BLG 336E - Analysis of Algorithms II
2019/2020 Spring
Final Project

- You should write all your code in C++ language. Your code should be run with the command line arguments specified for each question.
- Your code should be able to be compiled with default g++ compiler and run under Ubuntu OS. **Even if you are writing your code on a different OS, you should check it via ITU SSH.**
- This is a Final Course Project Assignment, cheating is absolutely unethical and morally unacceptable. It will be punished by a negative grade. Also disciplinary actions will be taken.
- For every part of the homework, programs should be run with different command line arguments. **The codes not using these arguments or giving output in a different layout will not be graded**.

1 Finding Shortest Paths with Dynamic Programming (25 pts)

Implementation [20 pts]
In this question, you are expected to implement the Bellman-Ford algorithm, which is used for finding the shortest paths in a graph that contains edges with negative weights [1]. Your algorithm should follow the equation 1 and pseudo-code in Algorithm 1 (Figure 1). Please check the textbooks [1], [2] and course slides for further details on this algorithm.

$$OPT(i,v) = \begin{cases} 0, & \text{if } i = 0 \text{ and } v = t \\ \infty, & \text{if } i = 0 \text{ and } v \neq t \\ \min\{\ \text{OPT(i - 1, v)},\ \min\limits_{(v,w)\in E}\{OPT(i-1, w + l_{vw})\}\}, & \text{if } i \geq 0 \end{cases} \quad (1)$$

A skeleton code **q3.cpp** and two test files **q3_test1.txt** and **q3_test2.txt** are provided with the question. The skeleton code already includes class definitions, and functions related to file reading and output formatting. Therefore, you only need to fill the empty functions related to Bellman-Ford.

Algorithm 1: Shortest-Path(G, s, t)

```
n = number of nodes in G;
Array M[0 ... n - 1, V];
Define M[0, t] = 0;
Define M[0, v] = ∞ for all other v ∈ V;
for i = 1 to n - 1 do
    for v ∈ V in any order do
        Compute M[i, v] using the equation 1
    end
end
Return M[n - 1, s];
```

Figure 1: Pseudo code for Bellman-Ford algorithm [1]

Each test file includes a single graph data. The first row in each file represents the number of nodes, and the second row represents the number of edges, whereas the rest of the rows represents the edges. Each edge includes three items separated by a space character: a source node, a destination node, and weight information, respectively. Each node is represented with a number, i.e., 0 or 2.

The output of your code should contain 1) the filled dynamic programming table, 2) shortest path value between the given nodes, 3) shortest path between the given nodes, and 4) a warning if there is a negative weight cycle in the graph (please check the coursebook to learn about negative weight cycles [1]). Your implementation will be graded as follow:

- Filling the dynamic programming table and finding the shortest path value between given two nodes [7 pts]
- Finding the shortest path between given two nodes [7 pts]
- Detecting the negative weight cycles in the graph [6 pts]

You can compile and test your implementation with the below commands. Figure 2 Figure 3 show the expected outputs for the given test cases.

```
g++ -std=c++11 q3.cpp -o q3
./q3 q3_test1.txt
./q3 q3_test2.txt
```

Report [5 pts]
Compare the Bellman-Ford algorithm with the other two classical shortest path algorithms: Dijkstra and Floyd-Warshall [3], [2]. Discuss the basic properties, advantages, and disadvantages of these algorithms in your own words using a maximum of 300 words.

```
************* Graph data *************
Number of total edges: 10
Number of total nodes: 6

Source node --> Destination node : Edge weigth
0 --> 5 : -3
0 --> 1 : -4
3 --> 0 :  6
3 --> 5 :  4
1 --> 3 : -1
1 --> 4 : -2
2 --> 1 :  8
2 --> 5 :  3
4 --> 2 : -3
4 --> 5 :  2
************************************

Enter the destination node: 5

Dynamic programming table: (shortest paths from each node to destination node):

               Node-0  Node-1  Node-2  Node-3  Node-4  Node-5
Iteration-0      ∞       ∞       ∞       ∞       ∞       0
Iteration-1     -3       ∞       3       4       2       0
Iteration-2     -3       0       3       3       0       0
Iteration-3     -4      -2       3       3       0       0
Iteration-4     -6      -2       3       2       0       0
Iteration-5     -6      -2       3       0       0       0

Enter a source node: 3

Shortest path value from 3 to 5: 0
Shortest path from 3 to 5: 3 -> 0 -> 1 -> 4 -> 2 -> 5
```

Figure 2: Output for q3_test1.txt.

```
Number of total edges: 6
Number of total nodes: 6

Source node --> Destination node : Edge weigth
0 --> 1 :  6
1 --> 2 :  3
2 --> 3 :  4
2 --> 5 :  5
3 --> 1 : -9
3 --> 4 :  3
**************************************

Enter the destination node: 5
Graph contains negative weight cycle!
```

Figure 3: Output for q3_test2.txt.

Bibliography

[1] J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education India, 2006. 1, 2

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009. 1, 2

[3] R. W. Floyd, "Algorithm 97: shortest path," *Communications of the ACM*, vol. 5, no. 6, p. 345, 1962. 2