CHAPTER 10

THEORY OF GRAPHS AND TREES

10.6 Spanning Trees and a Shortest Path Algorithm

Spanning Trees and a Shortest Path Algorithm

Definition

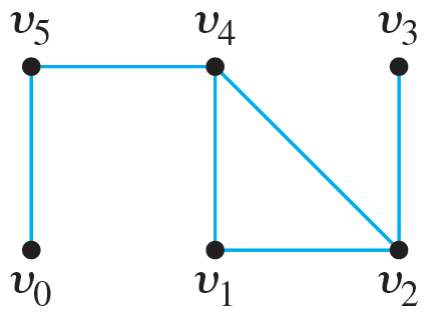
A **spanning tree** for a graph G is a subgraph of G that contains every vertex of G and is a tree.

Proposition 10.6.1

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

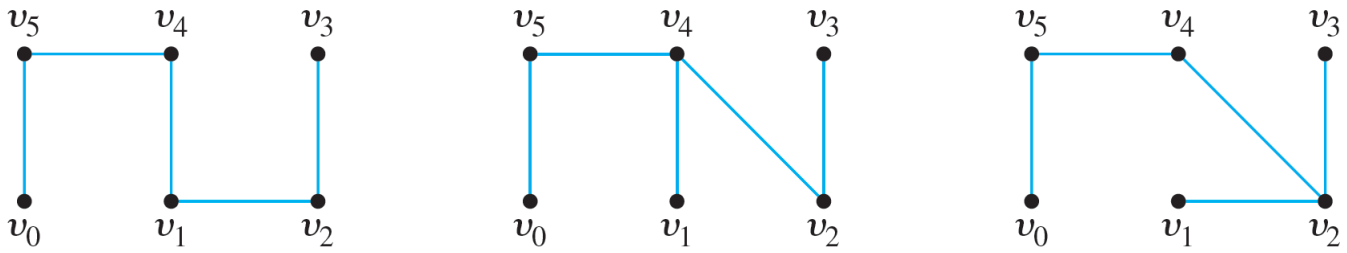
Example 10.6.1 – *Spanning Trees*

Find all spanning trees for the graph *G* pictured below.



Example 10.6.1 – *Solution*

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



Minimum Spanning Trees

Minimum Spanning Trees (1/4)

The graph of the routes allowed by the U.S. Federal Aviation Authority shown in Figure 10.6.1 can be annotated by adding the distances (in miles) between each pair of cities.

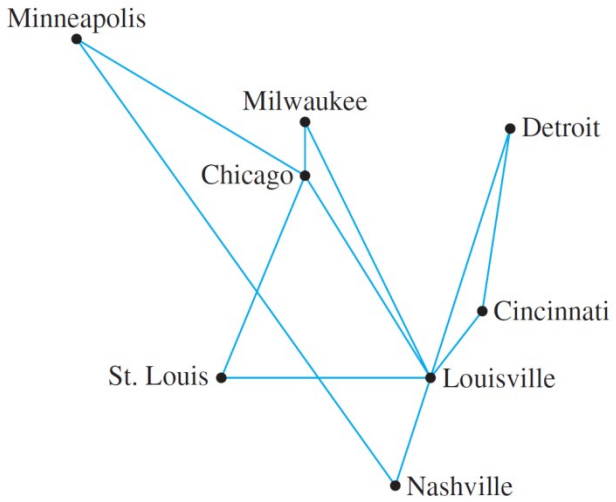


Figure 10.6.1

Minimum Spanning Trees (2/4)

This is done in Figure 10.6.3.

