CSIS1119_COMP2119 Introduction to Data Structures and Algorithms
Programming_Assignment **Due Date:** 7pm, 24th April 2017

Please send email to **sghu@cs.hku.hk** if you have any questions about the description, judging rule, test cases and datasets et al.

Rules: discussion of the problems is permitted, but writing the assignment together is not (i.e. you are not allowed to see the actual pages of another student).

Course Outcomes

- **[O4]. Implementation**

1 Background

Finding better ways for cities to move, work, and thrive! says the Uber slogan. Nowadays Uber has already become a very famous company with business spread all over the world.

Routing is critical to enabling Uber to maximize utilization of drivers and ensure that passengers can always get a nearest car whenever and wherever they may need a ride.

To simplify our problem, we use the nearest road junction to approximate each car's and passenger's location. Each vertex is represented by an integer id. The map Uber uses has n vertices and m edges. Each vertex represents a road junction. The weight of edge between vertices u and v is $w(u, v) >= 0$, which corresponds to the length of the road segment in reality. For simplicity, we assume $w(u, v) = w(v, u)$ and $w(u, u) = 0$.

2 Where is the nearest car?

Mike is waiting for a ride at vertex u. There are l available drivers at locations $v_0, v_1, \ldots, v_{l-1}$.

Can you find the nearest driver for Mike?

2.1 input format

The first line is a string "NearestDriver".

The next line has two integers n and m. n denotes the number of vertices, m denotes the number of edges.

Each of the next m lines contains three integers i, j, w. i, j are the integer ids of two vertices. w denotes the distance between vertices i and j.

Then the next line has a single integer u representing Mike's location.

The next line has a single integer l, the number of cars.

The last line contains cars' locations, represented by the ids of l vertices, $v_0, v_1, \ldots, v_{l-1}$.

For all data, $0 <= i, j, u, v_{0,1,\ldots,l-1} < n, i \neq j, 0 <= w <= 10,000, \qquad 0 \leq l \leq 1,000,000;$

For 60% of data, $2 <= n <= 100, \qquad 0 <= m <= 1000;$

For 100% of data, $2 <= n <= 100,000, \qquad 0 <= m <= 1,000,000.$

Pairs (i, j) and (j, i) appear at most once in the input. Notice that the graph might not be connected.

2.2 output format

Output a single integer v, the nearest road junction id if the driver(s) at v can reach Mike otherwise "NO".
If there are multiple choices for v, output the smallest v.

2.3 samples

2.3.1 example1

input
NearestDriver
3 3
0 1 54
1 2 54
0 2 54
0
2
1 2
output
1

2.3.2 example2

input
NearestDriver
4 2
0 1 54
2 3 54
0
2
2 3
output
NO

3 How much will it cost?

People care about costs. People would balance the choice between Uber and MTR or other public transportation after they check how much they cost. So every day there are many queries about how much it costs to travel from a place s to place t. We can assume that cost is proportional to the travel distance, that is to say cost is \$$w$ HKD if the distance is w. You are asked to design an efficient algorithm for answering a batch of k queries. Assume that the map is static and k could be very large.

3.1 input format

The first line is a string "QueryPrice".

The next line has two integers n and m. n denotes the number of verticies, m denotes the number of edges.

Each of the next m lines contains three integers i, j, w. w denotes the distance between vertices i and j.

The next line contains an integer k. k is the number of queries.

Each of the next k lines contains two integers s and t denoting the start and destination vertices, respectively.

For all the data, $0 <= i, j, s, t < n, i \neq j, 0 <= w <= 10,000, \quad 1 \leq k \leq 1,000,000$;

For 60% of data, $2 <= n <= 50, \quad 0 <= k <= 1000$;

For 100% of data, $2 <= n <= 500, \quad 0 <= k <= 1,000,000$.

Pairs (i, j) and (j, i) appear at most once in edges. Notice that the graph might not be connected and s might equal t.

3.2 output format

For each query, output the cost if t is reachable from s otherwise "NO" in a new line.

3.3 samples

3.3.1 example1

input
```
QueryPrice
3 2
0 1 54
1 2 45
1
0 2
```
output
```
99
```

3.3.2 example2

input
```
QueryPrice
4 2
```

```
0 1 54
2 3 54
2
0 2
2 3
```
output
```
NO
54
```

4 What is the diameter of a graph?

Facebook, along with the University of Milan, organized a study in 2011. They analyzed the information from 721 million active members, and researchers found that the average number of connections from one randomly selected person to another was 4.74. And if you limit it to just the United States, it was 4.37.

Six degrees of separation, which has been around for over 80 years, is the idea that all living things and everything else in the world is six or fewer steps away from each other so that a chain of "a friend of a friend" statements can be made to connect any two people in a maximum of six steps.

Given a graph G, $D(G)$, denoting the diameter of G, is the greatest distance between any pair of vertices in G. For unweighted graph, six degrees of separation is essentially saying that $D(G) \leq 6$. Our objective is to determine the diameter of a given graph.

Given an undirected weighted graph, can you find its diameter.

4.1 input format

The first line is a word “Diameter”.

The next line has two integers n and m. n denotes the number of vertices, m denotes the number of edges.

Each of the next m lines contains three integers i, j, w. w denotes the distance between vertices i and j.

For all data, $0 <= i, j < n, i \neq j, 0 <= w <= 10000, \qquad 0 <= m <= \frac{n(n-1)}{2}$;

For 60% of data, $2 <= n <= 50$;

For 100% of data, $2 <= n <= 500$.

Pairs (i, j) and (j, i) appear at most once in edges. Notice that the graph might not be connected.

