Mahinour Abdelgawad- 900231870

Marina Nazeh - 900233146

Mohammed Elsayed - 900231065

Habiba Khalil - 900232380

Mohamed Bashady – 900233394

**Mini Wasalny Application**

*Project Documentation*

**Purpose:**

This application aims to solve the problem of navigating from one city to another. When a user has a map of cities and only knows the direct paths from one city to another, or would like to navigate between cities that are not directly connected, this program computes the shortest path between any two cities.

**Functionalities:**

1. Add a city: This feature allows the user to add and save a city to a map
2. Add a path: This feature allows the user to add a path between two certain cities, with the distance in kilometers.
3. Update a path: This feature allows the user to modify the distance of an existing path between two cities, or completely delete the path.
4. Delete a city: This feature allows the user to delete a city from the map, which automatically deletes all of its associated paths
5. Display the map: This feature shows the user the map with all the cities they had saved, along with all of the direct paths
6. Navigate: This feature allows the user to select the two cities they would like to navigate between, and then displays the shortest possible distance between the two cities, along with all the paths and cities they need to cross in order to reach their destination.
7. Save the map: After the user is done, the program automatically saves the user’s changes when they quit the program. The user’s progress is restored when they start the program again.

**Data Structures:**

1. The graph: The graph is a user-defined class with an adjacency list that is an unordered map whose key is the name of the city, and the value is a vector representing all the edges (paths) connected to the city. The edges are represented as a pair to store both the name of the connected city as well as the weight of the edge (the distance). An unordered map in this situation is the most suitable data structure as it does not allow duplicates, allows quick access due to the constant time (O1), and is space efficient as it only stores necessary information. Vectors are the most suitable for storing edges as they are resizable and provide easy addition and removal of edges. Using pairs with the vectors is necessary as this is a weighted graph, so both the name of the city and the distance need to be stored.
2. Dijkstra’s algorithm: A queue was used to implement the shortest-path algorithm. This was necessary in order to explore the cities using breadth-first search, which makes use of the First-In-First-Out characteristic of queues. An unordered map was used to store and update the shortest-known distances, the reason for this choice is the same reason we chose an unordered map for the graph itself. Additionally, another unordered map was used to store the previous city of a path during the execution of the function. This map was later on used with a stack to generate the cities in the shortest path. A stack was used here for its Last-In-First-Out characteristic to maintain the order of the cities in the path.

**Technicalities:**

1. AddCity: This is a void function in the Graph class which takes a string as a parameter (name of a city) and inserts it into the adjacency list with an empty vector as its value.
2. AddEdge: This is a void function in the Graph class which takes two strings (two cities) and a float (distance between them) as parameters. For each of the cities, it adds the other city and the distance to their respective vectors of edges.
3. DeleteCity: This is a void function in the Graph class which takes a string as a parameter (name of a city) and erases it from the adjacency list. It then removes its edge from all the other cities this one was connected to.
4. FindShortestDis: This is a function in the Graph class which takes two strings (two cities) as a parameter and it returns a pair formed from a vector of strings, which stores the cities in the shortest path, and a float, which represents the shortest distance. It uses breadth-first search to compute the shortest distance between any two cities.
5. findPath: This is a private function in the Graph class which is a helper for FindShortestDis. It takes in an unordered map of the previous cities which was created in the shortest distance function, and two strings which represent the source and destination. It makes use of a stack to form the path in order, and returns a vector of strings, which the original function then returns. This was written as a separate function for simplicity.
6. UpdateGraph: This is a void function in Graph class that takes two strings (two cities) and a float (new distance between them) and changes the edges between the cities to the new distance (updates).
7. removeEdge: This is a void function in Graph class that takes two strings (two cities) and completely removes the edge between them
8. DisplayGraphData: This is a void function in Graph class that takes no parameters. It prints out each city in the graph and all the cities they are each connected to, along with the distances.
9. loadGraph: This is a void function in Graph class that takes a string as a parameter, representing the name of the text file which stores the graph. It reads the file and then initializes the graph using the data in the file
10. WriteToFile: This is a void function in Graph class that takes a string as a parameter, representing the name of the text file which stores the graph. It overwrites the file with the data in the graph after running the program.
11. cityExists: This is a boolean function in Graph class that takes a string (city name) as a parameter. It returns true if the city exists in the graph’s adjacency list and false otherwise. This function is not directly accessible by the user, but it is used by other functions in the main and inside the class
12. edgeExists: This is an overloaded boolean function in Graph class that takes two strings or two strings and a float. When a float is passed, it returns whether an edge of the same distance exists between two cities. This is used in functions inside the class. The second version returns whether an edge exists between the two cities, regardless of the distance. This is used in the main and functions inside the class
13. count: This is an int function that takes no parameters and returns the number of cities in the graph.
14. stringTest: This is a bool function in the main that takes a string as a parameter and returns whether it is a valid city name or not (only letters, no spaces, etc..) It is used in the input validation in main

\*All user input is validated before performing any of the graph functions