value of $L(v) + w(v, u)$, where $L(v)$ is the length of the shortest path to v (in T) and $w(v, u)$ is the weight of the edge joining v and u. If $L(v) + w(v, u) < L(u)$, then the value of $L(u)$ is changed to $L(v) + w(v, u)$.

At the beginning of execution of the algorithm, the tree consists only of the vertex a, and $L(a) = 0$. When execution terminates, $L(z)$ is the length of a shortest path from a to z.

As with Kruskal's and Prim's algorithms for finding minimum spanning trees, there is a simple but dramatically inefficient way to find the shortest path from a to z: compute the lengths of all the paths and choose one that is shortest. The problem is that even for relatively small graphs using this method to find a shortest path could require billions of years, whereas Dijkstra's algorithm could do the job in a few seconds.

Algorithm 10.7.3 Dijkstra

Input: *G [a connected simple graph with a positive weight for every edge], ∞ [a number greater than the sum of the weights of all the edges in the graph], $w(u, v)$ [the weight of edge $\{u, v\}$], a [the starting vertex], z [the ending vertex]*

Algorithm Body:

1. Initialize T to be the graph with vertex a and no edges. Let $V(T)$ be the set of vertices of T, and let $E(T)$ be the set of edges of T.
2. Let $L(a) = 0$, and for all vertices in G except a, let $L(u) = \infty$.
   [The number $L(x)$ is called the label of x.]
3. Initialize v to equal a and F to be $\{a\}$.
   [The symbol v is used to denote the vertex most recently added to T.]
4. **while** $(z \notin V(T))$

   4a. $F := (F - \{v\}) \cup \{$vertices that are adjacent to v and are not in $V(T)\}$
   [The set F is called the fringe. Each time a vertex is added to T, it is removed from the fringe and the vertices adjacent to it are added to the fringe if they are not already in the fringe or the tree T.]

   4b. For each vertex u that is adjacent to v and is not in $V(T)$,
   if $L(v) + w(v, u) < L(u)$ **then**

$$L(u) := L(v) + w(v, u)$$

$$D(u) := v$$

   [Note that adding v to T does not affect the labels of any vertices in the fringe F except those adjacent to v. Also, when $L(u)$ is changed to a smaller value, the notation $D(u)$ is introduced to keep track of which vertex in T gave rise to the smaller value.]

   4c. Find a vertex x in F with the smallest label
   Add vertex x to $V(T)$, and add edge $\{D(x), x\}$ to $E(T)$
   $v := x$ *[This statement sets up the notation for the next iteration of the loop.]*

   end while

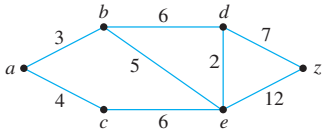
Output: *$L(z)$ [$L(z)$, a nonnegative integer, is the length of the shortest path from a to z.]*

Note The unique path in the tree T from a to z is the shortest path in G from a to z.

The action of Dijkstra's algorithm is illustrated by the flow of the drawings in Example 10.7.5.

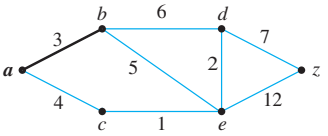
Example 10.7.5 Action of Dijkstra's Algorithm

Show the steps in the execution of Dijkstra's shortest path algorithm for the graph shown below with starting vertex a and ending vertex z.



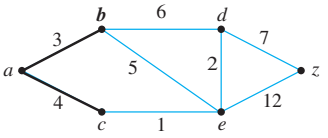
Solution

Step 1: Going into the **while** loop: $V(T) = \{a\},\ E(T) = \emptyset$, and $F = \{a\}$



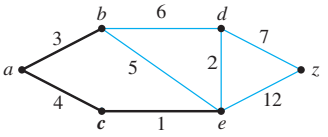
During iteration:
$F = \{b, c\},\ L(b) = 3,\ L(c) = 4.$
Since $L(b) < L(c),\ b$ is added to $V(T)$ and $\{a, b\}$ is added to $E(T)$.

