IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

METRO ASSISTANT FOR DMRC

[1]Nishant Kumar, [2]Ms. Nidhi Sengar
[1]B.tech I.T, [2]Assistant Professor I.T.
[1]Department of Information Technology,
[1]Maharaja Agrasen Institute Of Technology, Delhi, India

Abstract: The DMRC has divided its journey into lines which connect with each other at exchanges i.e stations common between two lines the trains running on one line are exclusive to that line so if a person needs to go from a station on one line to a station on some other line, they have to change their line in the middle of journey.

These changes make it difficult to navigate the metro but it is necessary as all trains can't go to each station .with Such a vast Network of stations and so many trains and almost 10-11 lines on top of that, it is extremely difficult for people to figure out the most efficient path for their journey from station A to station B. Although Prices will stay the same no matter what path you take the time taken to travel can be reduced drastically if we choose our exchanges properly.

This tool will function as an assistant to the user providing them with the fastest and shortest path to get from one station to another. The tool will also calculate the price of the journey. The user gives their origin and destination stations as an input and the tool finds the path for them and tells them where to change between lanes how many stops to ride in a lane to get to their destination.

***Keywords* - Graph Algorithm, Dijkstra algorithm, A* algorithm, shortest path.**

I. Introduction

Introduction to DMRC

The DMRC has divided its journey into lines which connect with each other at exchanges i.e stations common between two lines the trains running on one line are exclusive to that line so if a person needs to go from a station on one line to a station on some other line, they have to change their line in the middle of journey. These changes make it difficult to navigate the metro but it is necessary as all trains can't go to each station .with Such a vast Network of stations and so many trains and almost 10-11 lines on top of that, it is extremely difficult for people to figure out the most efficient path for their journey from station A to station B. Although Prices will stay the same no matter what path you take the time taken to travel can be reduced drastically if we choose our exchanges properly. This tool will function as an assistant to the user providing them with the fastest and shortest path to get from one station to another. The tool will also calculate the price of the journey. The user gives their origin and destination stations as an input and the tool finds the path for them and tells them where to change between lanes how many stops to ride in a lane to get to their destination.

Our tool will give the user multiple paths with differing time taken by each path and it will be ordered according to how many tasks it takes but user still has a choice to select a path with a larger distance due to some other work they may have a in the id journey.

1.2Basic Terminology

1.2.1 *C++*
1.2.2 STL Standard Template Library

1.2 Motivation & Problem Statement

The motivation behind this project is to facilitate the journey common man in crowded city like Delhi and understanding the complexities of its vast subway like public transportation network.

As an outsider I had to once face enormous challenges understanding the DMRC network and I had to literally sit down with its maps every time I wanted to go somewhere. DMRC is a very aspirational project but given how it has to be fitted in an already small, crowded city it has its limitations and this tool will be helping hand to make people enjoy the journey rather than focus on figuring out the complexities of the Map of DMRC.
These changes make it difficult to navigate the metro but it is necessary as all trains cant go to each station with Such a vast Network of stations and so many trains and almost 10-11 lines on top of that, it is extremely difficult for people to figure out the most efficient path for their journey from station A to station B. Although Prices will stay the same no matter what path you take the time taken to travel can be reduced drastically if we choose our exchanges properly.

II. LITERATURE REVIEW

A metro rail-based system was prescribed by RITES, containing a network of surface, underground and raised corridors accumulating to 213 kms, to stay aware of the traffic demand up to the year 2021. The entire task expense was assessed at Rs. 15000 Crores according to 1996 value level.
In this part let's review the basic principles which were used to break the Problem into various parts to solve it.

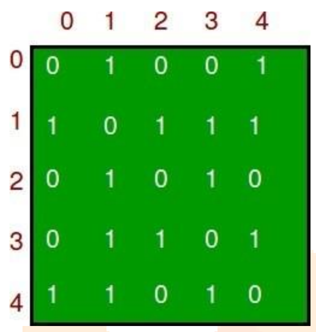
2.1 Graphs

A graph can be defined as group of vertices and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent child relationship.