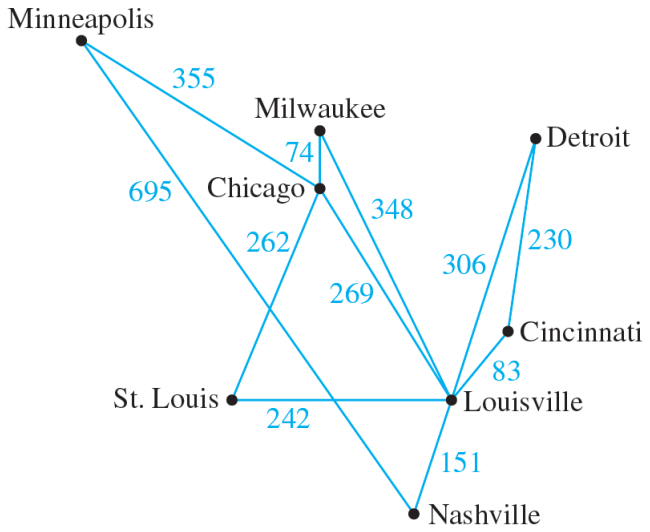


Figure 10.6.3

Minimum Spanning Trees (3/4)

Now suppose the airline company wants to serve all the cities shown, but with a route system that minimizes the total mileage of the system as a whole.

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, but it would reduce the total mileage of the system.

Minimum Spanning Trees (4/4)

More generally, a graph whose edges are labeled with numbers (known as *weights*) is called a *weighted 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 and Notation

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.

Kruskal’s Algorithm

Kruskal's Algorithm (1/3)

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.

Kruskal's Algorithm (2/3)

Algorithm 10.6.1 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 the 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.]*

Example 10.6.2 – *Action of Kruskal's Algorithm*

Describe the action of Kruskal's algorithm on the graph shown in Figure 10.6.4, where $n = 8$.

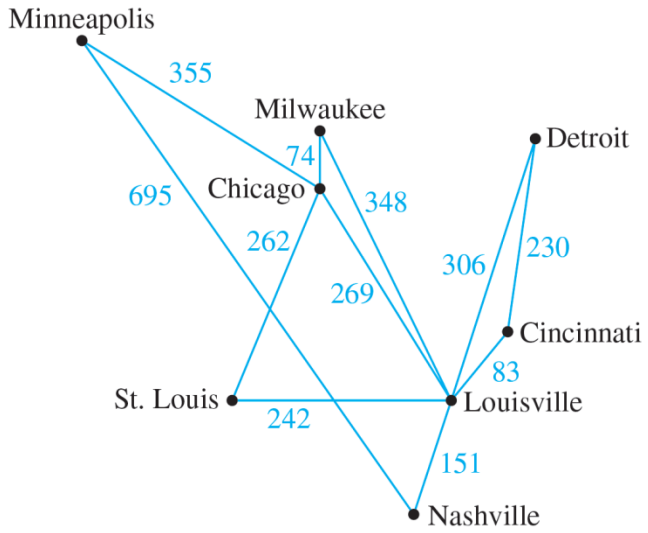


Figure 10.6.4

Example 10.6.2 – *Solution (1/2)*

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

Example 10.6.2 – *Solution (2/2)* continued

The tree produced by Kruskal's algorithm is shown in Figure 10.6.5.

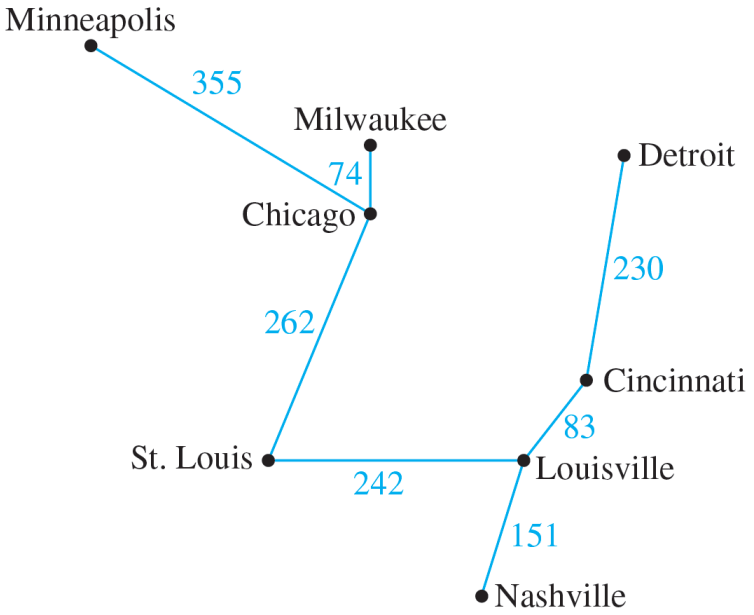


Figure 10.6.5

Kruskal's Algorithm (3/3)

Theorem 10.6.2 Correctness of Kruskal's Algorithm

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

Prim’s Algorithm

Prim's Algorithm (1/3)

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*.

