9.3 Dijkstra's Algorithm

In this section, we consider the *single-source shortest-paths problem*: for a given vertex called the *source* in a weighted connected graph, find shortest paths to all its other vertices. It is important to stress that we are not interested here in a single shortest path that starts at the source and visits all the other vertices. This would have been a much more difficult problem (actually, a version of the traveling salesman problem introduced in Section 3.4 and discussed again later in the book). 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.

There are several well-known algorithms for finding shortest paths, including Floyd's algorithm for the more general all-pairs shortest-paths problem discussed in Chapter 8. Here, we consider the best-known algorithm for the single-source shortest-paths problem, called *Dijkstra's algorithm*.[4] This algorithm is applicable to undirected and directed graphs with nonnegative weights only. Since in most applications this condition is satisfied, the limitation has not impaired the popularity of Dijkstra's algorithm.

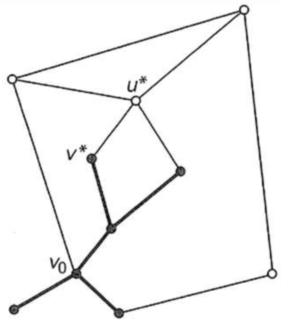
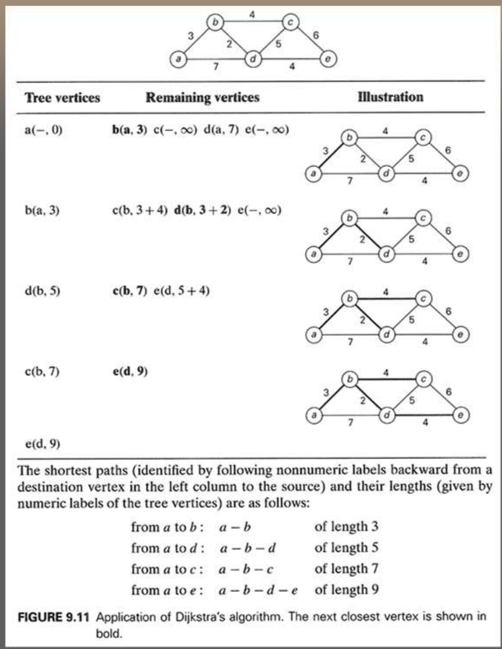


FIGURE 9.10 Idea of Dijkstra's algorithm. The subtree of the shortest paths already found is shown in bold. The next nearest to the source v_0 vertex, u^*, is selected by comparing the lengths of the subtree's paths increased by the distances to vertices adjacent to the subtree's vertices.

After we have identified a vertex u^* to be added to the tree, We need to perform two operations:

- **Move u^* from the fringe to the set of tree vertices.**
- **For each remaining fringe vertex u that is connected to u^* by an edge of weight $w(u^*, u)$ such that $d_{u^*} + w(u^*, u) < d_u$, update the labels of u by u^* and $d_{u^*} + w(u^*, u)$, respectively.**



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)		

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

FIGURE 9.11 Application of Dijkstra's algorithm. The next closest vertex is shown in bold.

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 d_v of a shortest path from s to v
// and its penultimate vertex p_v for every vertex v in V
Initialize(*Q*) //initialize priority queue to empty
for every vertex v in V
$d_v \leftarrow \infty$; $p_v \leftarrow$ null
Insert(Q, v, d_v) //initialize vertex priority in the priority queue

$d_s \leftarrow 0$; $Decrease(Q, s, d_s)$ //update priority of s with d_s

$V_T \leftarrow \emptyset$

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 $V - V_T$ 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. For the reasons explained in the analysis of Prim's algorithm in Section 9.1, 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 $O(|E| \log |V|)$. A still better upper bound can be achieved for both Prim's and Dijkstra's algorithms if the priority queue is implemented using a sophisticated data structure called the *Fibonacci heap* (e.g., [Cor09]). However, its complexity and a considerable overhead make such an improvement primarily of theoretical value.