9. PRIM'S ALGORITHM

A spanning tree of an undirected connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph. If such a graph has weights assigned to its edges, a minimum spanning tree is its spanning tree of the smallest weight, where the *weight* of a tree is defined as the sum of the weights on all its edges.

The minimum spanning tree problem is the problem of finding a minimum spanning tree for a given weighted connected graph.

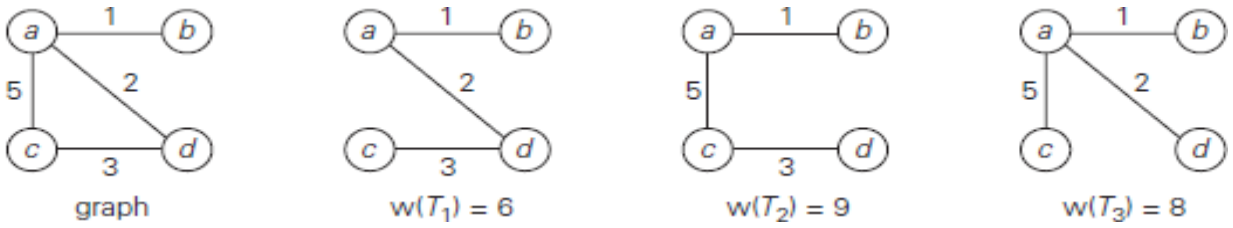


FIGURE 3.13 Graph and its spanning trees, with *T*1 being the minimum spanning tree.

The minimum spanning tree is illustrated in Figure 3. If we were to try constructing a minimum spanning tree by exhaustive search, we would face two serious obstacles.

First, the number of spanning trees grows exponentially with the graph size (at least for dense graphs).

Second, generating all spanning trees for a given graph is not easy; in fact, it is more difficult than finding a minimum spanning tree for a weighted graph.

Prim's algorithm constructs a minimum spanning tree through a sequence of expanding subtrees. The initial subtree in such a sequence consists of a single vertex selected arbitrarily from the set V of the graph's vertices.

On each iteration, the algorithm expands the current tree in the greedy manner by simply attaching to it the nearest vertex not in that tree. The algorithm stops after all the graph's vertices have been included in the tree being constructed

ALGORITHM *Prim(G)*

//Prim's algorithm for constructing a minimum spanning tree

//Input: A weighted connected graph $G = \{V, E\}$

//Output: E_T, the set of edges composing a minimum spanning tree of G $V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex $E_T \leftarrow \Phi$

for $i \leftarrow 1$ **to** $|V| - 1$ **do**

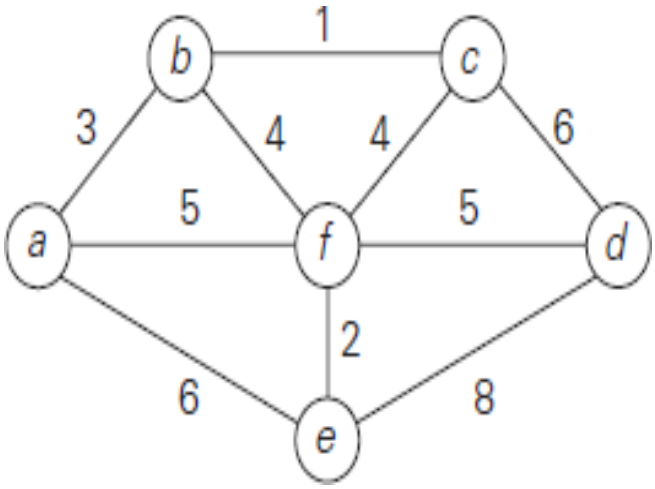
find a minimum-weight edgee*=(v*,u*)among all the edges(v,u) such that v is in V_T and u is in $V - V_T$

$V_T \leftarrow V_T \cup \{u^*\}$

$E_T \leftarrow E_T \cup \{e^*\}$

return E_T

If a graph is represented by its adjacency lists and the priority queue is implemented as a min-heap, the running time of the algorithm is O(|E| log |V |) in a connected graph, where $|V| - 1 \leq$
$|E|$.



