Minimum Spanning Tree

Description 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 ∈ V do
  u.key = ∞.
  u.π = NIL.
  Insert (Q, u).
end for
Decrease-key(Q, r, 0).
while Q ≠ ∅ do
  u = Extract-Min(Q).
  for all v ∈ Adj[u] do
    if v ∈ Q and w(u, v) < v.key then
      v.π = u.
      Decrease-Key(Q, v, w(u, v)).
    end if
  end for
end while
```

The code takes as input G, w and r where r is an arbitrary vertex the user can specify as root.

Input The input will have the following format. The first integer refers to the number of vertices, i.e. $|V|$. The second integer is the number of edges, i.e. $|E|$. Vertices are indexed by 0, 1, ..., $|V| - 1$. Then, three numbers $u, v, w(u, v)$ appearing in each line means an edge (u, v) with weight $w(u, v)$. Use vertex 0 as the root r. In every input, all edges will have distinct weights. Hence the resulting MST must be unique, and the above pseudo-code outputs the MST by π. Here $v.\pi = u$ means that u became v's unique parent in the process of Prim's algorithm; $r.\pi = NIL$ since r has no parent.

Output You are asked to output the MST by outputting the π value of a vertex in each line, in the order of $1, 2, ..., |V| - 1$. Here we excluded the root's parent. See the given example below.

Implementation Issues Prim's algorithm needs min-priority queue that supports Decrease-key operation, which is not supported by the standard c++ priority queue. So you are allowed to use an 'inefficient' priority queue which support 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 will be 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 $O(|E| \log |V|)$ time. 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 extract-min operation will take $O(\log |E|) = O(\log |V|)$ time. Since you will extract minimum elements at most $O(|E|)$ times, you will only need $O(|E| \log |V|)$ time for extracting valid min elements.

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
```

(The input is a graph with weights given per edge.)

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.)

Testing your code Your execution file name must be 'MST.exe'. Use Grade09 to test your program.

Submission As usual, before the posted deadline, submit a .zip or zipped tar archive of your program through the assignments page of CatCourses. Please use your UCMNetID as the filename for the zipped archive.

NEW: be sure to read LabDeadlines.txt in order to read the deadline rules. Be careful that the deadlines in the CatCourses entries (which are strictly enforced) are for the latest possible day for you to upload, according to the posted rules. Recall that submission alone is not enough, you must present your work to your TA before the deadline.

Important Reminder Never change the grading scripts of the files under the "testfiles" folder. If you do so, it will be considered as SERIOUS CHEATING.