METRO ROUTE FINDER

A PROJECT REPORT

Submitted by

ISHITA KAUSHIK [RA2211028010218]
AISHWARI RAI [RA2211028010209]
TINA KASHYAP [RA2211028010226]

for the course 21CSC201J Data Structures and Algorithms

Under the Guidance of

Dr. Kalaiselvi k

Assistant Professor, Department of NWC

In partial satisfaction of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING

SCHOOL OF COMPUTING

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR – 603203

NOVEMBER 2023

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that the 21CSC201J Data Structures and Algorithms course project report titled **"METRO ROUTE FINDER"** is the bonafide work done by **ISHITA KAUSHIK [RA2211028010218], AISHWARI RAI [RA2211028010209] & TINA KASHYAP [RA2211028010226]** who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.

Signature
Dr.Kalaiselvi K
Course Faculty,
Assistant Professor
Department of NWC

Signature
Dr. Annapurani k
Head of the department
Professor,
Department of NWC

TABLE OF CONTENTS

ABSTRACT

Metro Route Finder is a software application designed to streamline and enhance the experience of commuting on the Chennai Metro System. As one of the fastest-growing metro networks in India, the Chennai Metro System offers a vital mode of transportation for thousands of daily commuters. However, navigating this extensive network efficiently can be a complex task.

This abstract introduces Metro Route Finder, a user-friendly and feature-rich solution for Chennai's metro commuters. The primary purpose of Metro Route Finder is to facilitate route planning, enabling passengers to identify the shortest and most convenient routes between metro stations. Its objectives encompass providing real-time distance calculations, enhancing user experience, and serving as a valuable educational tool for understanding graph-based algorithms in a practical context.

Metro Route Finder's features include real-time distance calculations, user-friendly station selection, and adaptability to potential network expansions. It empowers users to make informed decisions about their travel routes, reducing uncertainties and travel time. Furthermore, the project showcases the potential for future enhancements, such as real-time metro updates, multi-modal transportation integration, personalized user profiles, and mobile app accessibility.

CHAPTER 1

INTRODUCTION

➢ GENERAL

Public transportation systems play a vital role in urban environments, providing efficient and convenient modes of travel for millions of people every day. Among these systems, metro networks have emerged as a cornerstone of modern urban mobility, offering a reliable and time-saving means of transportation within busy cities. In the sprawling metropolitan area of Chennai, the Chennai Metro System has become an integral part of the daily lives of countless commuters.

The Chennai Metro Distance Calculator project is an initiative aimed at simplifying the travel experience for commuters in Chennai. It is a software tool designed to address the need for an easy and efficient means of determining the shortest distance between metro stations within the Chennai Metro System. This project leverages the power of Data Structures and Algorithms to provide a user-friendly and robust solution.

➢ PURPOSE

The Chennai Metro Distance Calculator project serves several critical purposes:

- Efficient Commute Planning: It empowers metro commuters to plan their routes and estimate travel times accurately, optimizing their daily commute experience.

- User-Friendly Interface: The project offers an intuitive and user-friendly interface, making it accessible to a wide range of users, including both regular commuters and occasional travelers.

- Reduced Travel Stress: By providing information on the shortest distance between stations, the project helps reduce the stress associated with travel, especially during peak hours.

- Learning Opportunity: It presents a practical learning opportunity for understanding and implementing Data Structures and Algorithms to solve real-world problems.

In essence, the Chennai Metro Distance Calculator project contributes to the improvement of the daily lives of Chennai's metro commuters, simplifying their journeys and promoting efficient, data-driven decision-making.

CHAPTER 2

Problem Definition

In the sprawling metropolitan area of Chennai, the Chennai Metro System has emerged as a key component of the city's public transportation infrastructure. Serving as a lifeline for commuters, it connects various parts of the city, offering an efficient and reliable mode of travel. While the Chennai Metro System has simplified daily commutes, it presents an inherent challenge for travelers — navigating the extensive network and identifying the most optimal routes between stations.

