Crocodile's Underground City

https://wcipeg.com/problem/ioi1121

Problem

You are given a graph of N (3 ≤ N ≤ 100,000) vertices with M (2 ≤ M ≤ 1,000,000) weighted bidirectional edges. K (1 ≤ N ≤ K) of these vertices are designated as 'exit' vertices. From the chamber 0, find the shortest path to any 'exit' vertex, given that at any given vertex, a single edge connecting to that vertex is 'blocked' such that the shortest path to any exit vertex is maximized.

Solution

We will jury rig an implementation of Dijkstra to solve this problem.

Instead of setting the distance to vertex 0 as 0, like all other Dijkstra problems we have faced, we will instead set the distance to all the exit vertices as 0 (and push them into the priority queue). Thus, we will find the distance from any exit vertex to vertex 0 rather than the distance from vertex 0 to any exit vertex.

It is trivial to see that for any vertex, the same edge will always be blocked off, no matter how that vertex is arrived at. Thus, each vertex can have a shortest path to an exit vertex attached to it.

Suppose we are at some vertex A. If we know the shortest paths from all the adjacent vertices to an exit vertex, then the shortest path from vertex A to an exit vertex is simply the second smallest of the set of adjacent vertex paths plus the weights of their connecting edges - if there is only one adjacent vertex, then there is no possible path (because the only edge will be blocked).

Putting the observations together, we declare a new distance array `dist[N][2]`, where `dist[A][0]` is the shortest path to a given vertex A and `dist[A][1]` is the second shortest path. Only push new entries into the priority queue like in Dijkstra when `dist[N][1]` is filled.

```cpp
#include <bits/stdc++.h>
using namespace std;

#define f first
#define s second

int dist[100010][2],N,M,K;
vector<pair<int, int>> G[100010];
priority_queue<pair<int, int>, vector<pair<int, int>>,
greater<pair<int, int>>> PQ;

int main(){
    cin.sync_with_stdio(0); cin.tie(0);
    memset(dist, -1, sizeof dist);
    cin >> N >> M >> K;
    for(int i = 0; i < M; i++){
        int A,B,C;
        cin >> A >> B >> C;
        G[A].push_back({B,C});
        G[B].push_back({A,C});
    }
    for(int i = 0; i < K; i++){
        int A;
        cin >> A;
        dist[A][0] = 0;
        PQ.push({0,A});
    }
    while(!PQ.empty()){
        pair<int, int> u = PQ.top();
        PQ.pop();
        if(dist[u.s][0] == -1){
            dist[u.s][0] = u.f;
            continue;
        }
        if(dist[u.s][1] == -1){
            dist[u.s][1] = u.f;
            for(pair<int, int> i : G[u.s]){
                PQ.push({u.f + i.s, i.f});
            }
        }
    }
    cout << dist[0][1] << endl;
}
```