A graph G can be defined as an ordered set G (V, E) where V (G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices.

adjacency Matrix In graph theory and computer science, an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal. If the graph is undirected (i.e. all of its edges are bidirectional), the adjacency matrix is symmetric. The relationship between a graph and the eigenvalues and eigenvectors of its adjacency matrix is studied in spectral graph theory.
The adjacency matrix of a graph should be distinguished from its incidence matrix, a different matrix representation whose elements indicate whether vertex–edge pairs are incident or not, and its degree matrix, which contains information about the degree of each vertex.

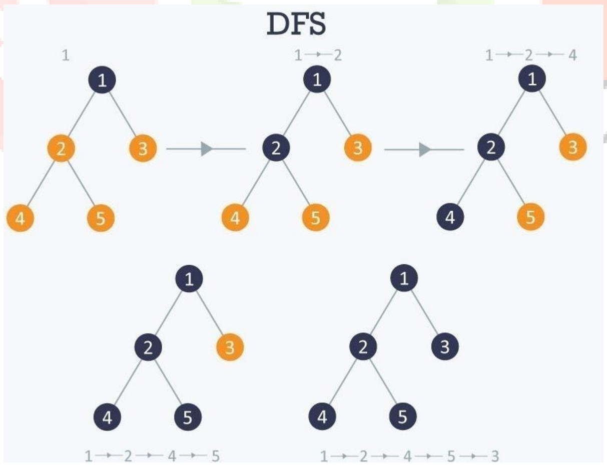


	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

This is used to represent the actual connections in the metro lines instead of using Graph to represent the entire map we made a graph only for the connection of lines.A graph representation is a technique to store graph into the memory of computer.

2.2DFS (Depth First Search)

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, a node may be visited twice. To avoid processing a node more than once, use a boolean visited array.



This approach by us to find link between the lines. Using an improved version of DFS called IDDFS

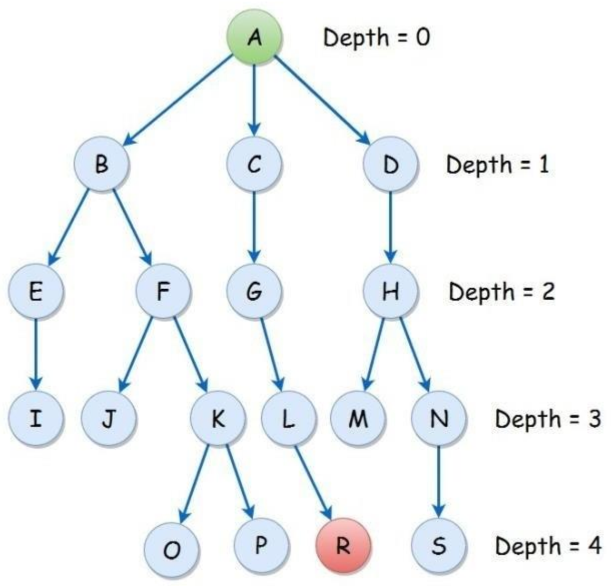
Why use IDDFs

- In DFS, you would recursively look at a node's adjacent vertex. DFS may not end in an infinite search space. Also, DFS may not find the shortest path to the goal. DFS needs O(d) space, where d is depth of search.

BFS consumes too much memory. BFS needs to store all the elements in the same level. Inthe case of a tree, the last level has N / 2 leaf nodes, the second last level has N / 4. So, BFS needs O(N) space.

Iterative deepening depth first search (IDDFS) is a hybrid of BFS and DFS. In IDDFS, we perform DFS up to a certain "limited depth," and keep increasing this "limited depth" after every iteration.

Let us take an example to understand this –



DEPTH	DLS traversal
0	A
1	A B C D
2	A B E F C G D H
3	A B E I F J K C G L D H M N
4	A B E I F J K O P C G L R D H M N S

This approach helps us find the resulting connections with minimum line changes. That is important as it helps us save time wasted on changing lines and gives us data back already sorted according to the no of different lines which is used to find the connection. Which helps us minimize the resultant path length and time.

2.3PATH FINDING ALGORITHMS

Blind search algorithms such as BFS and DFS all possibilities; starting from the given node, they iterate over all possible paths until they reach the goal node.

Path finding or pathing is the plotting, by a computer application, of the shortest route between two points. It is a more practical variant on solving mazes. This field of research is based heavily on Dijkstra's algorithm for finding the shortest path on a weighted graph..

A algorithm*

A* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

Informally speaking, A* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms. This fact is cleared in detail in below sections.
And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).

Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A* Search Algorithm comes to the rescue.

What A* Search Algorithm does is that at each step it picks the node according to a value-'f' which is a parameter equal to the sum of two other parameters – 'g' and 'h'. At each step it picks the node/cell having the lowest 'f', and process that node/cell.

III. METHADOLOGY

What does the project do?

This tool will function as an assistant to the user providing them with the fastest and shortest path to get from one station to another.

The tool will also calculate the price of the journey.

The user gives their origin and destination stations as an input and the tool finds the path for them and tells them where to change between lanes how many stops to ride in a lane to get to their destination.

3.1 OVERVIEW OF STEPS INVOLVED IN PRODUCING THE RESULT

- Gathering and brining the data into the codebase:
  The gathering of information such as names of stations, names of lines and connection between the lines and storing them into text files.
- Reading the Data into workable graphs and HashMap's which can used to define the metro connections and network. The code for this is attached in the below process.
- Getting the inputs in starting and ending stations.
- Finding the lines associated to the stations given by the user.
- Finding connection between those lines using the maps and graphs generated in the previous steps.
- Finding the most optimal path using the stations no to calculate the distance in the fashion of A* algorithm as it is used both the cost of reaching a given point and cost of the future journey into consideration.
- Representing the output as a set of line changes and stations at which it happens to reach from source to destination.

3.2 DATA GATHERING AND EXPLANATION OF THE GATHERED DATA

We collected the following data items:

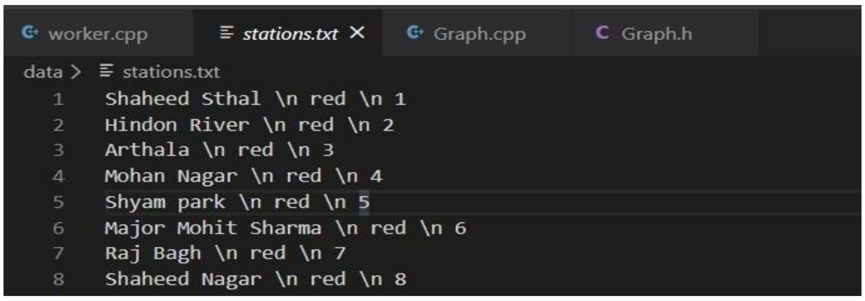
1. Names of all the stations.
2. Lines to which a particular station belongs to.
3. The station no that station has on the given line.

Name of files:

1. Stations.txt

This file contains the name of stations along with its line and station no.

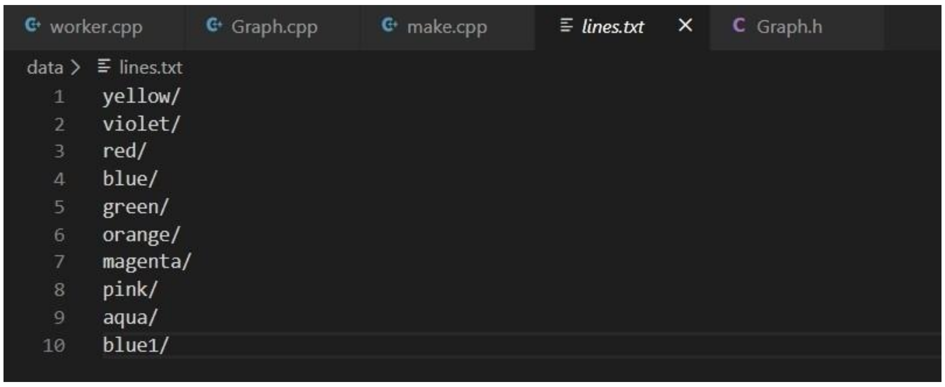
Format: Station name \n Line name \n station no



```
Shaheed Sthal \n red \n 1
Hindon River \n red \n 2
Arthala \n red \n 3
Mohan Nagar \n red \n 4
Shyam park \n red \n 5
Major Mohit Sharma \n red \n 6
Raj Bagh \n red \n 7
Shaheed Nagar \n red \n 8
```

*2. **Lines.txt***

This file contains the names of all the distinct lines in the metro. Format: Line name



```
yellow/
violet/
red/
blue/
green/
orange/
magenta/
pink/
aqua/
blue1/
```