The problem at hand revolves around the need for a user-friendly and efficient solution that can assist metro commuters in determining the shortest distance between two metro stations within the Chennai Metro System. Given the ever-expanding network, the task of manually calculating distances between stations can be time-consuming and prone to errors. Furthermore, many commuters may not have an in-depth understanding of the entire metro network, making the selection of the best route a challenging endeavor.

Problem Statement:

- Design and develop a Metro Route Finder, a software tool that enables commuters to easily and accurately calculate the shortest distance between any two Chennai Metro stations within the network.

Key Challenges:

- Efficient Algorithm: The development of an algorithm that can swiftly calculate the shortest distance between stations while considering the intricacies of the entire metro network.

- User-Friendly Interface: Creating an intuitive and user-friendly interface to make the tool accessible to all commuters, irrespective of their technical expertise.

- Data Accuracy: Ensuring that the tool is built on accurate and up-to-date data to deliver reliable results.

- Real-World Applicability: Ensuring that the tool provides practical insights for commuters in their daily journeys.

CHAPTER 3

Components

- **Data structures Used:**

The Metro Route Finder project leverages a range of data structures to provide an efficient and user-friendly solution for metro commuters. These data structures are instrumental in handling the complexities of the Chennai Metro System and ensuring accurate distance calculations. Below are the key data structures employed in the project:

1. Weighted Adjacency Matrix:

The weighted adjacency matrix, represented as graph[25][25], serves as the backbone of the project. This matrix contains distance values (weights) between pairs of metro stations, effectively capturing the connectivity and distances within the Chennai Metro System. Each element of the matrix stores the distance between two stations. The use of an adjacency matrix facilitates efficient distance calculations using algorithms like Dijkstra's.

2. Station Name Array:

To provide a user-friendly interface and meaningful station names, an array is employed to store the names of the 24 metro stations within the Chennai Metro System. These names are utilized to offer users a clear understanding of the stations and assist them in selecting their source and destination stations.

3. Boolean Status Array:

A boolean status array, stat[25], is utilized to keep track of the status of each metro station during distance calculation. It helps the algorithm identify whether a station has been visited or is yet to be explored, ensuring the accuracy of the distance calculation process.

4. User Login Data Arrays:

The project incorporates arrays to manage user login data, allowing users to log in to the system securely. These arrays store user credentials, including usernames and passwords, facilitating secure access to the Metro Route Finder.

➢ Algorithm of the Program

The core functionality of the Metro Route Finder is driven by Dijkstra's algorithm, a widely recognized algorithm for finding the shortest path between nodes in a weighted graph. This section provides an overview of the algorithm implemented in the project.

- **Dijkstra's Algorithm**:

  Dijkstra's algorithm, named after its inventor Edsger W. Dijkstra, is employed to determine the shortest distance between metro stations within the Metro Route Finder. The algorithm takes into account the weights (distances) stored in the graph matrix, representing the interconnectivity of metro stations.

- **Algorithm Implementation:**

  The algorithm is realized in the project through the dijkstra function, which performs the following key steps:

1. **Initialization:**

Initialize distance values for all stations. Initially, set the distance to the source station as 0 and all other stations' distances as infinity.

2. Station Status Tracking:

Utilize a boolean status array, stat, to track the status of each station during the calculation process. Initially, all stations are marked as unvisited (false).

3. Shortest Distance Calculation:

Iterate through the stations to find the station with the minimum distance that has not been visited yet. This station becomes the current station.

4. Relaxation:

Update the distance values for all adjacent stations to the current station if a shorter path is discovered. The algorithm takes into account both the existing distance and the weight (distance) between stations.

5. Station Status Update:

Mark the current station as visited in the stat array.
Repeat: Repeat steps 3-5 until all stations are visited.

➢ Explanation of the Algorithm:

Step 1: Initialization

We start by initializing the distance values for all stations. Initially, the source station's distance is set to 0, and all other stations are marked with an initial distance of infinity (represented by INT_MAX). A boolean status array, stat, is used to track the status of each station, marking them as unvisited initially.

