

Dijkstra's Shortest Path Algorithm

My presentation of Dijkstra's shortest path algorithm differs from that of the textbook (Tucker). Thus, I detail the algorithm below. The algorithm can be thought of as propagating a "signal" from the root (source) vertex, and keeping track at every other vertex of when the signal arrives for the first time, given that it takes a certain length of time for the signal to travel along each edge.

Input

The input to the algorithm is a graph $G = (V, E)$. We can assume, without loss of generality, that the vertices are labeled with integers — $V = \{1, 2, \dots, n\}$ — and that our goal is to construct a tree of shortest paths rooted at vertex 1. Edges are labeled with "lengths" ("weights", "costs") that are *nonnegative*; edge (i, j) has length $c_{ij} \geq 0$. (If there is *no* edge (i, j) , then $c_{ij} = +\infty$.)

Output

The algorithm will construct a tree of shortest paths rooted at the vertex 1. In particular, each vertex i will have two pieces of data associated with it: (1) a label u_i that will be equal to the length of a shortest path from vertex 1 to vertex i , and (2) a pointer \mathbf{pred}_i that gives the vertex that is the predecessor to i in a shortest path from 1 to i . (In other words, \mathbf{pred}_i is the parent of i in the shortest path tree constructed. By following the \mathbf{pred} pointers, one can trace out any shortest path from 1 to i .)

Algorithm

The algorithm keeps track of a partitioning of the vertex set V into two types of vertices: those that are "permanently labeled" (the set $P \subseteq V$), and those that are "temporarily labeled" (the set $T \subseteq V$). To be "permanently labeled" means that the value of u_i is not going to change ever again, since it is in fact equal to the length of a shortest path from 1 to i . To be "temporarily labeled" means that the value of u_i *may* change again (it may go down), so we do not yet know if it equals the length of a shortest path from 1 to i . At the beginning of the algorithm, only the root (vertex 1) is permanently labeled. At the end of the algorithm, all vertices are permanently labeled.

(0) Initialization.

$$\begin{aligned} u_1 &= 0 \\ u_j &= c_{1j}, \quad j = 2, 3, \dots, n \\ \mathbf{pred}_j &= 1, \quad j = 2, 3, \dots, n; \quad \mathbf{pred}_1 = \text{NIL} \\ P &= \{1\}, \quad T = \{2, 3, \dots, n\} \end{aligned}$$

- (1) **Assign a Permanent Label.** Find $k \in T$ with the smallest label: $u_k = \min_{j \in T} u_j$. (Ties may be broken arbitrarily, or by some tie-breaking rule, such as picking the lowest numbered vertex whenever there is a choice.)

$$T = T - \{k\}, \quad P = P \cup \{k\}$$

If $T = \emptyset$, STOP.

- (2) **Update Temporary Labels.** For each $j \in T$, do the following: If $u_k + c_{kj} < u_j$, then update u_j :

$$u_j \leftarrow u_k + c_{kj}, \quad \mathbf{pred}_j \leftarrow k$$

(Here, k is the vertex that was just assigned a permanent label in step (1).)

(Otherwise, do not change u_j or \mathbf{pred}_j .)

Go to Step (1).

Theorem 1 For a graph with n vertices and e edges, having nonnegative edge lengths, Dijkstra's algorithm can be implemented to run in time $O(e + n \log n)$ (i.e., the running time is at most a constant times $e + n \log n$).

Proof. The proof of this claim requires some more advanced knowledge of data structures. It is not hard, however, to implement the algorithm to run in time $O(n^2)$ or in time $O(e \log n)$. \square