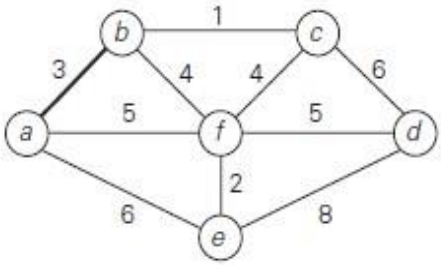
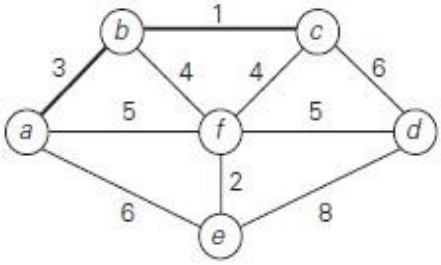
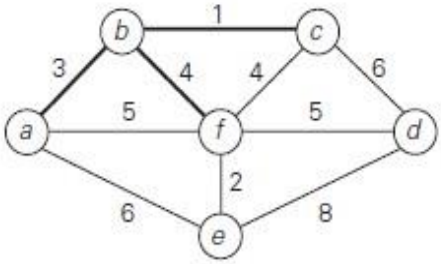
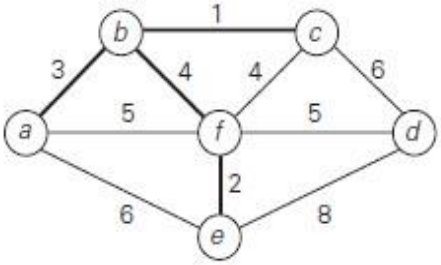
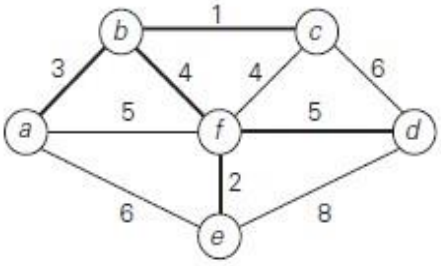
Tree vertices	Remaining vertices	Illustration
a(−, −)	**b(a, 3)** c(−, ∞) d(−, ∞) e(a, 6) f(a, 5)	
b(a, 3)	**c(b, 1)** d(−, ∞) e(a, 6) f(b, 4)	
c(b, 1)	d(c, 6) e(a, 6) **f(b, 4)**	
f(b, 4)	d(f, 5) **e(f, 2)**	
e(f, 2)	**d(f, 5)**	
d(f, 5)		

FIGURE 3.14 Application of Prim's algorithm. The parenthesized labels of a vertex in the middle column indicate the nearest tree vertex and edge weight; selected vertices and edges are in bold.

KRUSKAL'S ALGORITHM

Kruskal's algorithm looks at a minimum spanning tree of a weighted connected graph G= {V, E} as an acyclic subgraph with |V| − 1 edges for which the sum of the edge weights is the smallest.

The algorithm constructs a minimum spanning tree as an expanding sequence of subgraphs that are always acyclic but are not necessarily connected on the intermediate stages of the algorithm.

The algorithm begins by sorting the graph's edges in no decreasing order of their weights. Then, starting with the empty subgraph, it scans this sorted list, adding the next edge on the list to the current subgraph if such an inclusion does not create a cycle and simply skipping the edge otherwise.

Kruskal's algorithm looks at a minimum spanning tree of a weighted connected graph G = (V, E) as an acyclic subgraph with |V| − 1 edges for which the sum of the edge weights is the smallest.

ALGORITHM *Kruskal(G)*

//Kruskal's algorithm for constructing a minimum spanning tree

//Input: A weighted connected graph $G = (V, E)$

//Output: E_T, the set of edges composing a minimum spanning tree of G

sort E in nondecreasing order of the edge weights $w(e_{i1}) \leq . . . \leq w(e_{i|E|})$ $E_T \leftarrow \Phi$; $ecounter \leftarrow 0$ //initialize the set of tree edges and itssize

