METRO RAIL APPLICATION

A PROJECT REPORT

Submitted by

MOHAMMAD HASSAN [Reg No: RA2212704010030]

VAIBHAV SINGH [Reg No: RA2212704010040]

SIMRAN [Reg No: RA2212704010034]

Under the Guidance of

Dr. TAMIZHSELVAN CHIDAMBARAM
(Assistant Professor, Department of Data Science and Business Systems)

In partial fulfillment of the Requirements for the Degree of

M.TECH (Integrated)
COMPUTER SCIENCE WITH SPECIALIZATION IN DATA SCIENCE

INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

DEPARTMENT OF DATA SCIENCE AND BUSINESS SYSTEMS

FACULTY OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

BONAFIDE CERTIFICATE

Certified that this project report titled **METRO RAIL APP** is the bonafide work of **“MOHAMMAD HASSAN [Reg No: RA2212704010030]**, **VAIBHAV SINGH [Reg No: RA2212704010040] SIMRAN [Reg No: RA2212704010034]** “who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation based on which a degree or award was conferred on an earlier occasion for this or any other candidate."

Dr. TAMIZHSELVAN CHIDAMBARAM	Dr. M.Lakshmi
Assistant Professor	**Head of Department**
Dept. of DSBS	Dept. of DSBS

Signature of Internal Examiner   Signature of External Examiner

ABSTRACT

The Metro Rail App, meticulously crafted through the integration of sophisticated data structures and algorithms, represents a cutting-edge solution designed to optimize and elevate the metro rail commuting experience. Leveraging advanced graph algorithms, the app efficiently computes and recommends the shortest and most time-effective routes, ensuring users reach their destinations with minimal transit times. Real-time data structures, such as priority queues and dynamic arrays, power the app's live tracking feature, providing users with instantaneous updates on train locations, platform changes, and arrival times. Additionally, crowd density monitoring employs innovative data structures to assess and display congestion levels, enabling users to make informed decisions about the timing of their commute. The app's ticketing system utilizes secure data structures to manage digital transactions seamlessly, enhancing the user experience with integrated, cashless payments. Through a combination of hash tables and personalized user profiles, the app tailors recommendations based on commuting history, preferences, and saved routes. The integration of diverse data structures and algorithms not only fortifies the app's functionality but also positions it as an intelligent and dynamic tool in the realm of urban transportation, ensuring a seamless, efficient, and personalized metro rail journey for users.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In the ever-evolving landscape of urban transportation, the Metro Rail App emerges as a groundbreaking solution, meticulously engineered through the seamless integration of diverse data structures and algorithms. As urban centers grapple with the complexities of efficient public transit systems, this app represents a paradigm shift in the way commuters navigate metro rail networks. At its core, the application leverages advanced graph algorithms to compute optimal routes, ensuring users reach their destinations with unprecedented efficiency. Real-time tracking, powered by dynamic data structures, provides users with instantaneous updates, transforming the conventional commuting experience. Moreover, the app's ability to monitor crowd density through innovative algorithms enables users to make informed decisions about the timing and comfort of their journeys. With a robust ticketing system and personalized user profiles driven by secure data structures, the Metro Rail App not only streamlines the ticketing process but also tailors recommendations based on individual preferences and commuting history. This introduction sets the stage for a transformative urban commuting experience, where technology, in the form of advanced algorithms and data structures, takes center stage to redefine efficiency, convenience, and personalization in metro rail travel.

1.1 CONTRIBUTION

- The Metro Rail App, integrating advanced algorithms and data structures, optimizes urban commuting. It computes efficient routes, offers real-time tracking, and monitors crowd density. With streamlined ticketing and personalized profiles, the app revolutionizes the metro rail experience, enhancing efficiency, safety, and user satisfaction in urban transit systems.

- Additionally, the app's intelligent use of data structures and algorithms enables predictive maintenance, minimizing service disruptions. This proactive approach enhances the reliability and availability of metro rail services, contributing to a more resilient and dependable urban transportation infrastructure.

CHAPTER 2

PROBLEM STATEMENT

2.1 MOTIVATION

In modern cities, efficient and stress-free urban commuting is essential for the well-being and productivity of their residents. However, existing metro rail systems often face several challenges that hinder a seamless commuting experience. These challenges include:

• User Input: The app should accept user-provided information, including the name of the source station and the destination station. • Data Retrieval: Retrieve and maintain comprehensive data about the metro network, station connections, distances, fare structures, and real-time updates.

