BLG 336E - Analysis of Algorithms II
2020/2021 Spring
Final Project
Question 1

- Please submit your homework only through Ninova. Late submissions will not be accepted.
- Please do not forget to write your name and student ID on the top of each document you upload.
- You should write your code in C++ language and try to follow an object-oriented methodology with well-chosen variables, methods, and class names and comments where necessary.
- Your code should compile on Linux using g++. You can test your code through ITU's Linux Server.
- Because your codes will be processed with an automated tool, `Calico`, make sure that your output format matches the given sample output.
- You may discuss the problem addressed in the homework at an abstract level with your classmates, but you should not share or copy code from your classmates or any web sources. You should submit your individual homework. In case of detection of an inappropriate exchange of information, disciplinary actions will be taken.
- If you have any questions, please ask on related message board named **Final Exam Q1 Questions** for this question.

1 Widest Path with Modified Dijkstra Algorithm (25 pts)

Implementation [20 pts]
In this question, you are required to implement the Modified Dijkstra algorithm proposed by Wei et al. [1]. This algorithm is created to solve the problem of finding the maximum capacity path, also known as the widest path or the maximum bottleneck path. In a weighted graph, maximum capacity path is the path between two specific

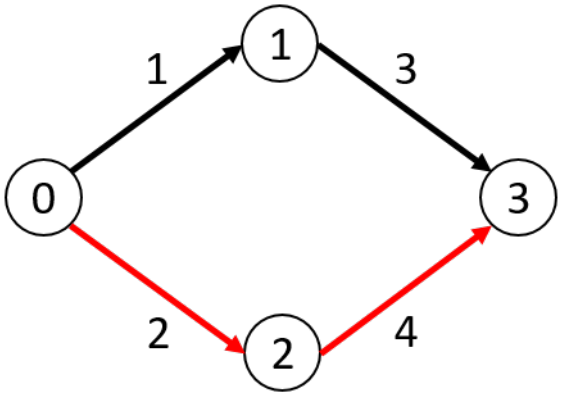


Figure 1: In this weighted graph, the widest path has the vertices 0-2-3.

vertices whose bottleneck edge (the edge with the minimum weight) is the maximum. Figure 1 depicts an example weighted graph whose starting and end vertices are 0 and 3, respectively. In this graph, the widest path is the red one because the bottleneck edge of this path is 2 while the bottleneck of the alternative path (the black one) is 1.

Algorithm 1: Modified Dijkstra Algorithm [1]

```
Input: a graph G, a source vertex s and a destination vertex t
Output: a path from s to t with the maximum capacity
for each vertex v do
    status[v] = 0; wt[v] = -1; dad[v] = -1;
end
status[s] = 2; wt[s] = INF;
for each edge [s,w] do
    status[w] = 1; wt[w] = weight(s,w); dad[w] = s;
end
while there are fringes do
    v = fringe with the maximum wt-value;
    status[v] = 2;
    for each edge [v,w] do
        if status[w] == 0 then
            status[w] = 1;
            wt[w] = min{wt[v], weight(v,w)};
            dad[w] = v;
        else if status[w] == 1 and wt[w] <min{wt[v], weight(v,w)} then
            wt[w] = min{wt[v], weight(v,w)};
            dad[w] = v;
    end
end
```

Algorithm 1 shows the pseudo-code of the algorithm. In this algorithm, **"status"** is the

array (or vector, depending on your preference) whose size is equal to the number of nodes in the graph. This array keeps track of the current status of each node: 0 means the vertex has not been visited yet; 1 means "fringe", indicating the node is visited but has not reached the final status yet; 2 means all operations are done on that vertex. The second array (or vector, depending on your preference) of the algorithm is named **"wt"** whose size is the same as the number of vertices in the graph. It holds the minimum edge weight on the path which starts from the starting vertex and ends at the corresponding vertex. The third array (or vector, depending on your preference) is called **"dad"** and it contains the vertex that comes right before the corresponding vertex, in other words it keeps the parent of each node. For your implementation, you can use arrays, vectors or any other data structure you choose. You can check the original source [1] for a detailed explanation and the process of the algorithm on a sample graph.

