

Prim's Algorithm

Here the set A always forms a single tree.

We add the minimum weight edge connecting the tree A to a vertex not in the tree. This edge is safe for A .

Let G be a connected graph and let r be some vertex in G . The algorithm builds a MST rooted at r .

During the algorithm, all vertices that are **not** in the tree are kept in a min-heap data structure Q based on a *key* defined as:

$key[v]$ is the minimum weight of any edge connecting v to a vertex in the tree. If there is no such edge then $key[v] = \infty$.

$\pi[v]$ names the parent of v in the tree.

The tree is kept implicitly as

$$A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}.$$

When the algorithm terminates Q is empty and the MST is given by

$$A = \{(v, \pi[v]) : v \in V - \{r\}\}.$$

MST – Prim(G, r)

```
1 for each  $u \in V[G]$ 
2    $key[u] = \infty$ 
3    $\pi[u] = NIL$ 
4  $key[r] = 0$ 
5  $Q = V(G)$ 
6 while  $Q$  is not empty
7    $u = EXTRACT - MIN(Q)$ 
8   for each  $v \in Adj[u]$ 
9     if  $v \in Q$  and  $w(u, v) < key[v]$ 
10       $DECREASE - KEY(Q, key[v], w(u, v))$ 
11       $\pi[v] = u$ 
12 return  $\{(v, \pi[v]) : v \in V - \{r\}\}$ 
```

Can be implemented in $O(|E| + |V| \log |V|)$ time.

Shortest Paths Problems

Input: a directed graph $G = (V, E)$ and a **weight** function $w : E \rightarrow R$.

The weight of a path $p = v_0, v_1, v_2, \dots, v_k$ is

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i).$$

The weight of the **shortest path** from u to v , $\delta(u, v)$ is the minimum of $w(p)$ for all p connecting u to v , and ∞ if there is no such path in G .

Variants:

Single Source Shortest Paths - Compute shortest paths from a given source to all vertices in the graph.

Single Destination Shortest Paths: Compute shortest paths to a given destination from all vertices in the graph.

Single Pair Shortest Path - Compute shortest path for a given pair of vertices.

All Pairs Shortest Paths - Compute the shortest paths for all pairs of vertices in the graph.

Negative weights are allowed, but the problem is not well defined when a **negative cycle** is present.

Single Source Shortest Paths

Compute the shortest path from s to all vertices.

Lemma 1. *If $p = v_0, v_1, \dots, v_j, \dots, v_k$ is a shortest path from v_0 to v_k , then $p' = v_0, v_1, \dots, v_j$ is a shortest path from v_0 to v_j .*

Proof. Assume that P'' is a shorter path from v_0 to v_j , then P'' followed by v_{j+1}, \dots, v_k would be a shorter path from v_0 to v_k . \square

The Dijkstra Algorithm

The algorithm constructs a **tree of shortest paths**.

The root of the tree is s .

A path on the tree corresponds to shortest paths to s for all vertices on that path.

The shortest paths tree is constructed in $|V|$ iterations.

Starting from $S = \emptyset$ and extending S by one vertex in each iteration, the algorithm computes a shortest paths tree restricted to internal vertices only from S .

When $S = V$ we get the shortest paths tree for the graph.

For all $v \in V$, and in each iteration,

$d[v]$ - the distance from s to v by a path that uses only vertices of S .

$\pi[v]$ - the predecessor of v on the tree.

$Extract_Min(Q)$ - the vertex with smallest $d[v]$ among the vertices in Q .

The Dijkstra Algorithm

Dijkstra (G, w, s)

1. For all $v \in V$ do
 - (a) $d[v] \leftarrow \infty$;
 - (b) $\pi[v] \leftarrow NIL$;
2. $d[s] = 0$;
3. $S \leftarrow \emptyset$;
4. $Q \leftarrow V$;
5. While $Q \neq \emptyset$ do
 - (a) $u \leftarrow Extract_Min(Q)$;
 - (b) $S \leftarrow S \cup \{u\}$;
 - (c) For all $v \in Adj[u]$ do
 - i. If $d[v] > d[u] + w(u, v)$ then
 - A. $d[v] \leftarrow d[u] + w(u, v)$;
 - B. $\pi[v] \leftarrow u$;

Correctness

Theorem 1. *The algorithm computes a correct shortest distance tree when applied to a graph G with no negative weight edges.*

Proof.

We'll show by induction on the size of S that for every $1 \leq k \leq n$, at the end of the while loop with $|S| = k$, the functions $\pi[]$ and $d[]$ satisfy:

1. If $v \in S$ then $d[v]$ is the shortest path distance of v from s , and $\pi[v]$ encodes the last edge of that path.
2. If $v \notin S$ then $d[v]$ is the shortest path of s from v using only internal vertices of S , and $\pi[v]$ encodes the last edge of that path.

The induction hypothesis holds for $|S| = 1$ since in that case $S = \{s\}$, and for all v either $d[v]$ is the weight of the edge (s, v) or ∞ .

Assume that the induction hypothesis holds for $|S| = j - 1$. Consider the j -th iteration of the while loop:

Lemma 2. *The shortest path from s to u uses only vertices of S .*

Proof. Assume that a shorter path from s to u contains a vertex in Q . Let v be the first such vertex, then $d[v] < d[u]$. \square

Thus, 1 of the induction hypothesis is satisfied.

Lemma 3. *If vertex $v \in Q$ had a correct value $d[v]$ at the beginning of the while iteration, it has a correct value at the end of the iteration.*

Proof.

By the lemma's assumption we need only to check paths from s to v that contain u .

The algorithm checks only paths in which v is adjacent to u .

How about the remaining paths?

If there is a shorter path

$$P = s, v_1, v_2, \dots, u, v_k, \dots, v$$

with u not adjacent to v , then since v_k joined S before u , there must be a path from s to v_k that does not use u and is not longer. Thus, $d[v]$ has the correct value without considering paths that use u . \square

2 of the induction hypothesis is satisfied. \square

Run Time

Theorem 2. *The algorithm can be implemented in $O(|E| + |V| \lg |V|)$ steps.*