$K \leftarrow 0$ //initialize the number of processededges

while $ecounter < |V| - 1$ **do**

$k \leftarrow k + 1$

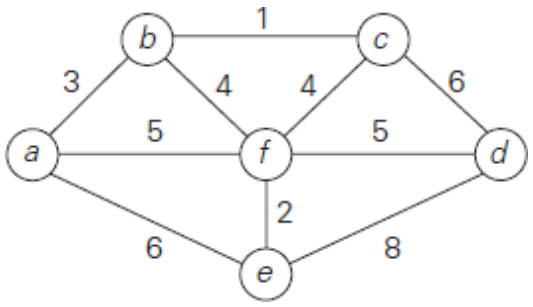
if $E_T \cup \{e_{ik}\}$ is acyclic

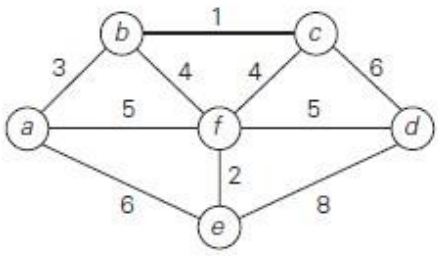
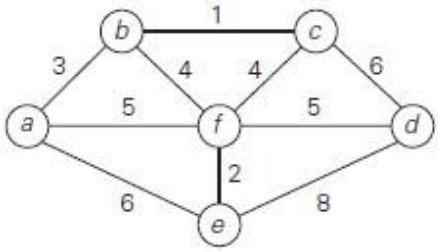
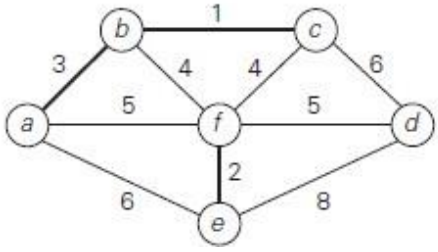
$E_T \leftarrow E_T \cup \{e_{ik}\}$; $ecounter \leftarrow ecounter + 1$

return E_T

The initial forest consists of |V | trivial trees, each comprising a single vertex of the graph. The final forest consists of a single tree, which is a minimum spanning tree of the graph. On each iteration, the algorithm takes the next edge (u, v) from the sorted list of the graph's edges, finds the trees containing the vertices u and v, and, if these trees are not the same, unites them in a larger tree by adding the edge (u, v).

Fortunately, there are efficient algorithms for doing so, including the crucial check for whether two vertices belong to the same tree. They are called union-find algorithms. With an efficient union-find algorithm, the running time of Kruskal's algorithm will be $O(|E| \log |E|)$.



Tree edges	Sorted list of edges	Illustration
	bc ef ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bc 1	bc **ef** ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
ef 2	bc ef **ab** bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	

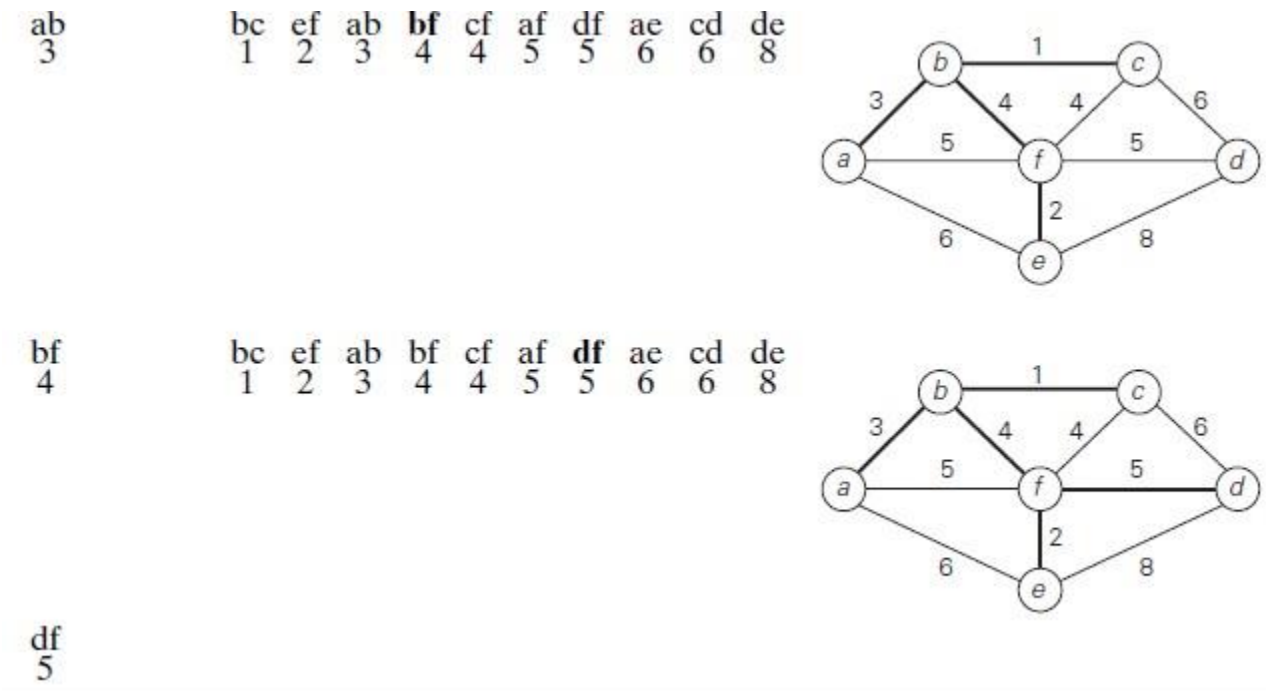


FIGURE 3.15 Application of Kruskal's algorithm. Selected edges are shown in bold.

