Quantum Shmovements Analysis

Timothy Gao

You are given a graph of N ($1 \leq N \leq 2 \cdot 10^5$) nodes and M edges ($1 \leq M \leq 2 \cdot 10^5$). Electron Ella and Ethan initially start on any two nodes (potentially the same). Every turn, either electron shmoves from its current node to an adjacent node.

However, there is a catch: Ella and Ethan are entangled! For each node, if one electron wants shmove to an adjacent node, there is a requirement that the other electron must currently be at a certain node. In other words, all edges all have a "requirement", that in order to use it the other electron must be at a particular node.

Find the maximum number of nodes that can be visited by either electron, given the starting locations of the Ella and Ethan.

1 Initial Observations

Let's start with a brute force approach. We construct a state (A, B) representing the nodes that Ella and Ethan are on. There are N^2 such states, so by traversing between them we can achieve $\mathcal{O}(N^2)$ with BFS, DFS, etc. This solution is too slow for our time limit, but what if it's not...

It turns out this kind of wishful thinking is helpful for coming up with the full solution. Instead of defining each state as locations of Ella and Ethan, let's take advantage of this "entanglement". If Ethan takes an edge with requirement C to move from node A to B, Ella must be on node $C \rightarrow$ If Ethan is on node B, Ella must be on the requirement nodes of one of its edges. Another important observation is that Ella and Ethan are interchangeable (i.e., we can swap the nodes they're on whenever we want), because the requirement of each edge is not electron-dependent.

2 Arriving at the Solution

Combining these two observations, the state (node A is on, previous edge taken by A), where A is the most recently moved electron, is sufficient to uniquely determine the locations of both electron. Notice we don't care whether A is Ella or Ethan. At the start, we can set A to either Ethan or Ella's location and previously moved edge to some special value (like -1) to indicate the starting location of the other electron.

How many such states are there? Well the degree of each node is the number of possible edges taken to get to it. Summing up the degrees of all nodes, each edge is counted twice, so the number of states is $2 \cdot M$, the number of edges.

It turns out we can simplify this further by noticing that we don't actually care about the previous edge taken, but only the location of the other electron. However, the same argument for the number of states still applies, because each "previous edge taken" corresponds to a unique location for the other electron. This allows us to simply use the state (A, B), the locations of each electron!

To neatly handle transitions between states, we create an "adjacency map", map⟨int, vector⟨int⟩⟩ adj[maxn], where each node stores a map with keys being requirement nodes and values being a list of all edges (rather the destination nodes of these edges) that have the aforementioned requirement node. With this, we can perform transitions in $\mathcal{O}(log(M))$ or $\mathcal{O}(1)$. We can store each state as a pair into a set. Recall there are $2 \cdot M$ states. Depending on the implementation of set and map (I chose the non-hashed versions), the final time complexity is either $\mathcal{O}(N + M)$ or $\mathcal{O}(N + Mlog(M))$. Both pass comfortably under the time limit.

3 Code

```cpp
//@timothyg

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;
typedef pair<int, int> pii;

#define pb push_back

#define f first
#define s second

const int maxn = 2e5 + 10;
map<int, vector<int>> adj[maxn];
set<pii> state_vis;
int node_vis[maxn];

void dfs(int A, int B){
    if(state_vis.count(pii(A, B))){
        return;
    }
    state_vis.insert(pii(A, B));
    node_vis[A] = 1; node_vis[B] = 1;
    for(int n : adj[A][B]){
        //Where A can move to given that B stays
        dfs(n, B);
    }
    for(int n : adj[B][A]){
        //Where B can move to given that A stays
        dfs(B, n);
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(0);
    int N, M, A, B; cin >> N >> M >> A >> B;
    for(int i = 0; i<M; i++){
        int u, v, other; cin >> u >> v >> other;
        adj[u][other].pb(v);
        adj[v][other].pb(u);
    }
    dfs(A, B);
    int tot = 0;
    for(int i = 1; i<=N; i++){
        tot += node_vis[i];
    }
    cout << tot << '\n';
    return 0;
}
```