CSE100 Algorithm Design and Analysis — Lab 16, Fall 2020

Last update: December 1, 2020 — Deadline: Friday, Dec 11, 5:00pm

1 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.

Initialize a min-priority queue Q.
for all $u \in V$ **do**
    $u.key = \infty$.
    $u.\pi = NIL$.
    Insert (Q, u).
end for
Decrease-key$(Q, r, 0)$.
while $Q \neq \emptyset$ **do**
    $u =$ Extract-Min(Q).
    for all $v \in Adj[u]$ **do**
        if $v \in Q$ and $w(u, v) < v.key$ **then**
            $v.\pi = u$.
            Decrease-Key$(Q, v, w(u, v))$.
        end if
    end for
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 just have to be careful when extracting a minimum element because what you really want is a minimum element that is valid. So extracting a valid min element could take a few iterations. However, at any point in time, 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.