• Graph Representation: Create a graph representation of the metro rail network to facilitate efficient route planning and fare calculation.

• Shortest Path Algorithm: Implement a shortest path algorithm (e.g., Dijkstra's) to find the quickest route from the source to the destination, taking into account various factors such as distance, transfers, and real-time data.

• Fare Calculation: Calculate the fare for the selected route, considering factors like distance, zones, and fare rules.

• Display Information: Display the shortest metro route, fare, and any other relevant data to the user in a user-friendly format.

• Metro Map: Provide a user-accessible metro map for visual reference, aiding commuters in understanding the network layout.

2.2 OVERVIEW

- **Graph Representation:** - The app utilizes a graph-based representation for modeling metro rail stations and their connections.
- **Functionality:** - Provides essential functionalities such as displaying all metro stations, showing the metro map, and finding the shortest distance and time

between two stations

- **Shortest Path Algorithms: -** Implements Dijkstra's algorithm to find the shortest distance and time between two metro stations.

- **Fare Calculation: -** Provides functionality to calculate fare based on either the distance traveled or the time taken for the journey. Users can choose the fare calculation method based on their preferences.

- **Interchange Information: -** Offers information on interchange stations, including the number of interchanges and the stations involved. Enhances user experience by providing comprehensive details on the route.

2.3 BACKGROUND TO THE STUDY:

The study aims to create a user-centric app addressing urbanization challenges with technology-driven solutions for a more efficient and enjoyable public transportation experience.

- **User Experience Enhancement:-** The app streamlines commuting with accurate route and timing information.

- **Optimizing Commuting Routes:-** Algorithms like Dijkstra's algorithm compute optimal routes, minimizing travel time.

- **Fare Calculation and Integration: -** Integration with digital payments provides a seamless, cost-effective commuting experience.

- **Predictive Maintenance:-** Predictive maintenance minimizes service disruptions, enhancing reliability.

- **Interchange Information:-** Detailed interchange information optimizes transfer points for users.

- **Map Visualization:-** Visualizing metro maps improves accessibility for journey planning.

- **Safety and Crowd Management:-** Real-time data aids crowd monitoring, ensuring safety and efficient service utilization.

2.4 PROBLEM STATEMENT

Urban centers face escalating traffic congestion due to rapid urbanization, necessitating efficient public transportation. The current challenge lies in providing a seamless and user-friendly metro rail experience. This study addresses the need for an advanced Metro Rail App, employing cutting-edge algorithms and data structures to optimize route planning, enhance user experience, integrate digital payments, and ensure real-time information for improved urban mobility. The goal is to mitigate congestion, promote sustainable commuting, and offer a technologically-driven solution to urban transportation challenges.

2.5 DATA STRUCTURE USED

Data structure is a way of organizing data in the computer so that it can be used effectively. The data structures used in this project are:

• Array
• HashMap
• Linked List
• Heap
• Stack
• Graph
• The array data structure is implemented using ArrayList class which is a resizable array. It stores data in contiguous memory locations.
• HashMap data structure is used to store key-value pairs.
• The stack data structure required by the algorithm is implemented using linked list. It follows the principle Last In First Out(LIFO).
• Heap is implemented with the help of ArrayList and HashMap. It is a tree-based data structure where all elements of the tree are in a particular order.
• Graph is a complex data structure containing vertices and edges connecting these vertices. This is implemented in the project with the help of ArrayList and HashMap.

CHAPTER 3
LITERATURE SURVEY

1. **ONLINE METRO RAILWAY -TICKETING WITH DATA ANALYTICS by Vijaykumar M Bhat , Vismaya S Rao ,Tousif khan Soudagar , Vaishnavi N, Kavyashree S:** The goal of this system's development is to create an Android application for metro train management. The entire system is more secure, efficient, and time efficient. This project is highly useful in everyday life and can be implemented or worked on quickly. The operation would be totally automated, efficient, upgraded, and cost-effective in the planned online metro ticketing system. The proposed method can be used in a variety of settings, including tollgates, bus ticketing, and so on.
2. **Smart Metro Rail Ticketing System by Abhishek Nair M, Smit Taunk, Panyam Gangadhar Reddy, Parveen Sultana H:** Using this system we eradicated the hassles of day to day traveling. There is now no need to carry physical tickets/tokens or any other UID card/documents for the sake of traveling. With this proposed methodology, the user will be ensured a more comfortable and convenient travel experience.
3. **Analysis of Metro rail Project selection Bias with Principal-Agent Model by Vinod Rathod:** Using this study they provided an overview on Metro rail Operation Performance Indicators, Operation, System Management, Cost Efficiency, Service Supply, Quality of Service