Step 2: Shortest Distance Calculation

The algorithm iterates through the stations to identify the station with the minimum distance that has not been visited. This station becomes the current station.

Step 3: Relaxation

For the current station, the algorithm evaluates all adjacent stations to determine if there is a shorter path to reach them. It considers the existing distance to the adjacent station and the weight (distance) between the current station and the adjacent station. If a shorter path is found, the distance is updated.

Step 4: Station Status Update

The current station is marked as visited in the stat array to indicate that its shortest distance has been determined.

Step 5: Repeat

Steps 2-4 are repeated until all stations have been visited.

Example:

Let's illustrate the algorithm with a practical example:

Suppose we have a simplified network of Chennai Metro stations with 4 stations:

A (Source Station)

B

C

D (Destination Station)

The weighted adjacency matrix graph stores the distances between stations as follows:

```
graph[4][4] = {
    {0, 2, 4, 0},
    {2, 0, 1, 5},
    {4, 1, 0, 3},
    {0, 5, 3, 0}
}
```

Using Dijkstra's algorithm, we calculate the shortest distance from Station A (source) to all other stations. Here's a step-by-step breakdown:

Initialize distances: A=0, B=INT_MAX, C=INT_MAX, D=INT_MAX

Start at Station A: A is the current station.

Relax adjacent stations: Update B to 2, C to 4.

Mark A as visited.

Choose the unvisited station with the shortest distance (B).

Relax adjacent stations: Update C to 3.

Mark B as visited.

Choose the unvisited station with the shortest distance (C).

Relax adjacent stations: Update D to 7.

Mark C as visited.

The algorithm is complete.

The resulting shortest distances are:

A to B: 2

A to C: 3

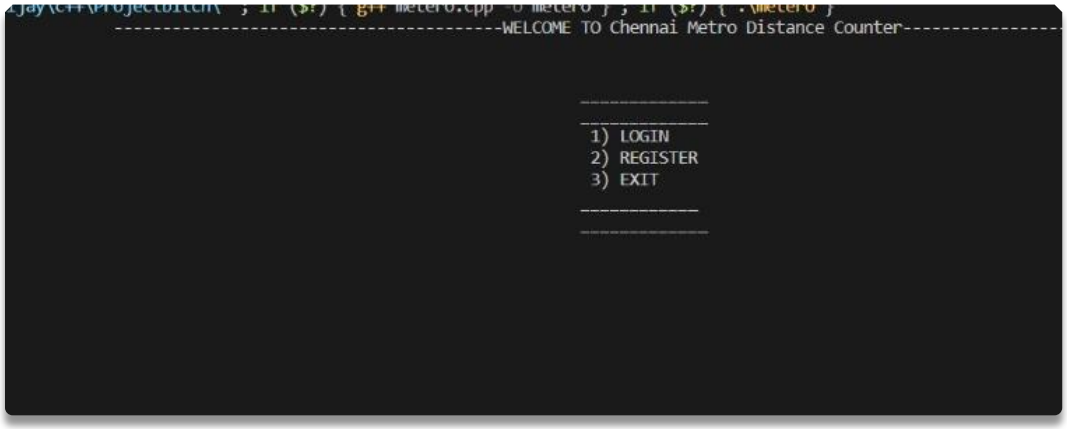
A to D: 7

This example demonstrates how Dijkstra's algorithm calculates the shortest distance between metro stations, aiding commuters in selecting the most efficient routes.