Step 2: Going into the **while** loop: $V(T) = \{a, b\},\ E(T) = \{\{a, b\}\}$



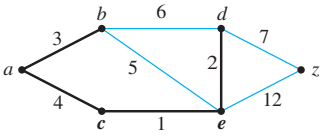
During iteration:
$F = \{c, d, e\},\ L(c) = 4,\ L(d) = 9,$
$L(e) = 8.$
Since $L(c) < L(d)$ and $L(c) < L(e),\ c$ is added to $V(T)$ and $\{a, c\}$ is added to $E(T)$.

Step 3: Going into the **while** loop: $V(T) = \{a, b, c\},\ E(T) = \{\{a, b\},\ \{a, c\}\}$



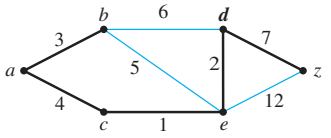
During iteration:
$F = \{d, e\},\ L(d) = 9,\ L(e) = 5$
$L(e)$ becomes 5 because *ace*, which has length 5, is a shorter path to e than *abe*, which has length 8.
Since $L(e) < L(d),\ e$ is added to $V(T)$ and $\{c,\ e\}$ is added to $E(T)$.

Step 4: Going into the **while** loop: $V(T) = \{a,\ b,\ c,\ e\},$
$E(T) = \{\{a,\ b\},\ \{a,\ c\},\ \{c,\ e\}\}$



During iteration:
$F = \{d, z\},\ L(d) = 7,\ L(z) = 17$
$L(d)$ becomes 7 because *aced*, which has length 7, is a shorter path to d than *abd*, which has length 9.
Since $L(d) < L(z),\ d$ is added to $V(T)$ and $\{e, d\}$ is added to $E(T)$.

Step 5: Going into the **while** loop: $V(T) = \{a, b, c, e, d\},$
$E(T) = \{\{a, b\},\{a, c\},\ \{\mathrm{c}, \mathrm{e}\},\ \{\mathrm{e}, \mathrm{d}\}\}$



During iteration: $F = \{z\},\ L(z) = 14$
$L(z)$ becomes 14 because *acedz*, which has length 14, is a shorter path to d than *abdz*, which has length 17.
Since z is the only vertex in F, its label is a minimum, and so z is added to $V(T)$ and $\{e, z\}$ is added to $E(T)$.

Execution of the algorithm terminates at this point because $z \in V(T)$. The shortest path from a to z has length $L(z) = 14$. ■

Keeping track of the steps in a table is a convenient way to show the action of Dijkstra's algorithm. Table 10.7.1 does this for the graph in Example 10.7.5.

Table 10.7.1

Step	$V(T)$	$E(T)$	F	$L(a)$	$L(b)$	$L(c)$	$L(d)$	$L(e)$	$L(z)$
0	$\{a\}$	$\emptyset$	$\{a\}$	**0**	∞	∞	∞	∞	∞
1	$\{a\}$	$\emptyset$	$\{b,\ c\}$	0	**3**	4	∞	∞	∞
2	$\{a, b\}$	$\{\{a, b\}\}$	$\{c, d, e\}$	0	3	**4**	9	8	∞
3	$\{a, b, c\}$	$\{\{a, b\},\ \{a, c\}\}$	$\{d, e\}$	0	3	4	9	**5**	∞
4	$\{a, b, c, e\}$	$\{\{a, b\},\ \{a, c\},\ \{c, e\}\}$	$\{d, z\}$	0	3	4	**7**	5	17
5	$\{a, b, c, e, d\}$	$\{\{a, b\},\ \{a, c\},\ \{c, e\},\ \{e, d\}\}$	$\{z\}$	0	3	4	7	5	**14**
6	$\{a, b, c, e, d, z\}$	$\{\{a, b\},\ \{a, c\},\ \{c, e\},\ \{e, d\},\ \{e, z\}\}$							

It is clear that Dijkstra's algorithm keeps adding vertices to I until it has added z. The proof of the following theorem shows that when the algorithm terminates, the label z goes the length of the shortest path to it from a.