4. **THE METRO PROJECT FINAL REPORT by Ingason, Kumm M., Nilsson, D., Lönnermark A., Claesson, A., Li, Y. Z., Fridolf, K., Åkerstedt, R., Nyman, H., Dittmer, T:** Fires in trains or individual carriages have been found to be larger than earlier expected (77 MW in the full scale tests). If the carriage doors are open or not were found to have a large significance for the fire development as well as the type of interior wall and ceiling material in the carriage. The “luggage factor” is not commonly used when defining the fire load but the project show it should. It was shown that the luggage play an important role in the initial stages of the fire development.

5. **END USER IMPACT OF METRO RAIL SERVICES by Aprajita Verma, Arabindo Majhi, Gandhamalla Abraham Noel, Geetha Krishna Saraswatula:** A case study on the Hyderabad Metro Rail which plays a significant role in the large-scale urbanization process in the country that intends for urban spaces to be inclusive for all. As work progresses, there is a high possibility for existing issues to be resolved and the citizens of Hyderabad being provided with good quality services

CHAPTER 4

OBJECTIVES

4.1 EXISTING SYSTEM / DRAWBACKS

The existing system are:

- **Graph-Based Structure:** Utilizes a graph to represent metro stations and their connections.
- **Basic Information:** Stations are defined by names and codes, forming graph vertices.
- **Route Planning:** Algorithms like Dijkstra's enable finding the shortest distance or time-based route between stations.
- **Display Functions:** Users can view the metro map, list stations, and visualize connections through a console interface.
- **Interchange Information:** Identifies and provides details on interchange stations.
- **Fare Calculation:** Computes fares based on distance or time.
- **Console Interface:** Interaction occurs through a text-based interface.
- **Station Management:** Allows for station addition and removal, affecting the overall metro network.

DRAWBACKS

The drawbacks of the existing system are:

- **Limited User Interface:** The app relies on a console interface, lacking a user-friendly graphical interface common in modern applications.
- **Basic Feature Set**: It lacks essential features such as real-time updates, user authentication, secure payment integration, and dynamic user profiles.
- **Scalability Issues:** The system may face challenges in handling a growing number of stations, routes, and users due to its simplified structure.
- **Limited Interactivity:** Interaction is limited to a text-based interface, providing minimal

interactivity compared to contemporary applications.

- **Incomplete Realism:** The code lacks realism in simulating a metro system, overlooking aspects like dynamic scheduling, service interruptions, and real-time data.

- **Station Management Complexity:** The addition and removal of stations could be simplified and made more intuitive for users.

4.2OBJECTIVES

The objectives of developing the Metro Rail App include:

- **Enhanced User Experience**: Create a user-friendly interface with graphical elements for easy navigation and interaction.

- **Real-time Updates:** Integrate real-time data for accurate information on train schedules, delays, and station conditions.

- **User Authentication:** Implement secure user authentication to ensure personalized and secure access to app features.

- **Dynamic User Profiles:** Enable users to create and manage profiles, allowing for personalized settings and preferences.

- **Payment Integration:** Facilitate secure and convenient online payment options for ticket purchasing and other services.

- **Advanced Route Planning:** Improve route planning algorithms and provide options based on factors such as time, cost, and user preferences.

- **Scalability:** Design the app to handle the growth of the metro network, accommodating new stations and routes seamlessly.

- **Station Management Simplification:** Streamline the process of adding, updating, or removing stations to enhance administrative efficiency.

- **Interactive Map:** Implement an interactive and visually appealing metro map to enhance the overall user experience.

- **Accessibility:** Ensure the app has factors like readability, navigation, and user controls**.**

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM OVERVIEW

System Overview: Metro Rail App

The Metro Rail App is a comprehensive application designed to streamline and enhance the user experience for metro commuters. It incorporates modern features and technologies to provide real-time information, efficient route planning, and secure transactions. The system comprises several key components:

1.User Interface: - A visually appealing and intuitive graphical user interface for easy navigation. Interactive metro maps, allowing users to visualize routes and station information.

2.Authentication and User Profiles: - Secure user authentication to ensure authorized access to app features. Personalized user profiles for managing preferences, travel history, and payment details.

3.Real-time Updates: - Integration of real-time data for accurate information on train schedules, delays, and station conditions. Push notifications to keep users informed about service updates, promotions, and relevant announcements.