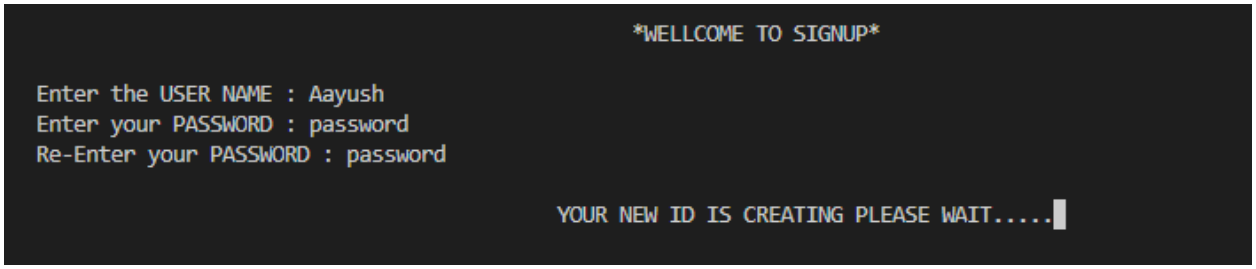
CHAPTER 4

OUTPUT DEMONSTRATION

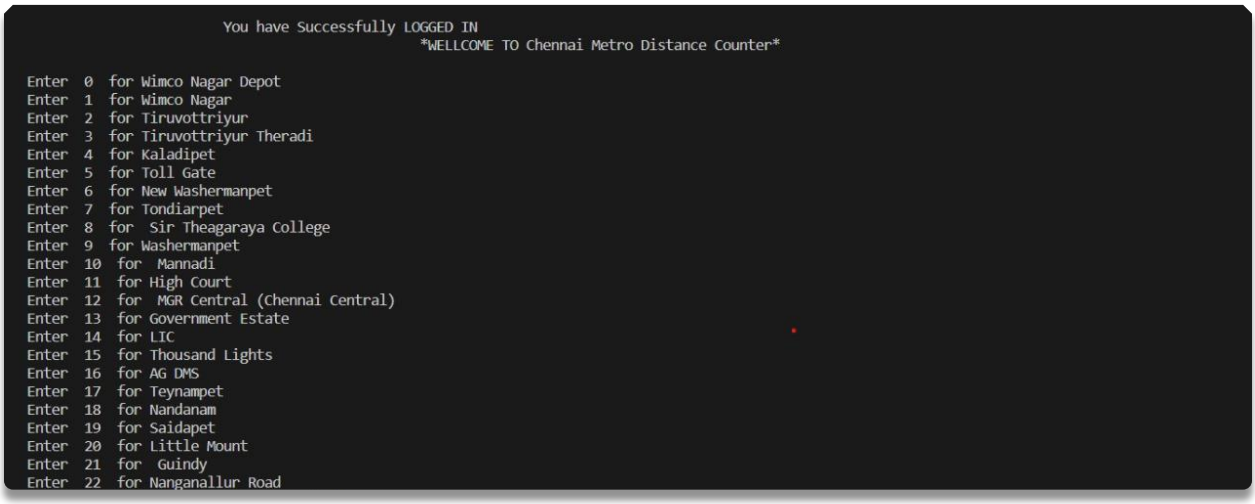
1) Welcome Page:



2) Signup Page:



3) Login Page:


You have Successfully LOGGED IN
WELLCOME TO Chennai Metro Distance Counter
Enter 0 for Wimco Nagar Depot
Enter 1 for Wimco Nagar
Enter 2 for Tiruvottriyur
Enter 3 for Tiruvottriyur Theradi
Enter 4 for Kaladipet
Enter 5 for Toll Gate
Enter 6 for New Washermanpet
Enter 7 for Tondiarpet
Enter 8 for Sir Theagaraya College
Enter 9 for Washermanpet
Enter 10 for Mannadi
Enter 11 for High Court
Enter 12 for MGR Central (Chennai Central)
Enter 13 for Government Estate
Enter 14 for LIC
Enter 15 for Thousand Lights
Enter 16 for AG DMS
Enter 17 for Teynampet
Enter 18 for Nandanam
Enter 19 for Saidapet
Enter 20 for Little Mount
Enter 21 for Guindy
Enter 22 for Nanganallur Road