This file is read to get all the line names into the code. With the following code piece.

```
void load_lines(vector <string> &lines)
    { string line;
    ifstream
    myfile("data/example.txt");
    if (myfile.is_open())
    {
        while (getline(myfile, line))
        {
            string
            s="";
            for(int
                   i=0;i<line.length();
                   i++){ i
                   f(line[i]=='/')
                    bre
                    ak;
                    s+=li
                    ne[
                    i];
            }
            lines.push_back(s);
        }
        myfile.close();
        return;
    }
    else cout  << " Unable to open file";
```

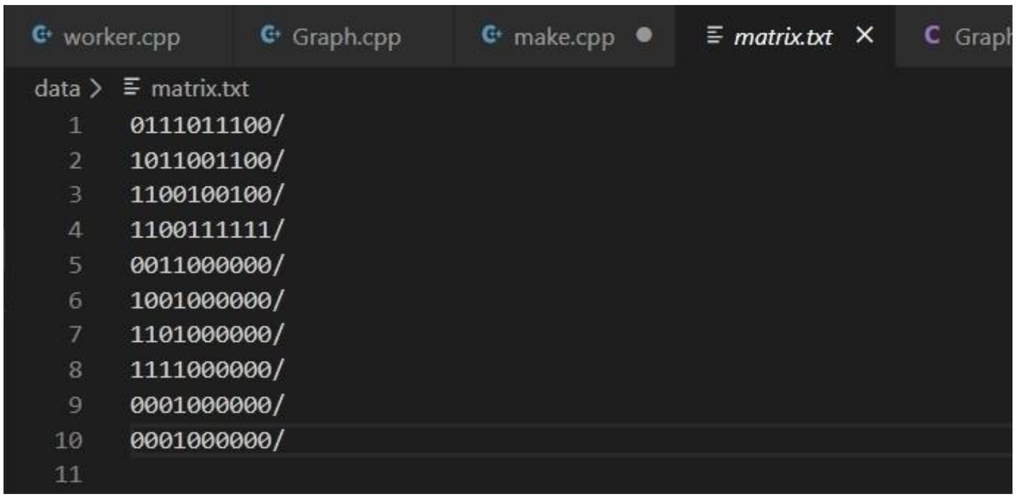
From here we get an array with names of lines as elements.

3. *Matrix.txt*

This file is used to store the adjacency matrix into the codebase to make a graph of all lines connection. This is one of the most important parts of the code as it helps us in the primary steps involved in the path finding algorithm.

Format: N*N matrix to show all the row no x represents the line no x and column j represents thelines no j in the array of lines we made above. Followed by the

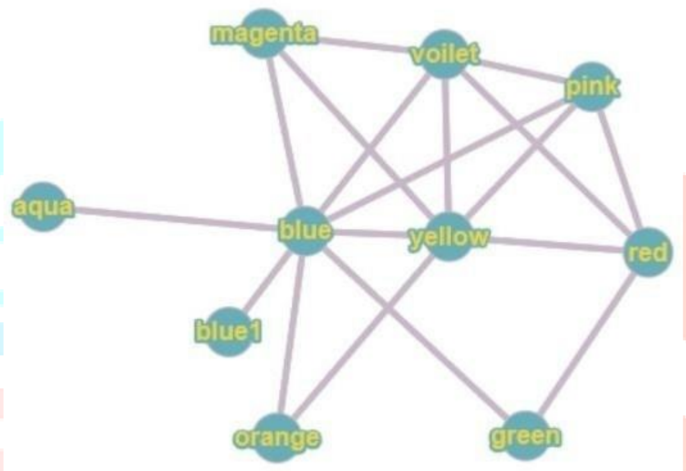
Format: N*N matrix to show all the row no x represents the line no x and column j represents thelines no j in the array of lines we made above. Followed by the



```
0111011100/
1011001100/
1100100100/
1100111111/
0011000000/
1001000000/
1101000000/
1111000000/
0001000000/
0001000000/
```

