**Programming Project for Data Structure Design and Algorithm Implementation**

**Environment used for Successful Generation of Output:**

OS: Windows 7

Language: Python

Interpreter version: Python 2.7.9

Download Location: <https://www.python.org/ftp/python/2.7.9/python-2.7.9.msi>

**Installation Instructions:**

1. Download the python interpreter from above location and install it.

2. Set the compiler path in the environmental variables in the control panel to use Python from the command prompt window.

Help link: <https://docs.python.org/2/using/cmdline.html#environment-variables>

3. Extract the source code files in the submitted zip file to any folder of your preference. Graph.py is the only source code file that is required to run this program.

4. Make sure the folder has the source code graph.py and the input txt file which contains the vertices to be added to the graph. (the sample network.txt is given in the submission)

5. Open command prompt and navigate to the particular folder where the source file is located

6. Now run the program with the following command:

**python** **graph.py <filename>**

**e.g.: python graph.py network.txt**

7. The queries can then be given in standard input as desired and the output can be verified in the next line after Enter is hit

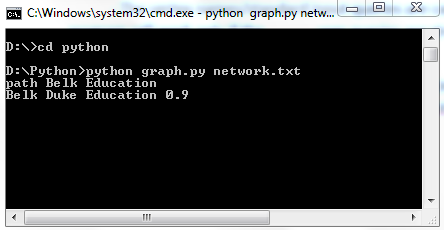
8. Following are the scenarios where the program either asks for valid user input or will throw an exception and terminate the program.

- The input file doesn’t exist in the given path

- The vertex names/standard input queries are case sensitive. So the program will work only if the user queries are given as in the sample queries file provided.

- Any invalid vertex names while providing the “path source\_vertex dest vertex“ query will provide valid output only if both source\_vertex and dest\_vertex are present in the graph.

Sample run of the program in command prompt



**Data Structure Design and Algorithm Implementation:**

Class **Vertex:**

This class holds the vertex information including name, UP/DOWN status, previous vertex and transmission time from the source vertex (used for Dijkstra's algorithm).

Class **Edge:**

This class holds the edge details including start vertex, end vertex, transmission time of an edge, UP/DOWN status of edge.

Class **Graph:**

A class to store the list of vertices and edges. Adjacency list of each vertex is maintained. All the user queries/ operations on the graph are performed in this class which include:

1. addedge

2. deleteedge

3. edgedown

4. edgeup

5. vertexdown

6. vertexup

7. path

8. print – Uses Dijkstra’s algorithm to compute the path. A priority queue is implemented on a min binary heap which makes sure the computation runs in O((V+E) log V)

9. reachable – Uses Breadth first traversal for each vertex that is UP via UP edges

So the computation time is O( V(V+E)) for all Vertices V that are “UP” and Edges E that are “UP” if and only if they are between two vertices that are “UP” .

Class **PriorityQueue:**

This class implements a Priority Queue with a min binary heap. This ensures the running time of Dijkstra's algorithm is O((V+E)Log V). The ShiftUp() and ShiftDown() methods perform the heapify operations. The ExtractMin() removes the minimum element which is usually present as the first element of the heap and can be extracted in O(1) time. Also the heapify operation is performed to maintain the heap order property. The DecreaseKey() method is used to place the element in proper position in the heap after finding the element in the position of the array. This runs in O(log n) time.

**Reading input files:**

The input file is read in Main function. The standard input is also implemented in main function.