BC Redux

locked

by volmahesh

> Given a undirected simple graph G with positive integer weighted edges, calculate the Betweenness Centrality (henceforth abbreviated **BC**) metric for each node in G *by using Brandes' Algorithm*.

This assignment's goal is to cultivate the habit of reading/understanding research papers and applying them.

Here are some links to *the* paper (**mirrors**)--
http://www.inf.uni-konstanz.de/algo/publications/b-fabc-01.pdf
http://www3.cs.stonybrook.edu/~jgao/CSE642/betweenness.pdf
citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.11.2024&rep=rep1&type=pdf

Your implementation must have a better bound than $O(N^3)$ worst-case for the whole calculation.

Same rules as the previous assignment--
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.

The test cases will be mixed with larger sparse graphs as well.

In the paper, we can observe that **BC** is accumulated in stages.
Your implementation must calculate the accumulation in increasing order of vertices i.e., 0 to N-1.
You must output 3 stages. 2 intermediate and final.
The intermediate versions are *after* the first accumulation and the middle accumulation.

To clarify, you must print all of BC_0, $BC_{\lfloor N/2 \rfloor}$ and the final BC for all v in G.

Input Format

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

Constraints

$N < 2000$

$0 < w_i \leq 2000$

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

$\forall u, v \in G, \quad \sigma_{uv} < 2^{32}$

Output Format

```
BC0(0)
BC0(1)
...
BC0(N-1)

BCN/2(0)
BCN/2(1)
```

```
...
BC_{N/2}(N-1)

BC(0)
BC(1)
...
BC(N-1)
```

> **NOTE 1:** Your output precision must be at least **1e-5**.
> **NOTE 2:** Print 1 entry per line (**check blank lines**). Otherwise, you might get "Wrong Answer" because of HR limitations.

Sample Input 0

```
9 12
1 4 1
5 8 1
4 7 1
1 2 1
0 3 1
7 8 1
6 7 1
3 6 1
4 5 1
0 1 1
3 4 1
2 5 1
```

Sample Output 0

```
0
1.5
0.25
1.5
1
0.25
0.25
0.25
0

1.083333333333333
3.833333333333333
0.6666666666666666
2.583333333333333
5.333333333333333
2.916666666666667
0.6666666666666666
1.666666666666667
0.25

1.333333333333333
5
1.333333333333333
4.999999999999999
10.66666666666667
5
1.333333333333333
5
1.333333333333333
```

Explanation 0

First block of numbers is $BC_0(0)...BC_0(N-1)$ i.e., after first stage.

Second block of numbers is $BC_4(0)...BC_4(N-1)$ i.e., after accumulating from 0 to 4.

Last block of numbers is the final/actual $BC(0)...BC(N-1)$

Submissions: 53
Max Score: 50
Difficulty: Medium

Rate This Challenge:

More

Current Buffer (saved locally, editable) C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
#include <stack>
#include <bits/stdc++.h>
using namespace std;


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

    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));
    }
    double Cb[N] = {};
    for(int s=0;s<N;s++){

    list<int> *pred;
    pred = new list<int> [N];
    stack <int> 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]={};

    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]){
            S.push(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] = 0;
                prQ.push(make_pair(distance[v1],v1));
                pred[v1].clear();
            }
            if(distance[v1] == (distance[uQc]+w)){
                paths[v1] = paths[uQc] + paths[v1];
               pred[v1].push_back(uQc);
            }
        }
    }

    double delta[N]={};
    while(!S.empty()){
        int w = S.top();
```

```
        S.pop();

        for (list<int>::iterator i=pred[w].begin(); i != pred[w].end(); ++i){
            int v = *i;
            delta[v] = delta[v] + ((paths[v]*(1.0))/paths[w])*(1+delta[w]);
        }


        if(w!=s){
            Cb[w] = Cb[w] + delta[w];
        }
    }

    if((s==0)||(s==(N/2))||s==(N-1)){
        for(int i=0;i<N;i++){
            printf("%.20f\n",(Cb[i]/2.0));
        }
        printf("\n");
    }

    }


    return 0;
}

```

Line: 1 Col: 1

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