Muhammad Usman Memon **Student ID: 6010878**

CSCI-203 - Assignment 3

Overall Solution: The Program is solved in Python Programming Language.

The program reads the input file line by line. Since the number of Vertices and Edges is on the first line it saves it in a list and then for the next lines it uses the data from the first line to read the vertices and edges between them. A *2-D matrix of Vertices x Vertices* named **adjMatrix** is created with **0** values. This is a nested list (2d list) to store information for the edges between vertices in the form of an *Adjacency matrix*. The code reads through the vertices and stores the data in a nested array for every vertex the **x,y coordinates** are stored in an array **vertices**. The edges are read and then stored in **adjMatrix** accordingly. Then the last line is read and split and we save the 2 values as start and end point.

Since the file has been read and all the data has been saved in a useful form. The function **Euclidian** calculates the distance between the start and end vertices that we saved from the file. The function uses the x,y coordinated from **vertices** array. Then the Euclidian distance is worked out and returned.

The class **Graph** is used to solve the adjacency Matrix and find the shortest and second shortest paths and distance from start to end. We use the *Dijkstra's Algorithm* since we are not expecting any negative edges between vertices. In the class, we have 4 functions **minDistance(), savePath(), Dijkstra() and secondShortestPath().**

The function Dijkstra is the main function which takes in the adjMatrix, start and an empty array step to store the path as an input. An array **dist** of length equal to number of vertices is created and the values in the array are initialized to maximum (since we store the minimum distance). Another array/list **parent** is used which hold the data for the parent of the vertex respectively according to their array position, -1 would mean it is the root. Create a list as **queue** which acts as a queue and hold the vertices that are to be visited. Once queue is filled with all vertices we loop through it until it is not empty. Then we pick the vertex which has smallest value in **dist** array and use its index value to take that vertex out from the queue. In the first iteration it is the start point since **dist[start] = 0**. After picking out the vertex we loop through all the Vertices in the chosen vertex's adjacency matrix and check if there is an edge between the 2 vertices and if vertice is still in queue and its current distance in the dist array is less than or more than the new distance being calculated. If it is less we update otherwise continue iteration and add the vertice to the parent list of that index/vertex. The savePath loops through the parent until we hit the parent and adds the parent vertices to the path.

The function second shortest is the secondary function which calculates the second shortest path for our Adjacency Matrix or Graph. It takes the original path of the shortest distance and loops through the path in such a way that it edits the graph by removing the edges between the two vertices. It starts with the first 2 vertices in the path and continues till the last 2 vertices. Saves the result for vertice pair in an array and returns the result. The function makes a copy of the original path so that during the function it does not alter it. The **gandu** is a list of list to store paths of different set of vertices and abc is list to store different shortest distances of vertices pairs. Prevy stores info for the previous edge. PreS and PreD store info for the previous vertices. S and d read vertices pairs from the original shortest path array. Then function loops through the length of the shortest path and alters the edges for the vertices in the shortest path and tries to find the shortest path by running the altered graph through the Dijkstra function without that edge and stores it in the abc array. This function took inspiration from the help provided by the lecturer in the assignment specification.

Finally, the results are printed. For the second shortest distance we loop through the results from the secondShortest function and find the index and value of the minimum distance from the array and print it out along with its index.

The math library was used to calculate the square root in the Euclidian function since it was difficult to figure out hard coding the square root so used the python math calculator for that.

Standard Algorithms used:

To search for the minimum values inside the lists we used linear search. Since the length of lists/arrays is not large we use for loop to iterate through the loop and find minimum values from list in the program and output our desired results. Linear search is simple and does require much computation for small lists so it was the best algorithm for this scenario. Other algorithms for search could be used but that would require to sort the lists first then search and since the aim of this assignment was to test knowledge on searching graphs, I decided to keep it simple.

Recursion was used in the function savePath to recurse through the parent list until the base case of -1 was hit. Hitting the base case meant that the root/start of the path had been reached, until then it would keep on calling itself on the parent list.

Data Structures Used:

Nested Lists: Nested lists or 2-D arrays are very useful here for storing lots of information including storing the Adjacency Matrix, to storing the data for the shortest paths on the secondShortest path for each edge removal. The array vertices is also a nested list/ array that hold information of x,y coordinates for the vertices.

Lists: From storing the distance values of all the vertices, to storing parent info for all vertices, to temporary storing data from the line read from the file and to soring the queue of arrays left behind. List are used where the data is in varying length and depends on the number of vertices or would vary on the situation of data.

Output Produced :

The program has a time complexity of O (n^2) since we are looping through a 2-D array of AdjMatrix for the Dijkstra Algorithm.

```
In [218]: runfile('C:/Users/intern.usman/Desktop/CSCI-203/a3/mum123_a3.py',
Users/intern.usman/Desktop/CSCI-203/a3')

Please enter the name of the file along with extension: a3-sample.txt
The filename you entered is a3-sample.txt
The number of Vertices in the Graph is 20 and edges is 100
Finding the shortest path between vertices 2 and 13

The Euclidian Distance is :  77.87810988975015
The shortest Path is :  2 13
Shortest Path length is  85
The second Shortest distance for the Start and End vertices is:
Distance : 134
Path :   2 17 20 13

In [219]:
```