Theorem 10.7.4 Correctness of Dijkstra's Algorithm

When a connected, simple graph with a positive weight for every edge is input to Dijkstra's algorithm with starting vertex a and ending vertex z, the output is the length of a shortest path from a to z.

Proof:

Let G be a connected, weighted graph with no loops or parallel edges and with a positive weight for every edge. Let T be the graph built up by Dijkstra's algorithm, and for each vertex u in G, let $L(u)$ be the label given by the algorithm to vertex u. For each integer $n \geq 0$, let the property $P(n)$ be the sentence

> After the nth iteration of the while loop in Dijkstra's algorithm, (1) T is a tree, and (2) for every vertex v in T, $L(v)$ is the length of a shortest path in G from a to v. ← $P(n)$

We will show by mathematical induction that $P(n)$ is true for all integers n from 0 through the termination of the algorithm.

Show that P(0) is true: When $n = 0$, the graph T is a tree because it is defined to consist only of the vertex a and no edges. In addition, $L(a)$ is the length of the shortest path from a to a because the initial value of $L(a)$ is 0.

Show that for all integers k ≥ 0, if P(k) is true then P(k + 1) is also true: Let k be any integer with $k \geq 0$ and suppose that

> After the kth iteration of the while loop in Dijkstra's algorithm, (1) T is a tree, and (2) for every vertex v in T, $L(v)$ is the length of a shortest path in G from a to v. ← $P(k)$ inductive hypothesis

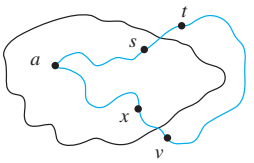
We must show that

> After the $(k + 1)$st iteration of the **while** loop in Dijkstra's algorithm, (1) T is a tree, and (2) for every vertex v in T, $L(v)$ is the length of a shortest path in G from a to v. ← $P(k + 1)$

continued on page 714

So suppose that after the $(k+1)$st iteration of the **while** loop in Dikjstra's algorithm, the vertex v and edge $\{x, v\}$ have been added to T, where x is in $V(T)$. Clearly the new value of T is a tree because adding a new vertex and edge to a tree does not create a circuit and does not disconnect the tree. By inductive hypothesis for each vertex y in the tree before the addition of v, $L(y)$ is the length of a shortest path from a to y. So it remains only to show that $L(v)$ is the length of a shortest path from a to v.

Now, according to the algorithm, the final value of $L(v) = L(x) + w(x, v)$. Consider *any* shortest path from a to v, and let $\{s, t\}$ be the first edge in this path to leave T, where $s \in V(T)$ and $t \notin V(T)$. This situation is illustrated below.



Let $LSP(a, v)$ be the length of a shortest path from a to v, and let $LSP(a, s)$ be the length of a shortest path from a to s. Observe that

$$LSP(a, v) \geq LSP(a, s) + w(s, t)$$ because the path from t to v has length ≥ 0

$$\geq L(s) + w(s, t)$$ by inductive hypothesis because s is a vertex in T

$$\geq L(x) + w(x, v)$$ t is in the fringe of the tree, and so if $L(s)+w(s, t)$ were less than $L(x)+w(x, v)$ then t would have been added to T instead of v.

On the other hand

$$L(x) + w(x, v) \geq LSP(a, v)$$ because $L(x)+w(x, v)$ is the length of a path from a to v and so it is greater than or equal to the length of the shortest path from a to v.

It follows that $$LSP(a, v) = L(x) + w(x, v),$$

and, since $$L(v) = L(x) + w(x, v),$$

$L(v)$ is the length of a shortest path from a to v. This completes the proof by mathematical induction.

