USACO DEC10 Problem 'bigmac' Analysis

by Fatih Gelgi

The exchanges between currencies remind us of weighted directed graph structure: nodes as countries and the exchange rate e_ij as the weight of edge (i,j) where i and j are the nodes. Now, the problem asks the path that leads the minimum price for Big Mac. Observe that the problem is a variation of shortest path problem.

The difference is, the path length is not the sum of the weights on the path but product of them. Since the weights can be less than 1, there may be a cycle with value less than 1 in the graph. Then there will be no minimum since including the cycle in the path will continually reduce the length of the path.

(Note that it is guaranteed that a node is reacheable from any node). Consider a graph with nodes 1 and 2, and edges (1,2) is 3 whereas (2,1) is 0.2. What is the length of the path from 1 to 2?

```
Path: 1 -> 2                            length: 3
Path: 1 -> 2 -> 1 -> 2                          length: 3*0.2*3 = 1.8
Path: 1 -> 2 -> 1 -> 2 -> 1 -> 2       length: 3*0.2*3*0.2*3 = 1.08
```

This issue will be negative cycle problem in shortest path when logarithmic conversion is used for edges to calculate the path length. Hence, we cannot use Dijkstra's algorithm but Bellman-Ford is a good one. Below is the pseudocode for Bellman-Ford algorithm:

```
for i:= 1 to N-1
    for each edge (i,j) in the graph
        if distance[j] > distance[i] + e_ij
           distance[j]:= distance[i] + e_ij

for each edge (i,j) in the graph
    if distance[j] > distance[i] + e_ij
       output "there is a negative cycle"
```

In the code, distance[x] specifies the distance of node x from source node. Dijkstra greedily chooses the node with minimum distance in each step. However, Bellman-Ford doesn't have a particular order while updating the distances. Distances are updated by scanning through all the edges.

The algorithm continues until there is no update in the distances. In fact, the algorithm guarantees to finish in N-1 steps. To determine the negative cycles, algorithm has a post-processing step. If there is still a shorter path in the graph then that means there is a negative cycle.

Bellman-Ford requires O(VE) running time, and O(V+E) memory is needed for adjacency list.

Here is Sherry's code:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cmath>

#define INF 99999999999999999999999999.
#define MAXN 2000
using namespace std;

struct edge {
    int dest;
    double xchg;
    edge () {}
    edge (int a, double b) {
        dest = a;
        xchg = b;
    }
};

ifstream in ("bigmac.in");
FILE* fout;
int N, M, A, B;
long double V;
vector<edge> adj[MAXN];
long double dist[MAXN];
int parent[MAXN];

int main () {
    fout = fopen ("bigmac.out", "w");
    in >> N >> M >> V >> A >> B;
    A--, B--;
    for (int i = 0; i < M; ++i) {
        int a, b;
        double c;
        in >> a >> b >> c;
        /* take the logs of the exchange rates */
        adj[--a].push_back (edge (--b, log(c)));
    }

    /* *** Bellman-ford *** */
    /* initialize graph */
    for(int i = 0; i < N; ++i) {
        dist[i] = (i != A) * INF;
        parent[i] = -1;
    }
    /* relax edges repeatedly */
    for (int i = 0; i+1 < N; ++i) {
        for (int j = 0; j < N; ++j) {
            for (int k = 0; k < adj[j].size(); ++k) {
                int u = j;
                int v = adj[j][k].dest;
                if (dist[u] + adj[j][k].xchg < dist[v]) {
                    dist[v] = dist[u] + adj[j][k].xchg;
                    parent[v] = u;
                }
            }
        }
    }
    /* check for negative-weight cycles */
    for (int i = 0; i < N; ++i) {
        for(int j = 0; j < adj[i].size(); ++j) {
            int u = i;
            int v = adj[i][j].dest;
            if(dist[u] + adj[i][j].xchg < dist[v]) {
                fprintf(fout, "0\n");
                fclose(fout);
                return 0;
            }
        }
    }
    long double cost = V*exp(dist[B]);
    fprintf (fout, "%.6Lf\n", cost);
    return 0;
}
```