Report

I ran this code using Python in Visual Studio code with no errors. I was having difficulty running the code initially due to directory/pathing problems, but worked around the issue with the line "os.chdir(r"C:\Users\28epi\Documents\ITCS 6114 Project 2_Ethan Pinto")". This line should be commented out or edited with the correct path to run.

Problem 1:

Briefly, the code works through the following steps:

- Create a Graph class with methods to add edges and perform Dijkstra's algorithm
- Read from the input file the details about the graph (whether it's directed or undirected, edges, weights)
- Set a source node
- Initialize all distances as INFINITY and distance of source as Zero
- Repeat until all vertices are included in the shortest path tree:
- Pick a vertex *u* which is not there in the shortest path tree and has a minimum distance value.
- Include *u* to the shortest path tree.
- Update distance value of all adjacent vertices of *u*.
- Print the shortest paths from source to all nodes

The time complexity of the algorithm is $O(V^2)$ where V is the number of vertices in a given graph. We come to this value due to the outer *for* loop that runs *V* times, where for each vortex, the program searches through all other vertices to find the minimum distance vertex.

The algorithm works for both directed and undirected graphs. For a directed graph, we only add the edge from the *from* node to the *to* node with the given weight. Additionally, for an undirected graph, we also add an edge in the opposite direction with the same weight. The implementation of this algorithm handles multiple text files, reading graph data from each file and printing the shortest path from the source node to all other nodes in the graph. The program also provides the time taken by the algorithm as required.

Problem 2:

Briefly, the code works through the following steps:

- Define a Graph class with methods to add edges and implement Kruskal's algorithm.
- Read the input file and retrieve graph details (edges and weights).
- Create a list to store the graph edges and weights.
- Sort the edges in non-decreasing order of their weight.
- Initialize a new list to store the minimum spanning tree.
- For each sorted edge, process it in order and do the following:
- If adding the edge doesn't form a cycle, add it to the MST list.
- If adding the edge forms a cycle, discard it.
- Continue this process until there are V - 1 edges in the MST list.
- Print the Minimum Spanning Tree and its total weight.

The primary operations in Kruskal's Algorithm are sorting the edges and performing the union-find operations. Sorting the edges takes O(E *log* E) time, where E is the number of edges. Each union-find operation takes nearly constant time, so performing these operations for all edges results in a time complexity of O(E *log* E). The MST is stored as a list of tuples, where each tuple represents an edge in the MST and its weight.

This implementation of Kruskal's algorithm works for both directed and undirected graphs, but for this case, it is being used on undirected graphs because Kruskal's algorithm typically applies to undirected graphs. The algorithm handles multiple text files, reading graph data from each file and printing the minimum spanning tree (MST) and the total weight (or cost) of the MST.