The algorithm terminates as soon as z is in T, and, since we have proved that the label of every vertex in the tree gives the length of the shortest path to it from a, then, in particular, $L(z)$ is the length of a shortest path from a to z.

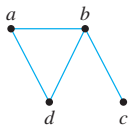
Test Yourself

1. A spanning tree for a graph G is _____.
2. A weighted graph is a graph for which _____, and the total weight of the graph is _____.
3. A minimum spanning tree for a connected, weighted graph is _____.
4. In Kruskal's algorithm, the edges of a connected, weighted graph are examined one by one in order of _____ starting with _____.
5. In Prim's algorithm, a minimum spanning tree is built by expanding outward from an _____ in a sequence of ____.
6. In Dijkstra's algorithm, a vertex is in the fringe if it is _____ vertex in the tree that is being built up.
7. At each stage of Dijkstra's algorithm, the vertex that is added to the tree is a vertex in the fringe whose label is a _____.

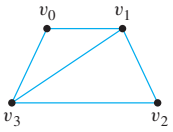
Exercise Set 10.7

Find all possible spanning trees for each of the graphs in 1 and 2.

1.

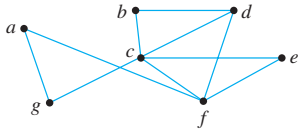


2.

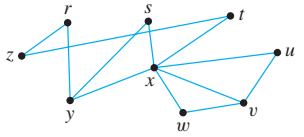


Find a spanning tree for each of the graphs in 3 and 4.

3.

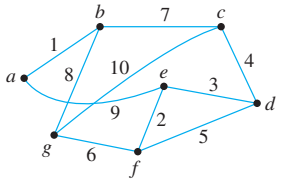


4.

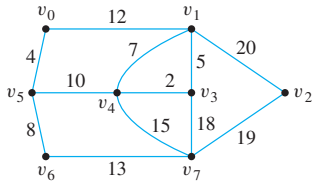


Use Kruskal's algorithm to find a minimum spanning tree for each of the graphs in 5 and 6. Indicate the order in which edges are added to form each tree.

5.



6.

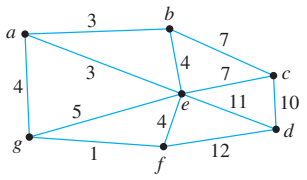


Use Prim's algorithm starting with vertex a or v_0 to find a minimum spanning tree for each of the graphs in 7 and 8. Indicate the order in which edges are added to form each tree.

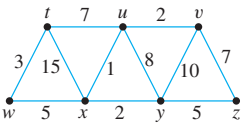
7. The graph of exercise 5. 8. The graph of exercise 6.

For each of the graphs in 9 and 10, find all minimum spanning trees that can be obtained using (a) Kruskal's algorithm and (b) Prim's algorithm starting with vertex a or t. Indicate the order in which edges are added to form each tree.

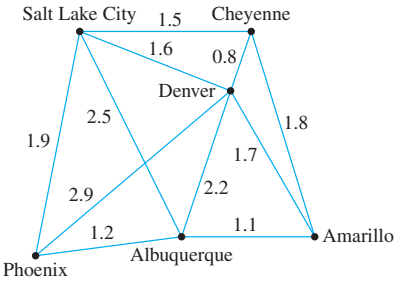
9.



10.



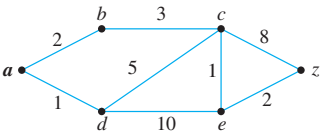
11. A pipeline is to be built that will link six cities. The cost (in hundreds of millions of dollars) of constructing each potential link depends on distance and terrain and is shown in the weighted graph below. Find a system of pipelines to connect all the cities and yet minimize the total cost.



