Luis Enrique Franco Marín

410921353

Week 10: Floyd-Warshall Algorithm

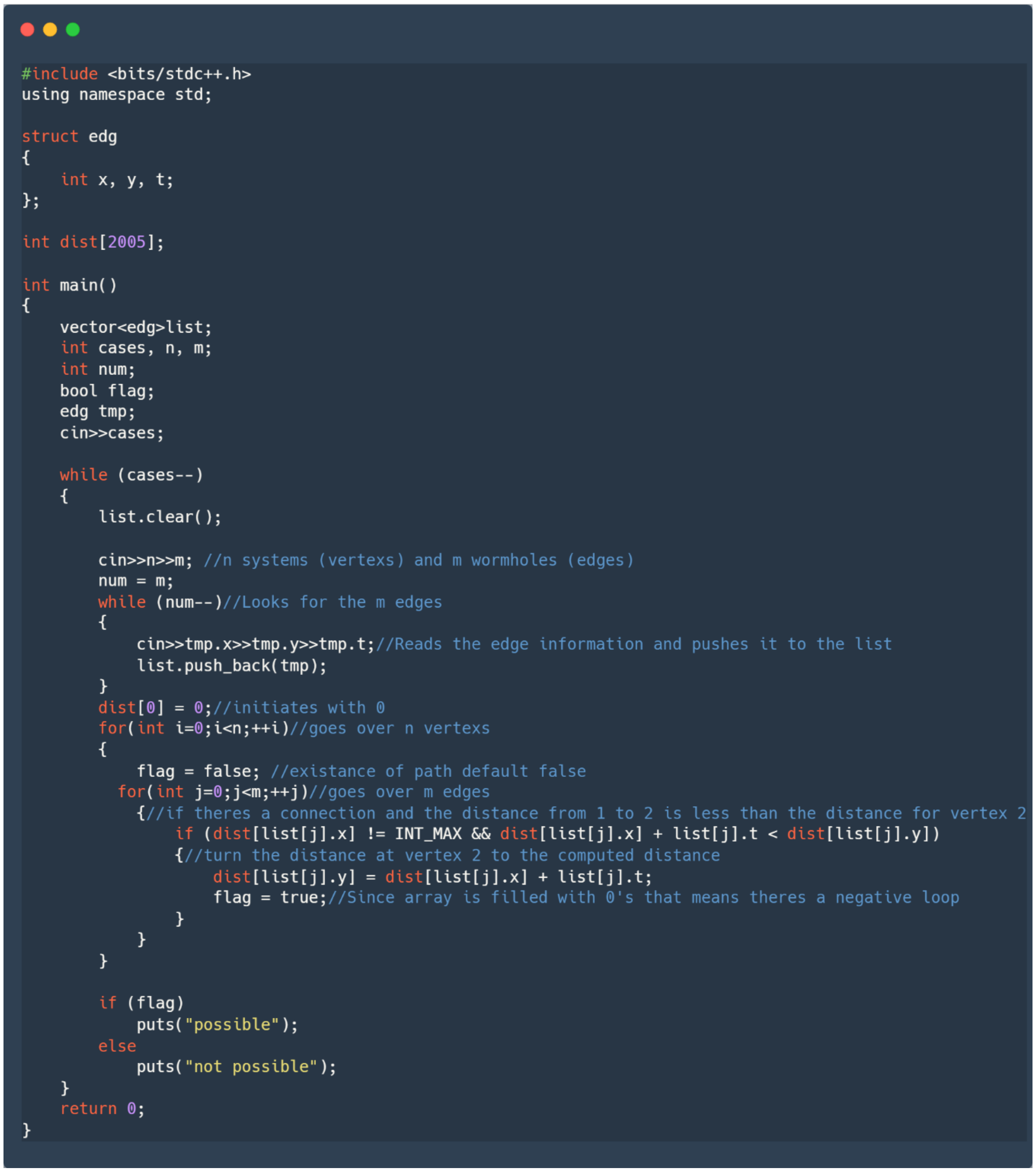
In a directed weighted graph with positive or negative edge weights, the Floyd–Warshall algorithm is used to identify the shortest pathways. The lengths (summed weights) of shortest paths between all pairs of vertices will be found in a single iteration of the algorithm. It is possible to reconstruct the paths with simple tweaks to the algorithm, even though it does not return details of the paths themselves.

Problem:

The input file starts with a line containing the number of cases cc to be analyzed. Each case starts with a line with two number n and m. These indicate the number of star systems (1≤n≤1000) and the number of wormholes (0≤m≤2000). The star systems are numbered from 0 (our solar system) through n−1. For each wormhole a line containing three integer numbers x, y and t is given. These numbers indicate that this wormhole allows someone to travel from the star system numbered xx to the star system numbered y, thereby ending up t (−1000≤t≤1000) years in the future.

The output consists of cc lines, one line for each case, containing the word possible if it is indeed possible to go back in time indefinitely, or 'not possible' if this is not possible with the given set of star systems and wormholes.

Code:



```cpp
#include <bits/stdc++.h>
using namespace std;

struct edg
{
    int x, y, t;
};

int dist[2005];

int main()
{
    vector<edg>list;
    int cases, n, m;
    int num;
    bool flag;
    edg tmp;
    cin>>cases;

    while (cases--)
    {
        list.clear();

        cin>>n>>m; //n systems (vertexs) and m wormholes (edges)
        num = m;
        while (num--)//Looks for the m edges
        {
            cin>>tmp.x>>tmp.y>>tmp.t;//Reads the edge information and pushes it to the list
            list.push_back(tmp);
        }
        dist[0] = 0;//initiates with 0
        for(int i=0;i<n;++i)//goes over n vertexs
        {
            flag = false; //existance of path default false
          for(int j=0;j<m;++j)//goes over m edges
            {//if theres a connection and the distance from 1 to 2 is less than the distance for vertex 2
                if (dist[list[j].x] != INT_MAX && dist[list[j].x] + list[j].t < dist[list[j].y])
                {//turn the distance at vertex 2 to the computed distance
                    dist[list[j].y] = dist[list[j].x] + list[j].t;
                    flag = true;//Since array is filled with 0's that means theres a negative loop
                }
            }
        }

        if (flag)
            puts("possible");
        else
            puts("not possible");
    }
    return 0;
}
```

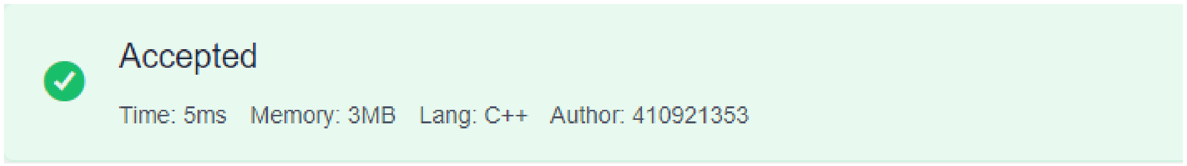
Discussion:

The algorithm initializes an array of size n filled with 0's and a list containing the m edges, each edge has 3 variables: the source vertex, end vertex, and the weight. The weight represents the time and since a negative time is required for the solution of the problem, a negative cycle has to be found. For this it is iterated over the n number of vertexes and the m number of edges. The Space for this is an array of size n vertexes and a list of m edges. The algorithm uses a nested loop, so the time complexity is n*m. It return true if there exists a negative loop, for this the time summed to get to another vertex should be less than the time subtracted by going to the "highest subtraction" path of that vertex.

Time: O(n*m)

Space: O(n+m)

Result:


Accepted

Time: 5ms Memory: 3MB Lang: C++ Author: 410921353