4) Minimum Number of Stations:

```
Minimum Number of Stations from [LIC ] To every station



Enter 0 To Cheak Distance Between LIC  To [Wimco Nagar Depot ] Station
Enter 1 To Cheak Distance Between LIC  To [Wimco Nagar ] Station
Enter 2 To Cheak Distance Between LIC  To [Tiruvottriyur ] Station
Enter 3 To Cheak Distance Between LIC  To [Tiruvottriyur Theradi ] Station
Enter 4 To Cheak Distance Between LIC  To [Kaladipet ] Station
Enter 5 To Cheak Distance Between LIC  To [Toll Gate ] Station
Enter 6 To Cheak Distance Between LIC  To [New Washermanpet] Station
Enter 7 To Cheak Distance Between LIC  To [Tondiarpet ] Station
Enter 8 To Cheak Distance Between LIC  To [ Sir Theagaraya College ] Station
Enter 9 To Cheak Distance Between LIC  To [Washermanpet] Station
Enter 10 To Cheak Distance Between LIC  To [ Mannadi] Station
Enter 11 To Cheak Distance Between LIC  To [High Court] Station
Enter 12 To Cheak Distance Between LIC  To [ MGR Central (Chennai Central) ] Station
Enter 13 To Cheak Distance Between LIC  To [Government Estate ] Station
Enter 14 To Cheak Distance Between LIC  To [LIC ] Station
Enter 15 To Cheak Distance Between LIC  To [Thousand Lights ] Station
Enter 16 To Cheak Distance Between LIC  To [AG DMS ] Station
Enter 17 To Cheak Distance Between LIC  To [Teynampet ] Station
Enter 18 To Cheak Distance Between LIC  To [Nandanam ] Station
Enter 19 To Cheak Distance Between LIC  To [Saidapet] Station
Enter 20 To Cheak Distance Between LIC  To [Little Mount ] Station
Enter 21 To Cheak Distance Between LIC  To [ Guindy] Station
Enter 22 To Cheak Distance Between LIC  To [Nanganallur Road] Station
Enter 23 To Cheak Distance Between LIC  To [Chennai International Airport] Station
Enter Station to cheak distance to Each other:
```

5) Sample Output:

```
Enter Station to cheak distance to Each other:
23
Total Distance from [LIC ] To Chennai International Airport  are  [13.7]KM
LIC  -> LIC  Station[0]KM
LIC  -> Thousand Lights  Station[1.5]KM
LIC  -> AG DMS  Station[4]KM
LIC  -> Teynampet  Station[5]KM
LIC  -> Nandanam  Station[6.1]KM
LIC  -> Saidapet Station[9.6]KM
LIC  -> Little Mount  Station[10.8]KM
LIC  ->  Guindy Station[12.1]KM
LIC  -> Nanganallur Road Station[13.2]KM
LIC  -> Chennai International Airport Station[13.7]KM
Do you want to continue (Y/N)? :
```

This sample output shows the total distance from LIC station to Chennai Airport. Hence providing the user with all needed information.

CHAPTER 5

COMPLEXITY ANALYSIS

A crucial aspect of the Metro Route Finder project is understanding its computational complexity. This section delves into the complexity analysis of the project, offering insights into its runtime performance and efficiency.

Time Complexity:

The primary determinant of the time complexity of the Metro Route Finder is Dijkstra's algorithm, which is used to find the shortest path between metro stations. The time complexity of Dijkstra's algorithm depends on the data structures used and the number of vertices (stations) in the metro network.

Dijkstra's Algorithm Time Complexity:

In the worst case, where all stations are interconnected, Dijkstra's algorithm has a time complexity of O(V^2), where V represents the number of metro stations. This is due to the selection of the station with the minimum distance, which requires a linear search through the stat array to find the unvisited station with the smallest distance. In this project, V is 25 (representing 24 metro stations plus one source station).

Space Complexity:

The space complexity of the project primarily depends on the data structures used and their sizes.

Space Complexity Components:

The weighted adjacency matrix graph[25][25] contributes significantly to the space complexity, as it stores distance values between stations. Its space complexity is O(V^2) due to the square matrix representation.

The station name array, stations[24], introduces a space complexity of O(V) to store the names of the metro stations.