DIJKSTRA'S ALGORITHM

- Dijkstra's Algorithm solves the **single-source shortest-pathsproblem**.
- For a given vertex called the ***source*** in a weighted connected graph, find shortest paths to all its othervertices.
- The single-source shortest-paths problem asks for a family of paths, each leading from the source to a different vertex in the graph, though some paths may, of course, have **edges in common**.
- The most widely used **applications** are transportation planning and packet routing in communication networks including the Internet.
- It also includes **finding shortest paths** in social networks, speech recognition, document formatting, robotics, compilers, and airline crew scheduling.
- In the world of **entertainment**, one can mention pathfinding in video games and

finding best solutions to puzzles using their state-spacegraphs.

- Dijkstra's algorithm is the best-known algorithm for the single-source shortest-paths problem.

ALGORITHM *Dijkstra(G,s)*

//Dijkstra's algorithm for single-source shortest paths

//Input: A weighted connected graph $G = (V, E)$ with nonnegative weights and its vertex s

//Output: The length dv of a shortest path from s to v and its penultimate vertex pv for every

// vertex v in V

Initialize(Q) //initialize priority queue to empty

for every vertex v in V

$dv \leftarrow \infty$; $pv \leftarrow$ **null**

Insert (Q, v, dv) //initialize vertex priority in the priority queue

$Ds \leftarrow 0$; *Decrease(Q, s, d_s)* //update priority

of s with d_s $V_T \leftarrow \Phi$

for $i \leftarrow 0$ **to** $|V| - 1$ **do**

$u^* \leftarrow$ *DeleteMin(Q)* //delete the minimum priority element

$V_T \leftarrow V_T \cup \{u^*\}$

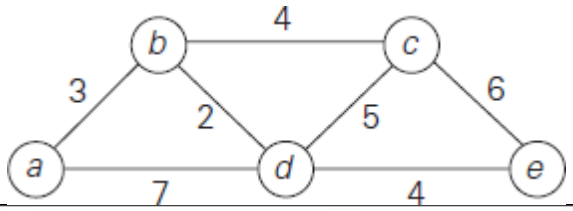
for every vertex u in $V - VT$ that is adjacent to u^* **do if** $d_u{}^* + w(u^*, u) < d_u$

$d_u \leftarrow d_{u^*} + w(u^*, u)$;

$p_u \leftarrow u^*$ *Decrease(Q, u,*

d_u*)*

The time efficiency of Dijkstra's algorithm depends on the data structures used for implementing the priority queue and for representing an input graph itself. It is in $\Theta(|V|^2)$ for graphs represented by their weight matrix and the priority queue implemented as an unordered array. For graphs represented by their adjacency lists and the priority queue implemented as a min- heap, it is in O(|E| log |V |).



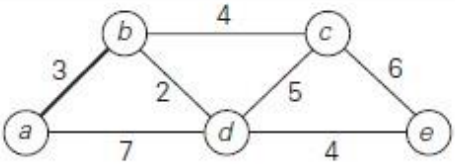
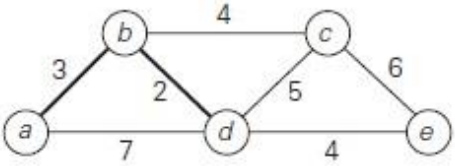
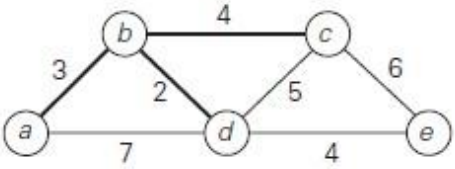
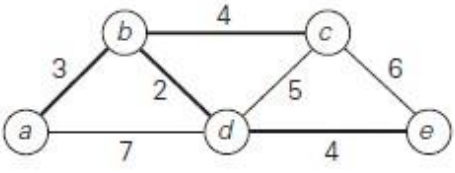
Tree vertices	Remaining vertices	Illustration
a(−, 0)	**b(a, 3)** c(−, ∞) d(a, 7) e(−, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(−, ∞)	
d(b, 5)	**c(b, 7)** e(d, 5 + 4)	
c(b, 7)	**e(d, 9)**	
e(d, 9)		

FIGURE 3.16 Application of Dijkstra's algorithm. The next closest vertex is shown in bold

The shortest paths (identified by following nonnumeric labels backward from a destination vertex in the left column to the source) and their lengths (given by numeric labels of the tree vertices) are as follows:

From a to b: a − b of length 3

From a to d: a − b − d of length 5

From a to c: a − b − c of length 7

From a to e: a − b − d − e of length 9