CSE100 Algorithm Design and Analysis — Lab 16, Fall 2018
Last update: November 19, 2018 — Deadline: Sunday, Dec 9, 11:59pm

Minimum Spanning Tree

In the Minimum Spanning Tree problem, we are given as input an undirected graph $G = (V, E)$ together with weight $w(u, v)$ on each edge $(u, v) \in E$. The goal is to find a minimum spanning tree for G. Recall that we learned two algorithms, Kruskal's and Prim's in class. In this assignment, you are asked to implement Prim's algorithm. The following is a pseudo-code of Prim's algorithm.

```
1: Initialize a min-priority queue Q.
2: for all u ∈ V do
3:    u.key = ∞.
4:    u.π = NIL.
5:    Insert (Q, u).
6: end for
7: Decrease-key(Q, r, 0).
8: while Q ≠ ∅ do
9:    u = Extract-Min(Q).
10:   for all v ∈ Adj[u] do
11:      if v ∈ Q and w(u, v) < v.key then
12:         v.π = u.
13:         Decrease-Key(Q, v, w(u, v)).
14:      end if
15:   end for
16: end while
```

Input structure The input is G, w, and r, where r is an arbitrary vertex the user can specify as root. The input has the following format. There are two integers on the first line. The first integer represents the number of vertices, $|V|$. The second integer is the number of edges, $|E|$. Vertices are indexed by $0, 1, \ldots, |V| - 1$. Each of the following $|E|$ lines has three integers $u, v, w(u, v)$ representing an edge (u, v) with weight $w(u, v)$. Use vertex 0 as the root r.

Output structure The above pseudo-code stores the MST by π, where $v.\pi = u$ means that u is v's unique parent; here, $r.\pi =$ NIL since r has no parent. Output the MST by outputting the π value of a vertex in each line, in the order $1, 2, \ldots, |V| - 1$. (Do not output the root's parent.)

Implementation Issues Prim's algorithm requires a min-priority queue that supports the `DecreaseKey` operation which is not supported by the standard C++ priority queue. You are allowed to use an "inefficient" priority queue that supports each operation in $O(|V|)$ time. Such an inefficient priority queue can be easily implemented using an array. Then, the running time of your implementation is roughly $O(|E||V|)$. However, you may still use the C++ priority queue with a simple "invalidation trick" and have your code to run in $O(|E| \log |V|)$. Instead of decreasing an element's key, just mark the element as invalid and push a new (valid) element with the new key value to the queue. Then you have to be careful when extracting a minimum element because what you really want is a minimum element that is valid. Extracting a valid min element could take a few iterations. However, the priority queue has at most $O(|E|)$ elements, so each `ExtractMin` operation takes $O(\log |E|) = O(\log |V|)$ time. Since you extract minimum elements at most $O(|E|)$ times, you only need $O(|E| \log |V|)$ time for extracting valid min elements.