4.2 output format

Output an integer, the diameter of the input graph. If the graph is not connected output "INF"

4.3 samples

4.3.1 example1

input

```
Diameter
3 2
0 1 54
1 2 45
```

output

```
99
```

4.3.2 example2

input

```
Diameter
4 2
0 1 54
2 3 45
```

output

```
INF
```

5 Less accurate but much faster!

In the previous section we may find it is time-consuming to decide the exact diameter for larger graphs. However it can be much quicker to get an approximate answer if we trust the following claims.

Claim 5.1 *Given a weighted undirected graph $G := (V, E)$ with non-negative edge weight, if we randomly pick a starting vertex s and find it's furtherest vertex u. The distance between s and u is at least half of $D(G)$.*

Claim 5.2 *If G is a tree, we randomly pick s and find u in the same way as in Claim 5.1. If we further find u's furtherest vertex v. The distance between u and v is $D(G)$.*

Can you prove whether the above claims are true or not?

Given a graph G, can you write a program to give an approximate answer for G? Remember your program should return the exact answer when G is a tree.

Notice. You may find Claim 5.2 useful to test the correctness of your code.

5.1 input format

The first line is a word "DiameterApproximation".

The next line has two integers n and m. n denotes the number of vertices, m denotes the number of edges.

Each of the next m lines contains three integers i, j, w. w denotes the distance between vertices i and j.

For all data, $0 <= i, j < n, i \neq j, 0 <= w <= 10000$

For 60% of data, $2 <= n <= 500$,

For 100% of data, $2 <= n <= 10,000 \qquad 0 <= m <= 1,000,000$.

Pairs (i, j) and (j, i) appear at most once in edges. Notice that the graph might not be connected.

5.2 output format

Output an integer, the diameter of the input graph. If the graph is not connected, output "INF"

5.3 samples

5.3.1 example1

input
DiameterApproximation
3 2
0 1 54
1 2 45
output
99

5.3.2 example2

input
DiameterApproximation
4 2
0 1 54
2 3 54
output
INF

6 Others

Languages. We only accept C/C++ programming languages.

Judging. Please note that your solution is automatically judged by a computer. Solutions that fail to compile or execute get 0 points. We have designed 10 test cases for each of the 4 problems. Every test case worths 2.5 points. The execution time limit for each test case is 1 second. For each test case, you will get 2.5 points if your program passes it otherwise you will get 0.

Self Testing. You should test your program by yourself using the provided sample input/output file. The sample input/output is **different** from the ones used to judge your program, and it is designed for you to test your program by your own. Note that your program should always use standard input/output. To test your program in Windows:
1. Compile your program, and get the executable file, “main.exe”
2. Put sample input file, “input.inp”, in the same directory as “main.exe”
3. Open command line and go to the directory contains “main.exe” and “input.inp”
4. Run main.exe < input.inp > myoutput.oup
5. Compare myoutput.oup with sample output file.
6. Your output needs to be **exactly** the same as the sample output file.

To test your program in Linux or Mac OS X:
1. Put your source code “main.cpp” and sample input file “input.inp” in the same directory.
2. Open a terminal an go to that directory.
3. Compile your program by “g++ main.cpp -o main”
4. Run your program using the given input file, “./main < input.inp > myoutput.oup”
5. Compare myoutput.oup with sample output file.
6. Your output needs to be **exactly** the same as the sample output file.
7. You may find the **diff** command useful to help you compare two files. Like “diff -w myOutput.inp sampleOutput.oup”. The $-w$ option ignores all blanks (SPACE and TAB characters)

Note that myoutput.oup file should be **exactly** the same as sample output. Otherwise it will be judged as wrong.

Sketch file. We provide a source file as a sketch. You write your code based on that. You can also write your own code as long as the format is correct.

Test files. We put the test cases in under *test* directory. *.inp are input files and *.oup are sample answer.

Real-world dataset. We also provide two real world datasets for you to play with, one has 500 vertices and another has 1000 vertices. *real_data/readme.txt* gives detailed explanation on data format.

Store graph You may notice that the number of vertices and edges vary a lot between different problems. You may use different data structures to store the graph for different questions.

Submission. Please submit your source file through moodle. You should submit the source file only (**without** compression). Please write your code in one file named in format of *university_number.cpp*, e.g. *1234567890.cpp*. **Do not use multiple files**.

sketch.cpp

```
#include <iostream>
#include <string>
#include <assert.h>


using namespace std;


void NearestDriver(){
    int n, m;

    cin >> n >> m;

    Graph g(n); // implement Graph class by yourself

    for(int i = 0; i < m; i++){
        int a, b, w;
        cin >> a >> b >> w;
        // read edge and insert it to graph
        g.addEdge(a, b, w);
    }

    int u;
    cin >> u;
    // implement your own shortest path algorithm
    g.shortestPath(u);

    int bestv = -1;
    int l;
    cin >> l;
    for(int i = 0; i < l; i++){
        // scan over every car
        int v;
        cin >> v;

        // change bestv when v is better than bestv
    }

    if(bestv == -1)
        cout << "NO" << endl;
    else
        cout << bestv << endl;
    return;
}


void QueryPrice(){
    //your code starts here
}

void Diameter(){
    //your code starts here
}

void DiameterApproximation(){
    //your code starts here
}

int main(){
    string section;
    cin >> section;

    if(section == "NearestDriver")
        NearestDriver();
    else if(section == "QueryPrice")
        QueryPrice();
    else if(section == "Diameter")
        Diameter();
    else if(section == "DiameterApproximation")
        DiameterApproximation();
    else{
        cout << "wrong␣input␣file!" << endl;
        assert(0);
    }

    return 0;
}
```