Dijk or Dyck?

locked

by volmahesh

Problem Submissions Leaderboard Discussions

Given a undirected simple graph G with positive integer weights, and a vertex $s \in G$, count the number of shortest paths from s to each vertex $v \in G$.

Standard notations for Graphs are followed.

There are N vertices and M edges in G.
Vertices are labelled 0 to N-1.
You will be given the number of edges in G and the edge list.

Try to implement this as efficiently as possible.

Input Format

```
N  M  s
u1 v1 w1
u2 v2 w2
.  .  .
.  .  .
.  .  .
uM vM wM
```

Here, an edge (u_i, v_i) has weight w_i.
s is the source vertex.

Constraints

$1 \leq N \leq 50,000$

$w_i > 0$ (Most w_i are small)

$\forall v \in G, \quad d_{SP}(s, v) < 2^{32}$

$\forall v \in G, \quad \sigma_{SP}(s, v) < 2^{32}$

Here,
$d_{SP}(u, v)$ denotes distance of the shortest path from u to v.
$\sigma_{SP}(u, v)$ denotes the number of shortest paths from u to v.

Output Format

```
c0 c1 c2 ... cN-1
```

Here, c_i is the number of shortest paths from s to v_i.

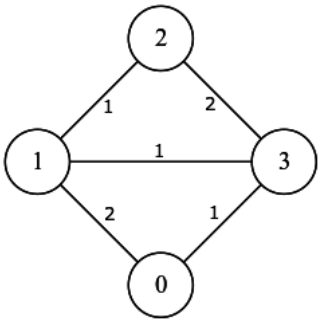
Sample Input

```
4 5 0
0 1 2
0 3 1
1 2 1
1 3 1
3 2 2
```

Sample Output

```
1 2 3 1
```

Explanation



From v_0,

v_1 : There are 2 shortest paths. (0--1) & (0--3--1).

v_2 : There are 3 shortest paths. (0--1--2), (0--3--2) & (0--3--1--2).

v_3 : There is a unique shortest path. (0--3).

Submissions: 61
Max Score: 50
Difficulty: Hard

Rate This Challenge:

More

Current Buffer (saved locally, editable) C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <bits/stdc++.h>

using namespace std;


int main() {
    int N,M,s,u1,v1,w,uQc;
    cin >> N;
    cin >> M;
    cin >> s;

    vector<int> distance(N,INT_MAX);
    vector<int> paths(N,0);
    priority_queue< pair<int,int> , vector<pair<int, int>> , greater<pair<int, int>> > prQ;
    int check[N]={};
    list< pair<int, int> > *adjList;
    adjList = new list< pair<int, int> > [N];
    for(int i=0;i<M;i++){
        cin >> u1;
        cin >> v1;
        cin >> w;
        adjList[u1].push_back(make_pair(v1,w));
        adjList[v1].push_back(make_pair(u1,w));
    }

    prQ.push(make_pair(0,s));
    distance[s] = 0;
    paths[s] = 1;
    check[s] = 1;
    while(prQ.empty()==false){
```

```
        uQc = prQ.top().second;
        int wt = prQ.top().first;
        prQ.pop();
        if(wt==distance[uQc]){
            check[uQc] = 1;
        }
        else{
            if(check[uQc]==1){
                continue;
            }
        }


        list< pair<int, int> >::iterator itr;
        for(itr = adjList[uQc].begin();itr!=adjList[uQc].end();itr++){
            int v1 = (*itr).first;
            int w = (*itr).second;

            if(distance[v1] > distance[uQc]+w){
                distance[v1] = distance[uQc]+w;
                paths[v1] = paths[uQc];
                prQ.push(make_pair(distance[v1],v1));
            }
           else if(distance[v1] == (distance[uQc]+w)){
                paths[v1] = paths[uQc] + paths[v1];
            }

        }


    }
    vector<int>::iterator i;
    for(i=paths.begin();i!=paths.end();i++){
        cout<<*i<<" ";
    }


    return 0;
}

```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code