Prim's Algorithm (2/3)

Algorithm 10.6.2

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.]*

Example 10.6.3 – *Action of Prim's Algorithm*

Describe the action of Prim's algorithm for the graph in Figure 10.6.6 using the Minneapolis vertex as a starting point.

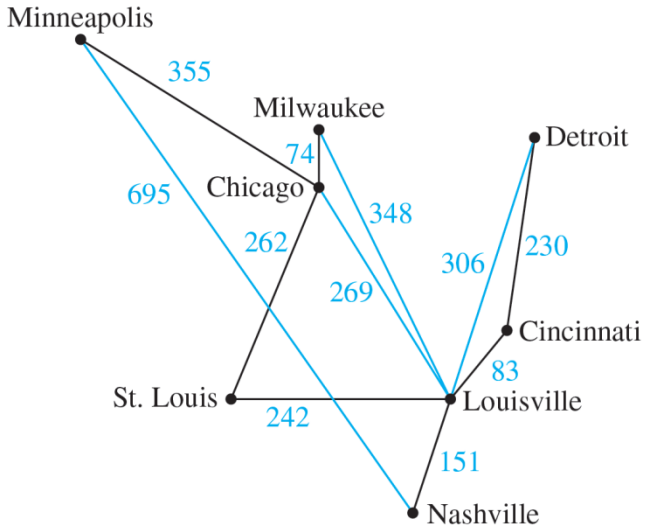


Figure 10.6.6

Example 10.6.3 – *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

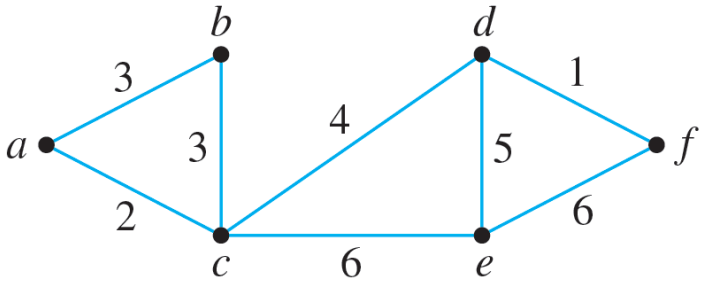
Prim's Algorithm (3/3)

Theorem 10.6.3 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.

Example 10.6.4 – *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.



Example 10.6.4 – *Solution (1/2)*

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

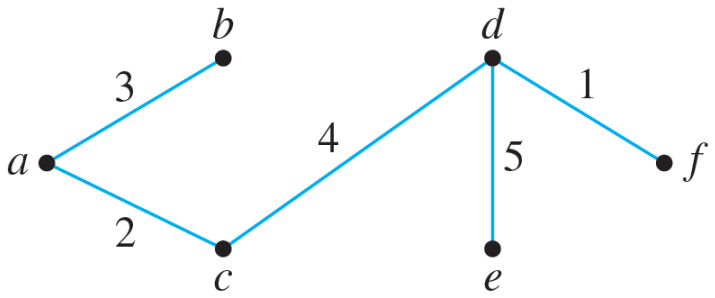
1. {*d*, *f* }, {*a*, *c*}, {*a*, *b*}, {*c*, *d*}, {*d*, *e*}
2. {*d*, *f*}, {*a*, *c*}, {*b*, *c*}, {*c*, *d*}, {*d*, *e*}

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

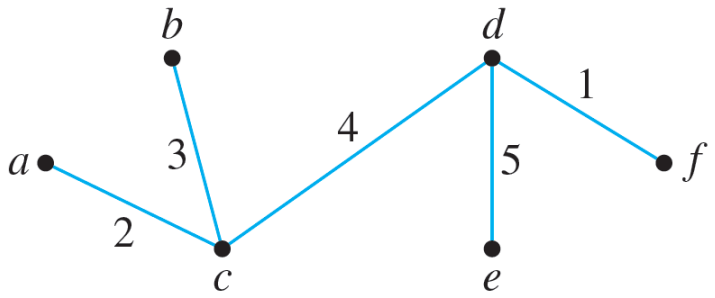
1. {*a*, *c*}, {*a*, *b*}, {*c*, *d*}, {*d*, *f*}, {*d*, *e*}
2. {*a*, *c*}, {*b*, *c*}, {*c*, *d*}, {*d*, *f*}, {*d*, *e*}

