Prim's Algorithm

We now present a second MST algorithm: **Prim's algorithm**. Like Kruskal's algorithm, Prim's algorithm depends on a method of determining which greedy choices are safe. The method is to continually enlarge a single connected component by adjoining edges emanating from isolated vertices.[1]

Algorithm: PRIM-MST(V, E, w)

Choose an arbitrary start vertex s
$C \leftarrow \{s\}$
$T \leftarrow ;$
while C is not the only connected component of T **do**
Select a light edge (u, v) connecting C to an isolated vertex v
$T \leftarrow T \cup \{(u, v)\}$
$C \leftarrow C \cup \{v\}$
return T

Proof of correctness for Prim's algorithm. Again, we use a loop invariant:

Prior to each iteration, T is a subset of an MST.

- *Initialization.* T has no edges, so trivially it is a subset of an MST.
- *Maintenance.* Suppose $T \subseteq T_*$ where T_* is an MST, and suppose the edge (u, v) gets added to T, where $u \in C$ and v is an isolated vertex. Since (u, v) is a light edge for the cut $(C, V \setminus C)$ which is respected by T, it follows by Proposition 4.1 that (u, v) is a safe edge for T. Thus $T \cup \{(u, v)\}$ is a subset of an MST.
- *Termination.* At termination, C is the only connected component of T, so by Proposition 3.1(v), T has at least $|V| - 1$ edges. Since T is also a subset of an MST, it follows that T has exactly $|V| - 1$ edges and is an MST. □

The tricky part of Kruskal's algorithm was keeping track of the connected components of T. In Prim's algorithm this is easy: except for the special component C, all components are isolated vertices. The tricky part of Prim's algorithm is efficiently keeping track of which edge is lightest among those which join C to a new isolated vertex. This task is typically accomplished with a data structure called a min-priority queue.

4.2.1 Min-Priority Queue

A **min-priority queue** is a data structure representing a collection of elements which supports the following operations:

INSERT(Q, x) – Inserts element x into the set of elements Q

MINIMUM(Q) – Returns the element of Q with the smallest key

[1] An **isolated** vertex is a vertex which is not connected to any other vertices. Thus, an isolated vertex is the only vertex in its connected component.

EXTRACT-MIN(Q) – Removes and returns the element with the smallest key DECREASE-KEY(Q, x,k) – Decreases the value of x's key to new value k.

With this data structure, Prim's algorithm could be implemented as follows:

Algorithm: PRIM-MST(G,s)

1 $T \leftarrow ;$

2 **for each** $u \in G.V$ **do**

3 $u.key \leftarrow \infty$ B initialize all edges to "very heavy"

4 B The component C will be a tree rooted at s. Once a vertex u gets added to C, $u.\pi$ will be a pointer to its parent in the tree.

5 $u.^{\pi} \leftarrow$ NIL

6 $s.key \leftarrow 0$ B this ensures that s will be the first vertex we pick

7 Let $Q \leftarrow G.V$ be a min-priority queue

8 **while** Q 6=; **do**

9 $u \leftarrow$ EXTRACT-MIN(Q)

10 **if** $u.\pi$6= NIL **then**

11 $T \leftarrow T \cup \{(u, u.\pi)\}$

12 B We assume that G is presented in the adjacency-list format*

13 **for each** $v \in G.adj[u]$ **do**

14 **if** $v \in Q$ and $w(u,v) < v.key$ **then**

15 $v.^{\pi} \leftarrow u$

16 $v.key \leftarrow w(u,v)$ B using DECREASE-KEY

17 **return** T

---

*For more information about ways to represent graphs on computers, see §22.1 of CLRS.

4.2.2 Running Time of Prim's Algorithm

Lines 1 through 6 clearly take $O(V)$ time. Line 7 takes $T_{\text{BUILD-QUEUE}}(V)$, where $T_{\text{BUILD-QUEUE}}(n)$ is the amount of time required to build a min-priority queue from an array of n elements. Within the "while" loop of line 8, EXTRACT-MIN gets called $|V|$ times, and the instructions in lines 14–16 are run a total of $O(E)$ times. Thus the running time of Prim's algorithm is

$$O(V) + T_{\text{BUILD-QUEUE}}(V) + V\, T_{\text{EXTRACT-MIN}} + O(E)\, T_{\text{DECREASE-KEY}}.$$