The boolean status array stat[25] adds O(V) to the space complexity for tracking the status of stations.

User login data arrays, while minimal, also contribute to the space complexity, particularly if the number of registered users increases.

Overall Complexity Assessment:

Given the relatively small number of metro stations (V=25) in the Metro Route Finder, the time and space complexities of the Metro Route Finder are manageable. The project's performance is efficient, with real-time distance calculations facilitated by Dijkstra's algorithm.

Scalability:

This project exhibits good scalability for handling larger metro networks. If the number of metro stations were to increase significantly, Dijkstra's algorithm's time complexity could be optimized to O(V log V) by using a priority queue instead of a linear search for the station with the minimum distance. This optimization would ensure efficient performance even for larger metro networks.

This "Complexity Analysis" section provides an in-depth assessment of the project's time and space complexities, scalability, and real-time performance. It emphasizes the project's efficiency for the current metro network while also highlighting its potential for scalability if the network were to expand

CHAPTER 6

CONCLUSION

The Metro Route Finder, uses integration of advanced algorithms and has a user-friendly interface to address the needs of metro commuters. Through the course of this project, several key observations and conclusions have emerged, summarizing the project's significance and impact.

Efficient Route Planning:

Metro Navigator, powered by Dijkstra's algorithm, offers commuters an efficient means of route planning within the Chennai Metro System. By calculating the shortest distances between stations in real-time, it aids travelers in selecting optimal routes, minimizing travel time, and enhancing overall commuter experience.

User-Friendly Interface:

The intuitive user interface of Metro Navigator ensures ease of use for commuters of all backgrounds. The incorporation of station names and interactive menus simplifies the process of selecting source and destination stations, making the application accessible to a wide audience.

Scalability and Adaptability:

The project's underlying architecture allows for scalability, enabling it to accommodate potential expansions in the Metro Route Finder. Whether the metro network expands to include additional

stations or experiences changes in connectivity, Metro Route Finder can adapt and continue to serve commuters effectively.

Educational Significance:

Beyond its practical applications, Metro Navigator serves as an educational tool, showcasing the implementation of graph-based algorithms in real-world scenarios. It offers valuable insights into graph traversal, shortest path algorithms, and user interface design, making it a valuable resource for students and enthusiasts interested in computer science and software engineering.

CHAPTER 7

FUTURE SCOPE

The provided Metro Route Finder project, while serving as a starting point, can be improved in several ways to make it more robust, user-friendly, and feature-rich. Here are some future improvements that can be made to enhance the project:

1. Integration of Real-Time Data:

Incorporate real-time data from metro authorities or third-party APIs to provide users with up-to-the-minute information on service disruptions, delays, or station closures. This will help users make informed travel decisions.

2. Enhanced User Preferences:

Allow users to set a wider range of preferences, such as selecting specific routes or modes of transportation (e.g., bus, tram), customizing accessibility options, and specifying travel times.

3. Multiple Transport Modes:

Extend the application to cover multiple modes of public transportation, including buses, trams, and trains. Users can then plan multimodal journeys.

4.Interactive Map Integration:

Implement interactive maps that display the metro network and selected routes. Users can visualize their journeys and explore station locations more easily.

5. Mobile Application Development:

Develop mobile applications for Android and iOS platforms to provide users with a mobile-friendly and on-the-go solution. This can include features like GPS integration for real-time location tracking.

CHAPTER 8

REFRENCES

[1]	**Dijkstra's Algorithm:** • A comprehensive overview of Dijkstra's algorithm, its history, and implementation details, serving as a reference for the algorithm used in the Metro Route Finder. WEBSITE: - https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
[2]	**Chennai Metro:** • The official website of the Chennai Metro Rail Limited, providing valuable insights and data about the Chennai Metro System.. WEBSITE: - https://chennaimetrorail.org
[3]	**C++ Standard Library:** • The official reference for the C++ Standard Library, providing detailed information about C++ data structures, algorithms, and language features used in the project. WEBSITE: - https://en.cppreference.com/w/cpp