Example 10.6.4 – *Solution (2/2)* continued

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



(a)



(b)

Dijkstra’s Shortest Path Algorithm

Dijkstra's Shortest Path Algorithm (1/5)

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.

In 1959 the computing pioneer, Edsger Dijkstra 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.

Dijkstra's Shortest Path Algorithm (2/5)

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.

Dijkstra's Shortest Path Algorithm (3/5)

Algorithm 10.6.3 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.]

Dijkstra's Shortest Path Algorithm (4/5)

Algorithm 10.6.3 Dijkstra

4. **while** $(z \notin V(T))$

4a. $F := (F - \{v\}) \cup \{\text{vertices that are adjacent to } v \text{ and are not in } V(T)\}$
[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.]

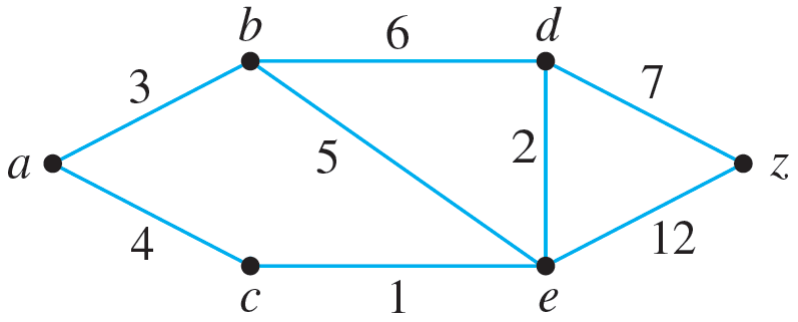
4c. Find a vertex x in F with the smallest label
Add vertex x to $V(T)$, and add edge $\{D(x), x\}$ to $E(T)$
$v := x$ *[This statement sets up the notation for the next iteration of the loop.]*

end while

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

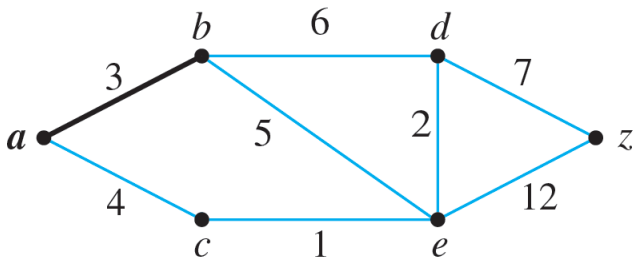
Example 10.6.5 – *Action of Dijkstra's Algorithm*

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*.



Example 10.6.5 – *Solution (1/7)*

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



During iteration:

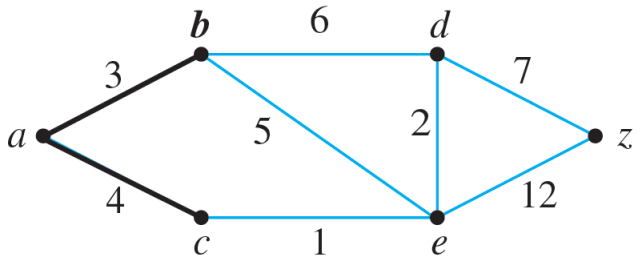
$F = \{b, c\}$, $L(b) = 3$, $L(c) = 4$.

Since $L(b) < L(c)$, b is added to $V(T)$, $D(b) = a$, and $\{a, b\}$ is added to $E(T)$.

