MCS – 253P ADVANCED PROGRAMMING AND PROBLEM SOLVING

LAB 8 Code(Path with Maximum Probability)

Aswin Sampath
saswin@uci.edu
53844684

Question)

Problem List | Run | Submit

Description | Editorial | Solutions | Submissions

1514. Path with Maximum Probability

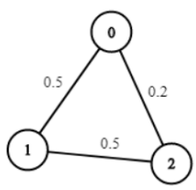
Medium | Topics | Companies | Hint

You are given an undirected weighted graph of `n` nodes (0-indexed), represented by an edge list where `edges[i] = [a, b]` is an undirected edge connecting the nodes `a` and `b` wit probability of success of traversing that edge `succProb[i]`.

Given two nodes `start` and `end`, find the path with the maximum probability of success to go from `start` to `end` and return its success probability.

If there is no path from `start` to `end`, **return 0**. Your answer will be accepted if it differs from the correct answer by at most **1e-5**.

Example 1:




```
Input: n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.2], start = 0, end = 2
Output: 0.25000
Explanation: There are two paths from start to end, one having a probability of success = 0.2 and the other has 0.5 * 0.5 = 0.25.
```

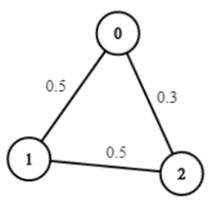
Example 2:

3K | 39

Problem List | Run | Submit

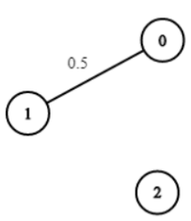
Description | Editorial | Solutions | Submissions

Example 2:




```
Input: n = 3, edges = [[0,1],[1,2],[0,2]], succProb = [0.5,0.5,0.3], start = 0, end = 2
Output: 0.30000
```

Example 3:




```
Input: n = 3, edges = [[0,1]], succProb = [0.5], start = 0, end = 2
Output: 0.00000
Explanation: There is no path between 0 and 2.
```

3K | 39

Code :

```cpp
class Solution {
public:
    double maxProbability(int n, vector<vector<int>>& edges, vector<double>& succProb, int start_node, int end_node) {
        vector<vector<pair<int,double>>>graph(n);
        int i=0;
        for(auto e:edges){
            graph[e[0]].push_back({e[1],succProb[i]});
            graph[e[1]].push_back({e[0],succProb[i]});
            i++;
        }

        priority_queue<pair<double,int>>q;
        vector<bool>visited(n,false);
        q.push({1.0,start_node});
        vector<double>dist(n,0);
        dist[start_node]=1.0;
        while(!q.empty()){
            auto top = q.top();
            q.pop();
            int node = top.second;
            double proba = top.first;
            if(visited[node])continue;
            visited[node]=true;

            for(auto conn:graph[top.second]){
                    if(dist[conn.first] < conn.second*proba){
                        dist[conn.first] = conn.second*proba;
                        q.push({dist[conn.first],conn.first});
                    }
                }
        }
        return dist[end_node];
    }
};
```

Output: