COMP2119 Introduction to Data Structures and Algorithms
Programming Assignment **Due Date:** 7pm, 30th October 2017

Please send email to **xli2@cs.hku.hk** if you have any questions about the description, judging rule, test cases, or datasets.

Course Outcomes

- **[O4]. Implementation**

1 Modeling

In this programming assignment, we implement and modify an elementary graph-based algorithm call the Dijkstra's algorithm, which has applications in location-based road-network systems. For simplicity, we model a road network as an undirected graph that consists of n vertices and m edges. Vertices model road junctions, each of which is given a numeric id from 1 to n. An edge that connects a vertex u to another vertex v models a road segment that connects the road junctions represented by u and v. An edge is denoted by a pair (u, v) and is given a weight $w(u, v) \geq 0$ that models the traveling distance of the road segment from u to v. For simplicity, we assume $w(u, v) = w(v, u)$ and $w(u, u) = 0$.

2 Problem 1: Where is the nearest gas station?

Given a road network, let us assume that there are k gas stations that are located at k different road junctions of the networks. Suppose there is a car at a road junction v, whose tank is low. We want to help the driver identify the nearest gas station to fill up his tank.

Hint: This problem can be solved by the Dijkstra's Algorithm, which solves the single-source shortest path problem. Given a vertex v, Dijkstra's find the shortest path from v to each node in the graph. In the process, Dijkstra's expands its exploration of the graph, starting with the source vertex v, and gradually reaching out to other nodes in a near-to-distant order. Hence, if we consider the location of a car as vertex v and performs Dijkstra's with it being the source vertex, the shortest path of the nearest gas station will generally be discovered before those of the distant stations. You should read Sections 25.1 and 25.2 of the textbook and attend the tutorial on Oct. 13 for the details of Dijkstra's.

Note (1): Given a source vertex v, if there are two or more gas stations that are at equal distance from v, the gas station with the smaller id is picked.

3 Problem 2: Will cars come to my station?

Suppose you are the owner of a gas station and would like to learn if your station is strategically well located. You thus want to know, among all the road junctions in the road networks, the number of such junctions at which a car in need of gas will come to your station. Specifically, let u be the vertex of your gas station. A road junction, as represented

by a vertex v, is said to be the *reverse nearest neighbor* of u if u is the nearest gas station of v. (Hence, a car located at v will go to u, your station, for gas.) Given u, we want to compute the number of u's reverse nearest neighbors.

Hint: The problem can be solved by determining, for each road junction v, the nearest gas station of v. In other words, you can run n Dijkstra's, one for each vertex, and determine the number of times u shows up as the nearest gas station. This approach, however, is very slow, considering that a typical road network consists of many road junctions (e.g., there are about 1.3 million road junctions in the Beijing road network). The problem can be solved by doing Dijkstra's only k times. Let's see if you can come up with such a strategy.

4 Program

Your program should implement the functionality of answering both Problems (1) and (2) above. The I/O specifications are given below:

4.1 Input Format

The input file is a text file. It consists of 2 parts.

The first part is a specification of the road network:

Line 1: two integers separated by a comma in the form

```
n,m
```

where n denotes the number of vertices and m denotes the number of edges.

Line 2: an integer:

```
k
```

where k denotes the number of gas stations.

Line 3: a list of k comma-separated distinct numbers in $[1..n]$:

```
n1,n2, ..., nk
```

where each n_i represent the road junction at which gas station i is located.

Line 3+1 to 3+m: each line contains three comma-separated integers of the form:

```
i,j,w
```

where $i, j \in [1..n]$ are two distinct road junctions id's, and w gives $w(i, j)$, the length of the road segment (i, j). You can assume that no road segment is mentioned twice in the input file.

The second part of the input file is a problem specification.

For Problem (1):

Line 3+m+1 is the line:

```
NearestGasStation
```

Line 3+m+2 is a single integer:

```
v
```

where v is the road junction at which a car is located. The task is to find the nearest gas station for v.

For Problem (2):

Line 3+m+1 is the line:

```
ReverseNeighbors
```

Line 3+m+2 is a single integer:

```
u
```

where u is the road junction at which your gas station is located. The task is to find the number of reverse neighbors of u in the road network.

4.2 Output Format

For Problem (1): output the nearest gas station id. Again, in case there are multiple gas stations that are at the same distance form v, output the one with the smallest id. v for the given car u if v exists, otherwise, especially in the case

For Problem (2): output the number of reverse nearest neighbors of u.

5 Sample Inputs and Outputs

Example 1:

input
4,4
2
1,2
1,2,54
2,3,54
1,3,54
1,4,52
NearestGasStation

3
output
1

Example 2:

input
4,3
3
1,2,3
1,3,54
2,3,10
2,4,54
NearestGasStation
4
output
2

Example 3:

input
4,4
2
2,3
1,2,54
1,3,45
2,4,12
1,4,5
ReverseNeighbors
2
output
3

Example 4:

input
4,3
2
1,2
1,3,54
1,4,54
2,3,10
ReverseNeighbors
2
output
2

6 Others

Languages. We only accept 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 5 test cases for each of the 2 problems. Every test case worths 10 points. For each test case, you will get 10 points if your program passes it otherwise you will get 0. The full mark is 100.

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.txt", in the same directory as "main.exe"
3. Open command line and go to the directory contains "main.exe" and "input.txt"
4. Run main.exe < input.txt > output.txt
5. Compare output.txt 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.txt" 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.txt > output.txt"
5. Compare output.txt 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 output.txt sampleOutput.txt". The $-w$ option ignores all blanks (SPACE and TAB characters)

Note that output.txt 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 under *test* directory. input*.txt are input files and output*.txt are sample answer.

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;


int nearestGasStation(int v){
    //your code starts here
}


int calculateNumofReverseNeighbors(int u){
    //your code starts here
}

int main(){

    int n, m;
    cin >> n >> m;

    int k;
    cin >> k; // the number of gas stations

    Graph g(n,m,k); // implement Graph class by yourself

    String stationIds;
    cin >> stationIds;

    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);
    }

    string section;
    cin >> section;

    if(section == "NearestGasStation")
        int v;
        cin >> v;
        NearestGasStation(v);
    else if(section == "ReverseNeighbors")
        int u;
        cin >> u;
        calculateNumofReverseNeighbors(u);
    else{
        cout << "wrong␣input␣file!" << endl;
        assert(0);
    }
```

```
    return 0;
}
```