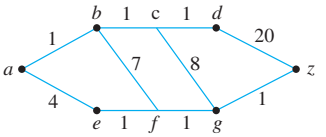
12. Use Dijkstra's algorithm for the airline route system of Figure 10.7.3 to find the shortest distance from Nashville to Minneapolis. Make a table similar to Table 10.7.1 to show the action of the algorithm.

Use Dijkstra's algorithm to find the shortest path from a to z for each of the graphs in 13–16. In each case make tables similar to Table 10.7.1 to show the action of the algorithm.

13.



14.



15. The graph of exercise 9 with $a = a$ and $z = f$

16. The graph of exercise 10 with $a = u$ and $z = w$

17. Prove part (2) of Proposition 10.7.1: Any two spanning trees for a graph have the same number of edges.

18. Given any two distinct vertices of a tree, there exists a unique path from one to the other.
    a. Give an informal justification for the above statement.
    ✶ **b.** Write a formal proof of the above statement.

19. Prove that if G is a graph with spanning tree T and e is an edge of G that is not in T, then the graph obtained by adding e to T contains one and only one set of edges that forms a circuit.

20. Suppose G is a connected graph and T is a circuit-free subgraph of G. Suppose also that if any edge e of G not in T is added to T, the resulting graph contains a circuit. Prove that T is a spanning tree for G.

21. **a.** Suppose T_1 and T_2 are two different spanning trees for a graph G. Must T_1 and T_2 have an edge in common? Prove or give a counterexample.
    b. Suppose that the graph G in part (a) is simple. Must T_1 and T_2 have an edge in common? Prove or give a counterexample.

***H* 22.** Prove that an edge e is contained in every spanning tree for a connected graph G if, and only if, removal of e disconnects G.

23. Consider the spanning trees T_1 and T_2 in the proof of Theorem 10.7.3. Prove that $w(T_2) \leq w(T_1)$.

24. Suppose that T is a minimum spanning tree for a connected, weighted graph G and that G contains an edge e (not a loop) that is not in T. Let v and w be the endpoints of e. By exercise 18 there is a unique path in T from v to w. Let e' be any edge of this path. Prove that $w(e') \leq w(e)$.

***H* 25.** Prove that if G is a connected, weighted graph and e is an edge of G (not a loop) that has smaller weight than any other edge of G, then e is in every minimum spanning tree for G.

***✶* 26.** If G is a connected, weighted graph and no two edges of G have the same weight, does there exist a unique minimum spanning tree for G? Use the result of exercise 19 to help justify your answer.

***✶* 27.** Prove that if G is a connected, weighted graph and e is an edge of G that (1) has greater weight than any other edge of G and (2) is in a circuit of G, then there is no minimum spanning tree T for G such that e is in T.

28. Suppose a disconnected graph is input to Kruskal's algorithm. What will be the output?

29. Suppose a disconnected graph is input to Prim's algorithm. What will be the output?

30. Prove that if a connected, weighted graph G is input to Algorithm 10.7.4 (shown below), the output is a minimum spanning tree for G.

Algorithm 10.7.4

Input: G *[a connected graph]*

Algorithm Body:

1. $T := G$.

2. $E :=$ the set of all edges of G, $m :=$ the number of edges of G.

3. while $(m > 0)$

3a. Find an edge e in E that has maximal weight.
3b. Remove e from E and set $m := m - 1$.
3c. if the subgraph obtained when e is removed from the edge set of T is connected **then** remove e from the edge set of T

end while

Output: T *[a minimum spanning tree for G]*

31. Modify Algorithm 10.7.3 so that the output consists of the sequence of edges in the shortest path from a to z.

Answers for Test Yourself

1. a subgraph of G that contains every vertex of G and is a tree. 2. each edge has an associated positive real number weight; the sum of the weights of all the edges of the graph 3. a spanning tree that has the least possible total weight compared to all other spanning trees for the graph 4. weight; an edge of least weight 5. initial vertex; adjacent vertices and edges 6. adjacent to a 7. minimum among all those in the fringe