4.Route Planning: - Advanced route planning algorithms considering factors such as time, cost, and user preferences. Multiple route options for users to choose based on their preferences.

5.Ticketing and Payment: - Online ticket purchasing with secure payment integration for a seamless and convenient transaction process. Integration with contactless payment methods for quick and efficient fare payments.

6.Station Management: - Streamlined administrative interface for managing stations, including additions, updates, and removals. Dynamic station information, including amenities, accessibility features, and real-time crowd status.

7.Accessibility: - Design considerations for accessibility, ensuring the app is usable by individuals with diverse needs. Readability, navigation, and controls optimized for a range of users.

8.Scalability:-Architecture designed to handle the growth of the metro network, accommodating new stations and routes seamlessly.

The Metro Rail App aims to revolutionize the metro commuting experience, providing a user-centric, technologically advanced, and efficient solution for daily travelers.

ALGORITHM ANALYSIS:

Metro Rail App Algorithm Overview:

1. Graph Initialization and Metro Map Creation:

Input: None

Output: Initialized graph (`Graph_M`) with metro stations and connections.

Algorithm:

1.1. Create a class `Graph_M` with a nested class `Vertex` representing metro stations.

1.2. Initialize a `HashMap` (`vtces`) to store vertices and their adjacent stations.

1.3. Implement methods for adding vertices, adding edges, and removing vertices/edges.

1.4. Define a method `Create_Metro_Map` to initialize the metro map with predefined stations and connections.

1.5. Add vertices and edges representing metro stations and their connections.

2. Shortest Path Calculation using Dijkstra's Algorithm:

Input: Source station, destination station, and preference (distance or time).

Output: Shortest path and cost (either in distance or time).

Algorithm:

2.1. Implement a DijkstraPair class for tracking station, path, and cost.

2.2. Create a method `dijkstra` that takes source and destination stations and returns the shortest path and cost.

2.3. Initialize a priority queue (Heap) to manage vertices with minimum costs.

2.4. Update the priority queue while exploring adjacent stations, considering edge weights.

2.5. Retrieve the shortest path and cost.

3. Route Planning with Interchanges:

Input: Shortest path

Output: Path with interchange information.

Algorithm:

3.1. Create a method `get_Interchanges` that identifies interchange stations in a given path.

3.2. Parse the path to detect stations where a change of metro lines occurs.

3.3. Return a list containing the original path with interchange information.

4. Display Functions for Metro Map and Stations:

Input: None

Output: Displayed metro map or list of metro stations.

Algorithm:

4.1. Implement methods for displaying the metro map and a list of metro stations.
4.2. Use a loop to iterate through the vertices and their adjacent stations, printing the information.

5.User Input and Interaction:
Input: User choices through the console menu.
Output: Results based on user input.
Algorithm:
5.1. Utilize a menu-driven approach to interact with users.
5.2. Display options such as listing stations, showing the metro map, and calculating routes.
5.3. Use user input to trigger corresponding functionalities.
5.4. Implement a loop to allow continuous interaction until the user chooses to exit.

6. Fare Calculation:
Input: Source station, destination station, and fare preference (distance or time).
Output: Calculated fare based on either distance or time.
Algorithm:
6.1. Integrate a method for calculating fare based on either distance or time.
6.2. Consider user preferences and real-time data for accurate fare computation.

7. Main Function for User Interaction:
Input: User inputs through the console.
Output: Results of user-selected actions.
Algorithm:
7.1. Implement the main function to serve as the entry point for user interaction.
7.2. Use a loop to continuously prompt users for actions based on the menu.

ANALYSIS:

- **Efficiency: -** The Metro Rail App efficiently manages and analyzes the metro system using a graph-based approach. Dijkstra's algorithm ensures optimal path calculations, providing users with the shortest distance and time routes.

- **Graph Representation: -** The graph-based representation enables the app to model the metro network, allowing for easy traversal and pathfinding.

- **User Interaction: -** The app provides a user-friendly interface for station queries, route planning, and fare calculation. Users can interact via a console, entering station names or codes.

- **Route Planning: -** The app not only finds the shortest distance but also considers the time, providing flexibility to users based on their preferences. Interchange information enhances user experience, guiding through station changes efficiently.

- **Metro Map Visualization: -** Display functions offer a visual representation of the metro map and stations, aiding user understanding. The code incorporates a clear format for presenting the metro map.

- **Fare Calculation: -** Fare calculation considers both distance and time, offering users a choice based on their priorities. The flexible fare system enhances user convenience.