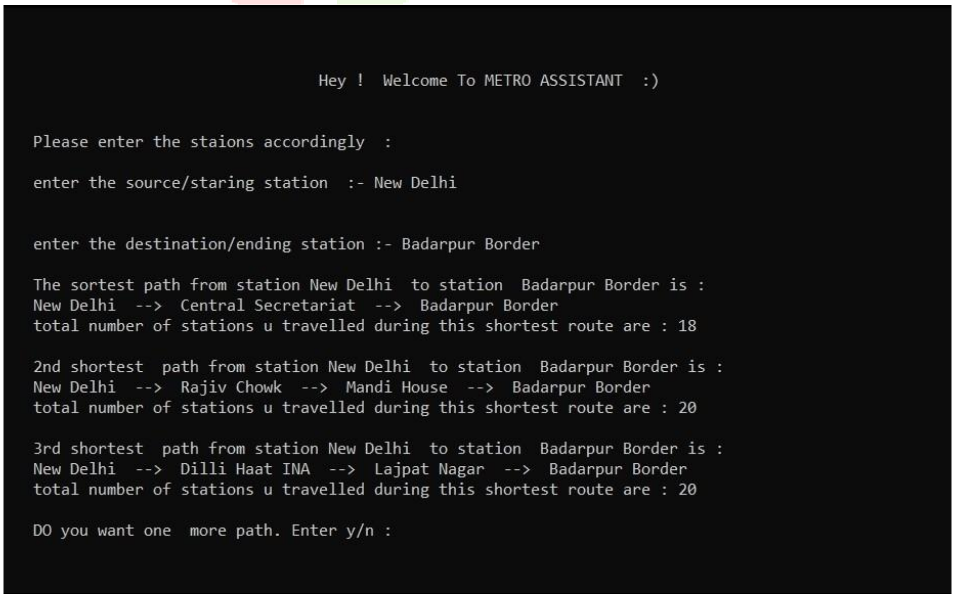
This matrix is used to represent the graph of all the lines. This graph is in other modules like HashMap to figure the optimal path.

The graph is as follows:



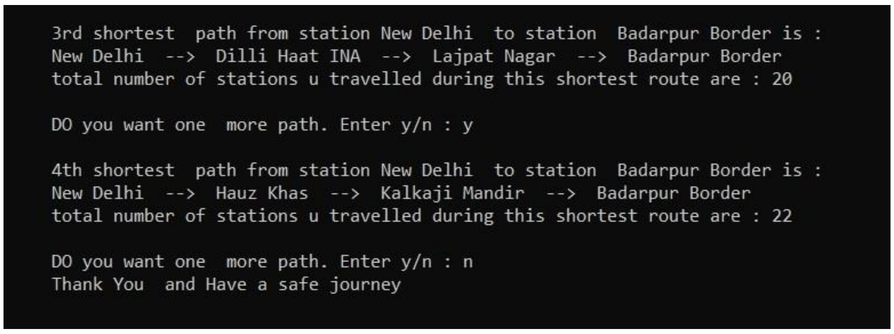
IV.RESULT

The Result shows all the paths with the most optimal path at the top. It contains the source station followed by exchange stations for where we need to change lines and then it gets to the destination station. Also gives the no of station You travel overall to reach your destination if you take the path.



