CSE 100 Algorithm Design and Analysis Lab 09 (20 points)
Due date: November 22, 23:59

Minimum Spanning Tree

In the Minimum Spanning Tree (MST) problem, we are given as input an undirected graph $G = (V, E)$ together with a 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 discussed two algorithms – Kruskal and Prim – in class. In this assignment, you are asked to implement Prim's algorithm. The following is the pseudocode of Prim's algorithm.

```
Initialize a min-priority queue Q
for all u ∈ V do
  u.key = ∞
  u.π = NIL
  Insert (Q, u)
end for
DecreaseKey(Q, r, 0)
while Q ≠ ∅ do
  u = ExtractMin(Q)
  for all v ∈ Adj[u] do
    if v ∈ Q and w(u, v) < v.key then
      v.π = u
      DecreaseKey(Q, v, w(u, v))
    end if
  end for
end while
```

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 $0, 1, \ldots, |V| - 1$. On the following $|E|$ lines, there are three integers $u, v, w(u, v)$ representing an edge (u, v) with weight $w(u, v)$. Use vertex 0 as the root r. The above pseudo-code outputs the MST by π, where $v.\pi = u$ means that u is v's unique parent. $r.\pi = NIL$ since r has no parent. MST is printed by outputting the π value of a vertex in each line, in the order $1, 2, \ldots, |V| - 1$. We exclude the root's parent.

Example of input and output

```
Input
9 14
0 1 40
0 7 85
1 2 80
1 7 110
2 3 70
2 5 45
2 8 22
3 4 90
3 5 140
```

```
4 5 100
5 6 25
6 7 10
6 8 60
7 8 75

Output
0
1
2
3
2
5
6
2
```

(The first number refers to the parent of vertex 1, and the second number to the parent of vertex 2, and so on.)

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.

Submission Submit the source code `MST.cpp` through the assignments page of CatCourses by the deadline. Be careful since CatCourses strictly enforces the assignment deadline. You may be asked to compile, run, and explain the code to the TA to prove that you understand what you wrote.

Grading We provide 10 test cases for you to try your implementation. Each of them is valued at 1 point if executed correctly. We will use 10 additional test cases to check that your implementation is general enough. These are not provided to you and are also valued at 1 point each.

Important Note We will use plagiarism software to detect cheating. Offenders will be subjected to the UCM Academic Honesty Policy which states: *if any violation of the UCM Academic Honesty Policy is suspected in a course, the instructor of record must fill out the Faculty Report for Academic Misconduct.*