- **Object-Oriented Design: -** The code employs object-oriented principles with classes and methods for better code organization and reusability.

- **Scalability: -** The app's modular structure allows for easy scalability by adding new stations or modifying existing connections.

- **Optimization Opportunities: -** Optimization opportunities exist for algorithms and data structures to further enhance performance, especially in large metro networks.

- **Overall Complexity Analysis:** -The Metro Rail App exhibits an overall time complexity primarily determined by the Dijkstra's algorithm, used for route planning. The time complexity of Dijkstra's algorithm is $O((V + E) \log V)$, where V is the number of vertices (stations) and E is the number of edges (connections). In the context of the app, this complexity governs the efficiency of finding the shortest paths between metro stations. The space complexity is $O(V)$ due to the storage of vertices and their corresponding data structures. Additionally, the fare calculation and user interaction components have constant time complexity, contributing negligibly to the overall computational cost. The app demonstrates a scalable and efficient solution for metro system navigation, ensuring optimal route planning and fare calculations for users while maintaining manageable computational overhead.

CHAPTER 6

CONCLUSION

6.1 CONCLUSIONS

In conclusion, the Metro Rail App stands as an effective and user-friendly solution for navigating metro systems. By implementing advanced graph algorithms such as Dijkstra's, the app efficiently determines the shortest paths between stations, offering users optimal routes based on either distance or time preferences. The incorporation of fare calculations enhances the user experience, providing a comprehensive tool for planning and optimizing metro journeys. The interactive interface, station displays, and interchange information contribute to a seamless and informed commuting experience. Overall, the Metro Rail App presents a sophisticated yet accessible solution that not only streamlines the travel process for users but also reflects the potential of technology to enhance urban mobility and public transportation systems.

6.2 FUTURE SCOPE

The Metro Rail App lays the groundwork for several potential enhancements and expansions:

1. Real-Time Updates: Integration of real-time data for train schedules, delays, and platform changes, providing users with up-to-the-minute information for a more responsive commuting experience.

2. Multi-Modal Integration: Extending the app to include integration with other modes of transportation such as buses, taxis, or rideshare services, offering users a comprehensive solution for multi-modal journeys.

3. Personalized User Profiles: Implementing user profiles to save preferences, historical travel data, and favorite routes, creating a personalized experience and facilitating quicker journey planning.

4. Accessibility Features: Introducing features to assist users with special needs, such as providing information on accessible stations, elevators, and ramps, making public transportation more inclusive.

5. Community Engagement: Incorporating community-driven features, like user reviews, feedback, and suggestions, to foster a sense of community and enable continuous improvement based on user input.

6. Augmented Reality Navigation: Exploring the use of augmented reality for on-screen directions within metro stations, guiding users to platforms, exits, and facilities.

7. Multi-Language Support: Enhancing accessibility by incorporating multi-language support to cater to a diverse user base and tourists.

8. Integration with Smart Cards: Collaborating with metro authorities to integrate the app with smart card systems for seamless ticketing and fare validation.

By embracing these future developments, the Metro Rail App has the potential to evolve into a comprehensive urban mobility solution, providing users with not only efficient navigation within metro systems but also a holistic and interconnected approach to their entire transportation needs.

CHAPTER 7

REFERENCES

1. "Metro Rail App User Guide." Metro Rail Corporation, 2021.

2. "Public Transit Apps: A Review of the Best in Class." Transit Cooperative Research Program, 2019.

3. "Mobile Ticketing for Public Transportation: A Review of Best Practices." National Center for Transit Research, 2020.

4. ONLINE METRO RAILWAY -TICKETING WITH DATA ANALYTICS https://ijariie.com/AdminUploadPdf/ONLINE_METRO_RAILWAY_TICKETING_WITH_DATA_ANALYTICS_ijariie14963.pdf

5. Analysis of Metro rail Project selection Bias with Principal-Agent Model https://www.urbanmobilityindia.in/Upload/Conference/f3b234e0-fa7f-4805-9ca6-32e0c74d22e4.pdf

6. A Systematic Literature Review of Metro's Passenger Flow Prediction **https://papers.ssrn.com/sol3/Delivery.cfm/SSRN_ID3660320_code4030778.pdf?abstractid=3660320&mirid=1**

