# VE281 — Data Structures and Algorithms

*Programming Assignment 4*

Instructor: Hongyi Xin
— UM-SJTU-JI (SU 2022)

**Notes**

- Due Date: 7/31 23:59
- Submission: on JOJ

# 1 Introduction

In this project, you are asked to implement an algorithm to find the shortest path in a given directed graph. The detailed data structure and algorithm you use are up to your choice.

You will gain experiences about choosing and designing algorithms under required specifications from this project.

# 2 Programming Assignment

## 2.1 Input and Output

You are expected to read input from `<stdin>` and print output through `<stdout>`. The input will provide you a graph, as well as subsequent queries which are pairs of sources and destinations. Your output should provide the shortest distance of each query.

### 2.1.1 Input Specification

The first line in the input specifies the number of nodes $|V|$ in the graph $G = (V, E)$, where the nodes in the graph are indexed from 0 to $|V| - 1$. The second line specifies the number of edges, assumed as $E$, in the graph. Each of the subsequent $E$ lines represents a **directed** edge by 3 numbers in the form:
`<start_node> <end_node> <weight>`

where both `<start_node>` and `<end_node>` are integers in the range $[0, |V| - 1]$, representing the start node and the end node of the edge, respectively, and `<weight>` is an **integer** representing the edge weight. Here you can assume that there will be no parallel edges in this input sequence.
After $E$ lines of edges, the program will be fed with a series of queries. Each query line represents a pair of source and destination nodes that we want to find shortest path from the source node to the destination node, in the form:
`<source_node> <destination_node>`

The graph specified in our input format is a **directed graph with integer edge weights**. Notice that the edge weights might be negative in some inputs.
Hint: When some edges with negative weights exist in the graph, are there possibilities that the shortest path between two nodes do not exist, i.e. will there exist a situation where the source and the destination nodes are connected, but **the shortest path between connected nodes becomes infinite**?

We define a graph to be **invalid** if there exists a pair of source and destination nodes (not necessarily the input source-destination pairs) that are connected but the shortest path is not finite. The graph is **valid** otherwise.

### 2.1.2 Terminate Condition

Your program should exit when a) the graph is invalid or b) the `<source_node>` in a query pair is -1. When the graph is invalid, your program is expected to output:

```
Invalid graph. Exiting.
```

### 2.1.3 Example

Here is an example of a valid graph and a few subsequent queries:

```
4
4
0 1 5
1 2 -1
2 3 4
3 1 6
1 3
1 2
3 0
0 3
-1 0
```

It represents the following directed graph with 4 pairs of source and destination node:

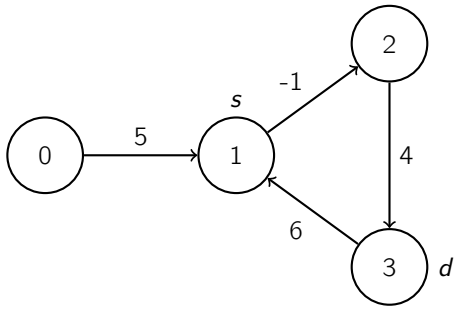


Figure 1: Sample Input

The input will be provided from the **standard input**.

### 2.1.4 Output Specification

Your program writes to the **standard output**. Your program should first read the input graph, then respond to each of the input source-destination pairs. During the computation of shortest paths, you might find that the input graph is invalid. For invalid graphs, you should not output any of the calculated shortest path results, but instead print the warning information and exit the program.

#### a) Invalid Graph

If the graph is invalid, your program should print:

```
Invalid graph. Exiting.
```

And **exit the program as soon as possible** (Hint: you should have enough knowledge about this before any queries).

#### b) Valid Graph

If the graph is valid, you should respond to **each input line** of the source-destination pair sequentially in the output. The detailed requirements for each input line are as follows.
If the input graph is valid and the shortest path exists for the input source-destination pair, your program should print the length of the shortest path. Specifically, it should print the following line for each query:

`<length>`

where `<length>` specifies the length of the shortest path. In the special case where the source node is the same as the destination node, the shortest path length is 0.

When the source is not connected with the destination (when no path exists), the program should output the text *INF*.
The output for previous example should be:

```
3
-1
INF
8
```

For more information related to the input and output of the whole program, you may refer to the provided `main.cpp`, which is identical to what we will use to test your program. Do not include this `main.cpp` in your submission file.

## 2.2 Methods to implement

In this project, you are asked to implement your algorithm with two methods: `readGraph()` and `distance()`. However, you are free to add internal helper functions up to your choice.

### 2.2.1 Graph reading

In the method `readGraph()`, you should take all the inputs related to the graph and store it in the data structure you choose. Initialization of the graph should be done within this method.
Notice that the queries are **not** read here.

### 2.2.2 Shortest path finding

In the method `distance()`, you should output the length of the shortest path found for the current query in the form specified in 2.1.2.

## 2.3 Data Structure and Algorithm

In this project, we do not explicitly ask you to use specific data structures or algorithms. You are free to modify the definition of the `Graph` class in `shortestP2P.hpp` by adding internal data and functions, but do not change the name of the class as well as the methods specified in 2.2.

# 3 Implementation Requirements and Restrictions

## 3.1 Error checking

You do not need to do any error checking of the input graph except for its validness and you can assume that all the inputs are syntactically correct.

## 3.2 Requirements

- You must make sure that your code compiles successfully on a Linux operating system with `g++` and the options `-std=c++1z -Wconversion -Wall -Werror -Wextra -pedantic`.
- You should only hand in `shortestP2P.hpp`.
- You are not allowed to include additional libraries in `shortestP2P.hpp`. Do not submit `main.cpp`.
- Output should only be done where it is specified.

## 3.3 Memory Leak

You may not leak memory in any way. To help you see if you are leaking memory, you may wish to call valgrind, which can tell whether you have any memory leaks. (You need to install valgrind first if your system does not have this program.) The command to check memory leak is:

```
valgrind --leak-check=full <COMMAND>
```

You should replace `<COMMAND>` with the actual command you use to issue the program under testing. For example, if you want to check whether running program

```
./main < input.txt
```

causes memory leak, then `<COMMAND>` should be "`./main < input.txt`". Thus, the command will be

```
valgrind --leak-check=full ./main < input.txt
```

# 4 Grading

Your program will be graded along five criteria:

## 4.1 Functional Correctness

Functional Correctness is determined by running a variety of test cases against your program, checking your solution using our automatic testing program.

## 4.2 Implementation Constraints

We will grade Implementation Constraints to see if you have met all of the implementation requirements and restrictions. In this project, we will also check whether your program has memory leak. For those programs that behave correctly but have memory leaks, we will deduct some points.

### 4.3 General Style

General Style refers to the ease with which TAs can read and understand your program, and the cleanliness and elegance of your code. Part of your grade will also be determined by the performance of your algorithm.

### 4.4 Performance

We will test your program with some large test cases. If your program is not able to finish within a reasonable amount of time, you will lose the performance score for those test cases.

## 5 Acknowledgement

This project is jointly designed by VE281 FA21 teaching group.