Task 0: building the graph

** I have used the skeleton code for Graph, Vertex and Edge class from lecture coding in my implementation.

I have made a Graph class that is implemented in all the task below. This class will read in vertex and edges from 2 different input file(let's say vertices file and edges file). During reading in of values from the vertices file, I will add in the responding vertex into my graph (an array with the number of vertices read from the input file). After finish adding in the vertices, I will read in the edges from the edges file and add them to my corresponding vertices. I would calculate and add in the weight of each edge when I insert them into the graph. In my Graph class, I also have a reset_graph() function to reset the graph so that it does not confuse the next operation that will be done using the same graph.

I have made a Vertex class that will instantiate Vertex object that will be used in the Graph class. I also made a few functions to get the data/values of the vertices. Other than that, I have an Edge class that would instantiate Edge object that is used in Graph class.

The function runs in O(V+E), as I need to add V number of vertices and E number of edges into the Graph.

Task 1: Solve ladder

** I got the idea of the BFS algorithm from lecture codes.

After I have built a Graph with the files given, I will implement Breadth First Search Algorithm to find all the shortest distance from source vertex to all the vertex in the graph. After that, I will backtrack from the target vertex towards the source (by choosing the shortest path that was calculated in the previous BFS) to get my path if a path can be found, or else the function will return False.

Each vertex and edge is traverse once, therefore the complexity is O(V+E)

Task 2 : Cheapest ladder

**coded the MinHeap class by referring to the lecture codes back for FIT1008

The main idea is to use Dijkstra twice to get the path from source vertex to target vertex by passing through a vertex that has the required character. This means that I break down the path into 2 parts : source to specific vertex, specific vertex to target vertex. If I was unable to find either one, my search is considered as failed.

Thus, I might only find a valid path if my graph has a vertex that has the required character. So, if I could not find a vertex from my graph it means that there is no valid path.

In order to find a path that traverse through a vertex with specific requirements, I will first do Dijkstra by using the start vertex as my source and find all the shortest path from my source to all the other vertex in the graph. Then, I will find the closest vertex that meet the requirement and return the path of source -> that vertex.

After that, I will do the second Dijkstra using the vertex I found in the first Dijkstra as source and find the shortest path from that vertex to the all the other vertex in the graph. I will select the path that leads to the target vertex if any is found. If a complete path is found, I will return the path.

In this task, I have implemented Dijkstra with priority queue (by implementing minheap), therefore the complexity is O(ElogV)