

Text Book: Introduction to the Design and Analysis of Algorithms

Author: Anany Levitin 2 nd Edition

Unit-4

8. Dijkstra's Algorithm

Dijkstra's algorithm finds the shortest paths to a graph's vertices in order of their distance from a given source.

Single Source Shortest Paths Problem: Given a weighted connected (directed) graph G , find shortest paths from source vertex s to each of the other vertices

Dijkstra's algorithm: Similar to Prim's MST algorithm, with a different way of computing numerical labels: Among vertices not already in the tree, it finds vertex u with the smallest sum,

$$d_v + w(v, u)$$

where

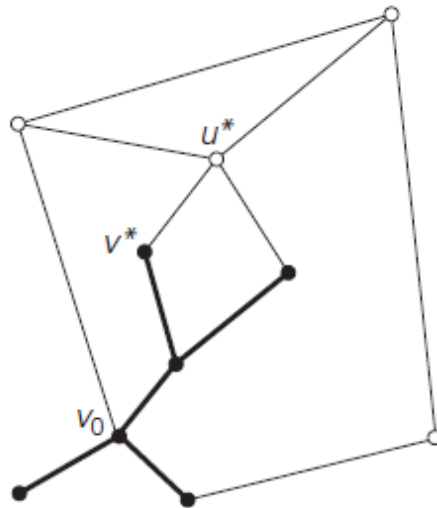
v is a vertex for which shortest path has been already found on preceding iterations (such vertices form a tree rooted at s)

d_v is the length of the shortest path from source s to v

$w(v, u)$ is the length (weight) of edge from v to u

First, it finds the shortest path from the source to a vertex nearest to it, then to a second nearest, and so on. In general, before its i^{th} iteration commences, the algorithm has already identified the shortest paths to $i - 1$ other vertices nearest

to the source. These vertices, the source, and the edges of the shortest paths leading to them from the source form a subtree T_i of the given graph.



- 0 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.

Since all the edge weights are nonnegative, the next vertex nearest to the source can be found among the vertices adjacent to the vertices of T_i . The set of vertices adjacent to the vertices in T_i can be referred to as “fringe vertices”; they are the candidates from which Dijkstra's algorithm selects the next vertex nearest to the source. To identify the i th nearest vertex, the algorithm computes, for every fringe vertex u , the sum of the distance to the nearest tree vertex v (given by the weight of the edge (v, u)) and the length dv of the shortest path from the source to v (previously determined by the algorithm) and then selects the vertex with the smallest such sum. The fact that it suffices to compare the lengths of such special paths is the central insight of Dijkstra's algorithm.

To facilitate the algorithm's operations, we label each vertex with two labels. The numeric label d indicates the length of the shortest path from the source to this vertex found by the algorithm so far; when a vertex is added to the tree, d indicates the length of the shortest path from the source to that vertex. The other label indicates the name of the next-to-last vertex on such a path, i.e., the parent of the vertex in the tree being constructed. (It can be left unspecified for the source s and vertices that are adjacent to none of the current tree vertices.) With such labeling, finding the next nearest vertex u^* becomes a simple task of finding a fringe vertex with the smallest d value. Ties can be broken arbitrarily. 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.

ALGORITHM *Dijkstra(G, s)*

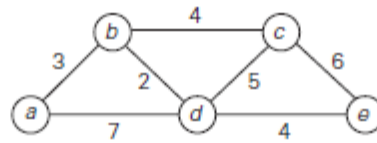
```

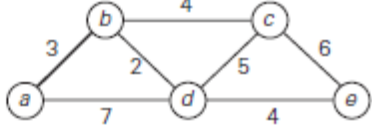
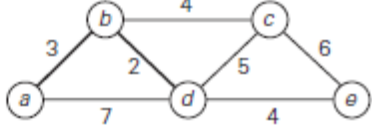
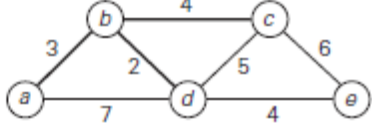
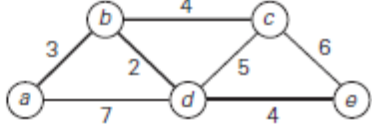
//Dijkstra's algorithm for single-source shortest paths
//Input: A weighted connected graph  $G = \langle V, E \rangle$  with nonnegative weights
//      and its vertex  $s$ 
//Output: The length  $d_v$  of a shortest path from  $s$  to  $v$ 
//      and its penultimate vertex  $p_v$  for every vertex  $v$  in  $V$ 
Initialize( $Q$ ) //initialize priority queue to empty
for every vertex  $v$  in  $V$ 
     $d_v \leftarrow \infty$ ;  $p_v \leftarrow \text{null}$ 
    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
     $u^* \leftarrow \text{DeleteMin}(Q)$  //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( $Q, u, d_u$ )
  
```

Time Efficiency

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 $\Theta(|V|^2)$ 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 $O(|E| \log |V|)$.

Example



Tree vertices	Remaining vertices	Illustration
$a(-, 0)$	$\mathbf{b(a, 3)}$ $c(-, \infty)$ $d(a, 7)$ $e(-, \infty)$	
$\mathbf{b(a, 3)}$	$\mathbf{c(b, 3 + 4)}$ $\mathbf{d(b, 3 + 2)}$ $e(-, \infty)$	
$\mathbf{d(b, 5)}$	$\mathbf{c(b, 7)}$ $\mathbf{e(d, 5 + 4)}$	
$\mathbf{c(b, 7)}$	$\mathbf{e(d, 9)}$	
$\mathbf{e(d, 9)}$		

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