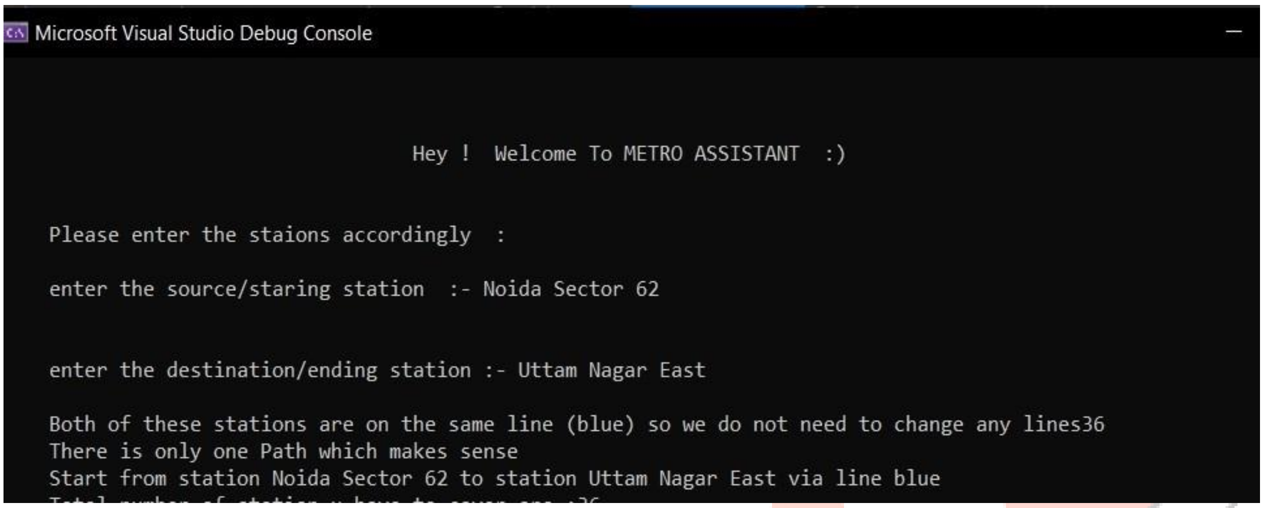
```
                    Hey !  Welcome To METRO ASSISTANT  :)

Please enter the staions accordingly  :

enter the source/staring station  :- New Delhi


enter the destination/ending station :- Badarpur Border

The sortest path from station New Delhi  to station  Badarpur Border is :
New Delhi  -->  Central Secretariat  -->  Badarpur Border
total number of stations u travelled during this shortest route are : 18

2nd shortest  path from station New Delhi  to station  Badarpur Border is :
New Delhi  -->  Rajiv Chowk  -->  Mandi House  -->  Badarpur Border
total number of stations u travelled during this shortest route are : 20

3rd shortest  path from station New Delhi  to station  Badarpur Border is :
New Delhi  -->  Dilli Haat INA  -->  Lajpat Nagar  -->  Badarpur Border
total number of stations u travelled during this shortest route are : 20

DO you want one  more path. Enter y/n :
```

The output first shows 3 most prominent paths and asks if we want to see more path say yes it shows more paths if no it ends.



```
3rd shortest  path from station New Delhi  to station  Badarpur Border is :
New Delhi  -->  Dilli Haat INA  -->  Lajpat Nagar  -->  Badarpur Border
total number of stations u travelled during this shortest route are : 20

DO you want one  more path. Enter y/n : y

4th shortest  path from station New Delhi  to station  Badarpur Border is :
New Delhi  -->  Hauz Khas  -->  Kalkaji Mandir  -->  Badarpur Border
total number of stations u travelled during this shortest route are : 22

DO you want one  more path. Enter y/n : n
Thank You  and Have a safe journey
```

If the both source and destination are on the same line it shows us only one path as it doesn't make sense to go back change lines just to reach a station on the same line.



```
                    Hey !  Welcome To METRO ASSISTANT  :)


Please enter the staions accordingly  :

enter the source/staring station  :- Noida Sector 62


enter the destination/ending station :- Uttam Nagar East

Both of these stations are on the same line (blue) so we do not need to change any lines36
There is only one Path which makes sense
Start from station Noida Sector 62 to station Uttam Nagar East via line blue
```

V.CONCLUSION

The project was highly successful in finding the Paths between stations provided to it and it can find the solution in a very small time.

Thus, it can be practical tool to navigate within the DMRC.

We tested the project at each step and phase of development with manually entering various inputs and testing their output against the most optimal path that we could find using the map and it can out accurately almost every time.

so, we can say that this is an accurate system which works quickly and efficiently to give its user the most optimal path from the DMRC on being given the input.

VI. REFERENCES

1. To get data we used: https://www.mapsofindia.com/maps/delhi/delhi-metro-map.html
2.
3. To understand the Working and project making in C++ we used:
4. Geeks for Geeks: https://www.geeksforgeeks.org/
5. Used this site a lot to understand how and when and why to use which algorithm and which data structure. Medium:https://medium.com/swlh/pathfinding-algorithms-6c0d4febe8fd and Various other articles.
6. Hacker Rank: https://www.hackerrank.com/
7. to understand implementation of key concepts through Problem solving The channel on YouTube: https://www.youtube.com/playlist?list=PLlrATfBNZ98dudnM48yfGUldqGD0S4FFb This playlist Can teach you a lot about C++
8.
9. Graph making Tools: https://graphonline.ru/en/create_graph_by_matrix
10. Understanding Algorithms:
11. https://medium.com/kredo-ai-engineering/search-algorithms-part-3-uninformed- search-algorithms- 2-1bc85c0a3900
12. Stack Overflow https://stackoverflow.com/
13. To communicate with professionals who help me understand the errors in my code