**Jaypee Institute of Information Technology**

****

**DATA STRUCTURES LAB PROJECT**

**SUBMITTED TO: Dr. Raju Pal**

**SUBMITTED BY –**

Lakshit Bakshi- (9921103102)

Harsh Khamankar- (9921103109)

Sanat Bhatia- (9921103119)

Kanishk Singhal-(9921103144)

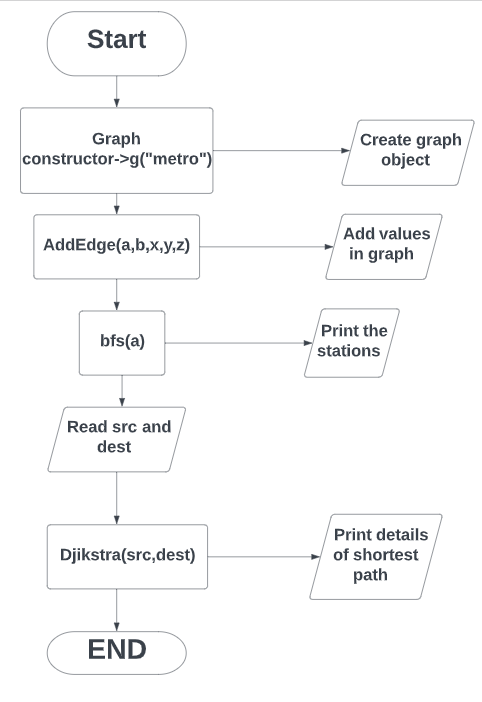
|  |  |
| --- | --- |
| **Index** | **Page No.** |
| 1. Title Page | 1 |
| 1. Index | 2 |
| 1. Problem Statement | 3 |
| 1. Solution Approach & Flowchart | 3 |
| 1. Data Structures used and Why | 4 |
| 1. Implementation and Results | 5 |
| 1. Utility of Project and Complexity Analysis | 6 |
| 1. Acknowledgements | 7 |
| 1. References | 8 |

**Problem Statement:**

A metro network consists of many stations and various routes. Its difficult for the consumer to decide which route is cost-effective as well as time-saving. For help, a person may need to go to the helpdesk or ask another person making the process tiring and troublesome; and more resource needy for the metro corporation.

**Solution Approach & Flowchart:**

To solve the above problem, we have implemented a C++ program based on graphs and other concepts. The consumer can easily get help from it, by entering the source and destination stations, after which the program will display the details of the shortest path. The individual can effortlessly interact with the program and there will be no need of extra personnel.



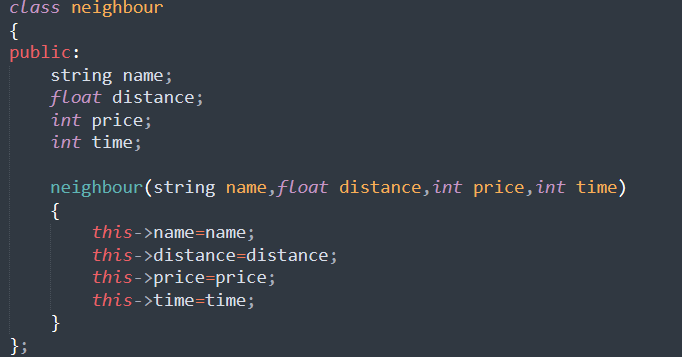
**Data Structures Used and Why:**

* For this project, we have used the adjacency list representation of graph, in which we have used Map and List; and a self-made class:- neighbour, which acts as a data packet.
* The map’s key is of type string i.e. name of the station and that of the value is list of neighbours.



* In the map, for each station its list of neighbours is stored(the stations which are adjacent), hence called adjacent list representation.

Our data packet, neighbour has member variables as below:



* If we access neighbour list of any station, let’s say ‘A’ from the neighbour map and take any neighbour from it say ‘B’ then the distance, price and time of that object of neighbour class is the distance, price and time taken to go from A to B.

Other data structures used are Set, Unordered\_set, pair, queue.

* Set is used in the Dijkstra algorithm for sorting the ordered pairs of distance(from a source station),name and accessing the smallest valued pair(acc. to distance) and updating it in the algo.
* Unordered\_set is set in which the values are not sorted. It is used in Dijkstra as well as the Breadth First Search (BFS). In Dijkstra , the edges which are relaxed are added to unordered\_set(explored) and while exploring nodes, they are checked for their presence in the explored u\_set so that they are not processed again.