A skeleton code **q1.cpp** and two test files **test1.txt** and **test2.txt** are provided with the question. The skeleton code already includes class definitions, and code snippets related to output formatting. Therefore, you need to fill the empty functions related to file reading and Modified Dijkstra algorithm. If you find necessary, you can add extra functions.

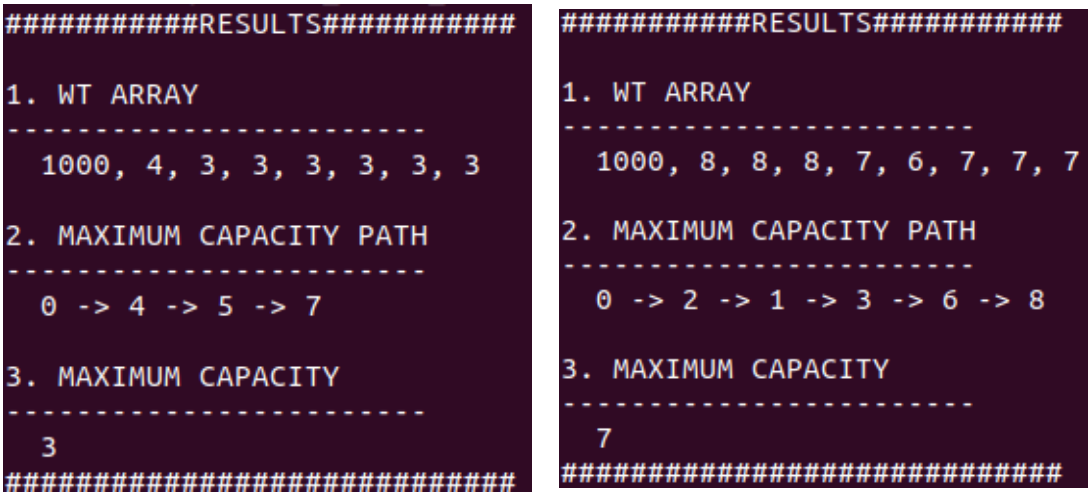
Each test file includes a single undirected graph data. The first row in each file represents the number of nodes and the rest of the rows represent the edges. Each edge includes three items separated by a tab ("\t") character: a source node, a destination node, and weight information, respectively. Each node is represented with a number, i.e., 0 or 2. The source node of the path must be 0 and the destination node must be the node with the largest number.

The output of your code should contain 1) the final content of the "wt" array (see the pseudo-code) which contains the smallest edge weight from stating vertex to the corresponding vertex [1], 2) maximum capacity path including the source and destination vertices, 3) the bottleneck (minimum edge) of the path. Your implementation will be graded as follows:

- Finding the correct final "wt" array [7 pts]
- Finding the maximum capacity path between the source and destination nodes [9 pts]
- Finding the maximum capacity of the graph [4 pts]

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

```
g++ -std=c++11 q1.cpp -o q1
./q1 test1.txt
./q1 test2.txt
```



(a) Output for "test1.txt" (b) Output for "test2.txt"

Figure 2: Console output for given test cases.

Your projects will be checked with an automated tool, namely Calico. Along with the text files that contain sample graphs, a test file is also submitted ("test.t"). You can test your source code yourself to see if it passes the given test cases (case 1 is for test1.txt, case 2 is for test2.txt), before submitting. This way, you can avoid possible errors that arise from simple character problems.

Report [5 pts]

- Give 2 examples for real world applications of maximum capacity path algorithms. If you benefit from a source, obey citation rules [3 pts].
- Analyze the time complexity of Modified Dijkstra algorithm. If you benefit from a source, obey citation rules [2 pts].

Bibliography

[1] K. Wei, Y. Gao, W. Zhang, and S. Lin, "A modified dijkstra's algorithm for solving the problem of finding the maximum load path," in *2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT)*. IEEE, 2019, pp. 10–13. 1, 2, 3