Example 10.6.5 – *Solution (2/7)* continued

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

$E(T) = \{\{a, b\}\}$



During iteration:

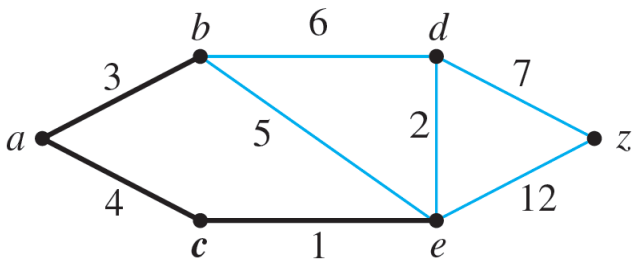
$F = \{c, d, e\}$, $L(c) = 4$, $L(d) = 9$, $L(e) = 8$.

Since $L(c) < L(d)$ and $L(c) < L(e)$, c is added to $V(T)$, $D(c) = a$, and $\{a, c\}$ is added to $E(T)$.

Example 10.6.5 – *Solution (3/7)* continued

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

$E(T) = \{\{a, b\}, \{a, c\}\}$



During iteration:

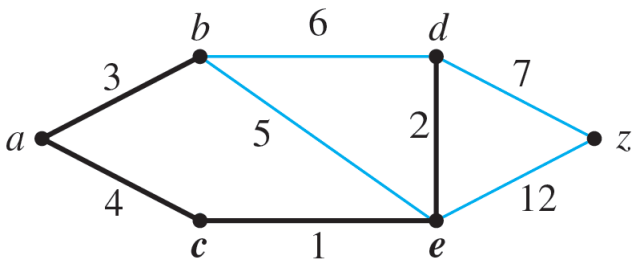
$F = \{d, e\}$, $L(d) = 9$, $L(e) = 5$
$L(e)$ becomes 5 because *ace*, which has length 5, is a shorter path to *e* than *abe*, which has length 8.

Since $L(e) < L(d)$, *e* is added to $V(T)$, $D(e) = c$, and $\{c, e\}$ is added to $E(T)$.

Example 10.6.5 – *Solution (4/7)* continued

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

$E(T) = \{\{a, b\}, \{a, c\}, \{c, e\}\}$



During iteration:

$F = \{d, z\}$, $L(d) = 7$, $L(z) = 17$

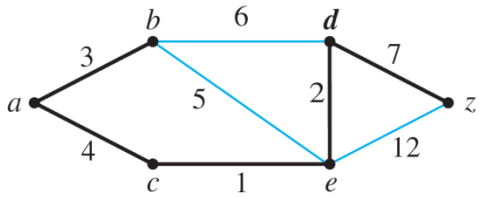
$L(d)$ becomes 7 because *aced*, which has length 7, is a shorter path to d than abd, which has length 9.

Since $L(d) < L(z)$, d is added to $V(T)$, $D(d) = e$, and $\{e, d\}$ is added to $E(T)$.

Example 10.6.5 – *Solution (5/7)* continued

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

$E(T) = \{\{a, b\}, \{a, c\}, \{c, e\}, \{e, d\}\}$



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 $V(T)$, $D(z) = d$, and $\{d, z\}$ is added to $E(T)$.

Example 10.6.5 – *Solution (6/7)* continued

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 the algorithm. Table 10.6.1 does this for the graph in Example 10.6.5.

Step	$V(T)$	$E(T)$	F	$L(a)$	$L(b)$	$L(c)$	$L(d)$	$L(e)$	$L(z)$
0	$\{a\}$	$\varnothing$	$\{a\}$	**0**	∞	∞	∞	∞	∞
1	$\{a\}$	$\varnothing$	$\{b, c\}$	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	$\{a, b, c, e, d, z\}$	$\{\{a, b\}, \{a, c\}, \{c, e\}, \{e, d\}, \{e, z\}\}$							

Table 10.6.1

Example 10.6.5 – *Solution (7/7)* continued

In step 1, $D(b) = a$; in step 2, $D(c) = a$; in step 3, $D(e) = c$; in step 4, $D(d) = e$; and in step 5, $D(z) = e$.

Working backward gives the vertices in the shortest path. Because $D(z) = d$, $D(d) = e$, $D(e) = c$, and $D(c) = a$, the shortest path from a to z is $acedz$.

Dijkstra's Shortest Path Algorithm (5/5)

Theorem 10.6.4 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.