In BFS, the unordered\_set is used for keeping track of the visited nodes i.e. the ones that are printed.

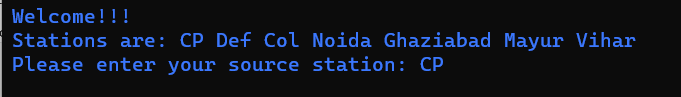
* Pair is used for taking out values and putting values when dealing with map, as well as when dealing with pair of distance and station name in set and unordered\_sets.
* Queue is used in the BFS algorithm for keeping track of the unexplored(not printed vertices of the graph).

**Implementation and Results**

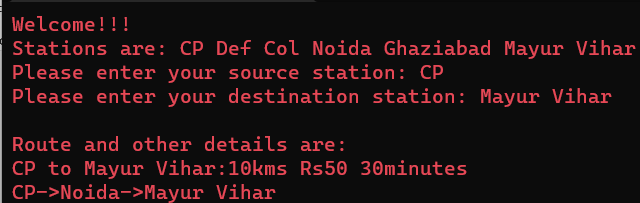
User is greeted and then the stations are displayed.

****

Now, is asked for the source station.

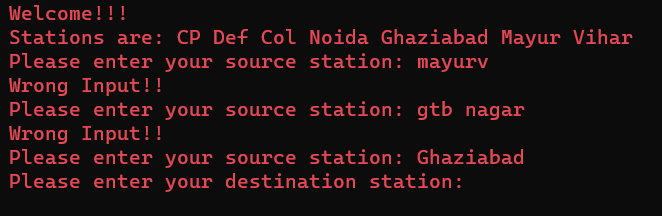
****

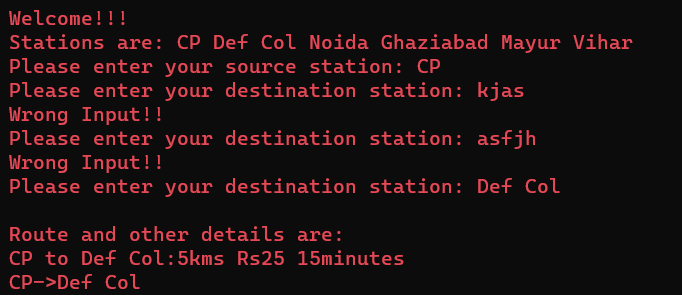
Now, destination station and then the shortest route details are displayed using the Dijkstra Algorithm on the metro network Graph.

****

Exception Handling:

If the user inputs a value which is not part of the network, appropriate message is displayed and re-entry of the name is asked for.





**Utility of Project and Complexity Analysis**

The above program can be incorporated by any metro corporation for easy, efficient and fast calculation of the best route possible between two station; for the convenience of its consumers. Similar approach can also be used in creating a navigation system, or delivery chain management helper.

It can also come in aid in networking, especially social networking because graph is used by social network giants and the Dijkstra algorithm will prove trustworthy in the transfer of data and information across the network.

* The addEdge() function in the graph class takes constant time i.e. it is of order O(1)
* PrintPath() is of order O(v), where v is the number of vertices in the graph; when the graph is a straight line and the source station is the first whereas the destination station is the last.
* The time complexity of Dijkstra() is O(V+ElogV+ElogV+V), estimated to O(V+ElogV), where V is the number of vertices and E is the number of edges in the graph.
* In bfs() for each vertex, its adjacency list is accessed and its neighbours are added to the queue while the current vertex is removed. Complexity is

V\*(O(1)+O(E’)+O(1)

O(V)+O(V\*E’)+O(V)

2V+E

O(V+E)

E’ is number of adjacent edges, E is total edges, V is number of vertices.

**Acknowledgements**

We would like to express our special thanks of gratitude to our professor Dr. Raju Pal for their able guidance and support in completing our project using graphs, which motivated us to research and know more about the topic and learn new things alongside.

We would also like to thank our parents in providing us with all the necessary

facilities which were required to complete our project.

**References:**

* <https://stackoverflow.com>
* <https://devdocs.io/cpp/>
* <https://cplusplus.com/doc/>
* <https://www.geeksforgeeks.org/>
* <https://www.javatpoint.com/>
* <https://www.hackerearth.com/>
* <https://www.freecodecamp.org/>
* <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>
* <https://www.log2base2.com/data-structures/graph/adjacency-list-representation-of-graph.html>
* <https://www.tutorialspoint.com/>
* <https://www.digitalocean.com/community/tutorials/compare-strings-in-c-plus-plus>