Assignment: Dijkstra Shortest Path Algorithm

1. Approach

To create the algorithm, I followed the suggested approach in the assignment brief. Three different classes are constructed: Graph, PriorityQueue and ShortestPath. Graph and PriorityQueue classes can be found in the header files graph.h and queue.h. The ShortestPath class can be found on the shortest.cpp, which #include graph.h and queue.h.

STLs are used to implement these classes to carry out basic operations. For example, using vector and its operations for storing and retriving information. However, STLs that provide complex functionalities to generate graph, implement priority queue and find shortest path out-of-the-box are avoided. For example, there is an existing STL for priority queue which was not used for this project. Having to implement these functionalities on my own ensures that I understand their purpose and behaviour.

2. Classes

Graph. To implement the Graph class, I referred to the following source:

> Geeks for Geeks (2020, Sep 8). *Graph and Its Representations*. https://www.geeksforgeeks.org/graph-and-its-representations/

The graph class helps to generate random graph based on the desired number of nodes and graph density. It also provides methods for basic graph operations such as inserting new nodes, getting the total number of edges in the graph and getting the edge value between 2 nodes. For this class, I use vector for pairs to track the edges and their respective value.

PriorityQueue. To implement the PriorityQueue class, I referred to the following source:

> Geeks for Geeks (2021, March 8). *Binary Heap*. https://www.geeksforgeeks.org/binary-heap/

The priority queue class stores nodes and their corresponding value. It is based on min. heap implementation and therefore, the top element is one with the least node value. This class supports basic operations such as insertion and deletion. During insert and delete, the class will re-organize itself to maintain the min. heap structure.

ShortestPath. This class utilizes both the graph and priority queue classes to implement Dijkstra's shortest path algorithm. In this class, I also implemented a container to store shortest path results, allowing the computation of average shortest path after running the shortest path algorithm from the node in focus to every other nodes.

3. Lessons

Importance of Data Structure. Coming from a Python background, C++ is much less forgiving and more demanding in ensuring that the data type are declared correctly. One of the issue I faced when starting the assignment is deciding how the private data members should be declared. For instance, for the graph class, my approach requires me to initialize a vector with a given size. However, initiating this directly with a dynamic size is not allowed. After a lot reading and research, the best solution to this turns out to initiate a pointer variable to a vector and then initialize the vector with a given size using `new` in the constructor method.

Pointers and Memory Management. Another thing I have to pay extra attention to is to keep track of which variables are pointers. The available operations required between pointers and non-pointers are different. Extra caution must also be given when deciding what data type should be returned by a method as it can affect how its output can be processed by other functions.

Error Message Clarity. One of biggest challenge in coding in C++ for me is figuring out how to fix an error. This is especially true when the error pertains to memory related error which usually manifested in segmentation error. Segmentation error does not provide any traceback and therefore I have to review the code line by line to fix this error. It is possible that there is a better approach to solving segmentation error, however, I was not able to find a better approach to solve this.

4. Program Printout

```
Finding average shortest path from Node 0
0 -> 1 : 4.7
0 -> 2 : 3.2
0 -> 3 : 5.2
0 -> 4 : 5.4
0 -> 5 : 7.1
0 -> 6 : 3.1
0 -> 7 : 3
0 -> 8 : 6.8
0 -> 9 : 12.9
0 -> 10 : 10.9
0 -> 11 : 5.7
0 -> 12 : 4.5
0 -> 13 : 11.9
0 -> 14 : 3.6
```

```
0 -> 15 : 15.4
0 -> 16 : 11.6
0 -> 17 : 5.9
0 -> 18 : 7.6
0 -> 19 : 9.5
0 -> 20 : 11.4
0 -> 21 : 17.4
0 -> 22 : 7.5
0 -> 23 : 5.5
0 -> 24 : 8.2
0 -> 25 : 8
0 -> 26 : 1
0 -> 27 : 10.9
0 -> 28 : 6.2
0 -> 29 : 7.8
0 -> 30 : 5.7
0 -> 31 : 5.7
0 -> 32 : 6.8
0 -> 33 : 5.7
0 -> 34 : 5.4
0 -> 35 : 2.8
0 -> 36 : 2.6
0 -> 37 : 9.1
0 -> 38 : 4.8
0 -> 39 : 10.6
0 -> 40 : 4.9
0 -> 41 : 6.4
0 -> 42 : 8.5
0 -> 43 : 5.1
0 -> 44 : 2.2
0 -> 45 : 2
0 -> 46 : 8.8
0 -> 47 : 1.5
0 -> 48 : 5.7
0 -> 49 : 3.7
Average shortest path from Node 0 is 6.598
```