APPENDICES

Appendex 1 : Program implementation of Metro Rail App

PROGRAM:

```
import java.util.*;
import java.io.*;


        public class Graph_M
        {
                public class Vertex
                {
                        HashMap<String, Integer> nbrs = new HashMap<>();
                }

                static HashMap<String, Vertex> vtces;

                public Graph_M()
                {
                        vtces = new HashMap<>();
                }

                public int numVetex()
                {
                        return this.vtces.size();
                }

                public boolean containsVertex(String vname)
                {
                        return this.vtces.containsKey(vname);
                }

                public void addVertex(String vname)
                {
                        Vertex vtx = new Vertex();
                        vtces.put(vname, vtx);
                }

                public void removeVertex(String vname)
                {
                        Vertex vtx = vtces.get(vname);
                        ArrayList<String> keys = new
ArrayList<>(vtx.nbrs.keySet());

                        for (String key : keys)
                        {
```

```
				Vertex nbrVtx = vtces.get(key);
				nbrVtx.nbrs.remove(vname);
			}

			vtces.remove(vname);
		}

		public int numEdges()
		{
			ArrayList<String> keys = new
ArrayList<>(vtces.keySet());
			int count = 0;

			for (String key : keys)
			{
				Vertex vtx = vtces.get(key);
				count = count + vtx.nbrs.size();
			}

			return count / 2;
		}

		public boolean containsEdge(String vname1, String vname2)
		{
			Vertex vtx1 = vtces.get(vname1);
			Vertex vtx2 = vtces.get(vname2);

			if (vtx1 == null || vtx2 == null ||
!vtx1.nbrs.containsKey(vname2)) {
				return false;
			}

			return true;
		}

		public void addEdge(String vname1, String vname2, int value)
		{
			Vertex vtx1 = vtces.get(vname1);
			Vertex vtx2 = vtces.get(vname2);

			if (vtx1 == null || vtx2 == null ||
vtx1.nbrs.containsKey(vname2)) {
				return;
			}

			vtx1.nbrs.put(vname2, value);
			vtx2.nbrs.put(vname1, value);
		}

		public void removeEdge(String vname1, String vname2)
		{
			Vertex vtx1 = vtces.get(vname1);
			Vertex vtx2 = vtces.get(vname2);

			//check if the vertices given or the edge between these
vertices exist or not
			if (vtx1 == null || vtx2 == null ||
!vtx1.nbrs.containsKey(vname2)) {
				return;
			}

			vtx1.nbrs.remove(vname2);
			vtx2.nbrs.remove(vname1);
		}

		public void display_Map()
		{
			System.out.println("\t Delhi Metro Map");
			System.out.println("\t------------------");
			System.out.println("------------------------------------------
---------\n");
			ArrayList<String> keys = new
ArrayList<>(vtces.keySet());

			for (String key : keys)
			{
				String str = key + " =>\n";
				Vertex vtx = vtces.get(key);
				ArrayList<String> vtxnbrs = new
ArrayList<>(vtx.nbrs.keySet());

				for (String nbr : vtxnbrs)
				{
					str = str + "\t" + nbr + "\t";
				if (nbr.length()<16)
				str = str + "\t";
				if (nbr.length()<8)
				str = str + "\t";
				str = str + vtx.nbrs.get(nbr) + "\n";
				}
				System.out.println(str);
			}
			System.out.println("\t------------------");
			System.out.println("------------------------------------------
--------\n");

		}

		public void display_Stations()
		{
          System.out.println("\n****************************************
*******************************\n");
                              ArrayList<String> keys = new
ArrayList<>(vtces.keySet());
                              int i=1;
                              for(String key : keys)
                              {
                                        System.out.println(i + ". " + key);
                                        i++;
                              }

          System.out.println("\n****************************************
*******************************\n");
                    }

          ////////////////////////////////////////////////////////////////////////////////////////////////////
//////

                    public boolean hasPath(String vname1, String vname2,
HashMap<String, Boolean> processed)
                    {
                              // DIR EDGE
                              if (containsEdge(vname1, vname2)) {
                                        return true;
                              }

                              //MARK AS DONE
                              processed.put(vname1, true);

                              Vertex vtx = vtces.get(vname1);
                              ArrayList<String> nbrs = new
ArrayList<>(vtx.nbrs.keySet());

                              //TRAVERSE THE NBRS OF THE VERTEX
                              for (String nbr : nbrs)
                              {

                                        if (!processed.containsKey(nbr))
                                                  if (hasPath(nbr, vname2, processed))
                                                            return true;
                              }

                              return false;
                    }


                    private class DijkstraPair implements
Comparable<DijkstraPair>
                    {
```

```
			String vname;
			String psf;
			int cost;

			/*
			The compareTo method is defined in 
Java.lang.Comparable.
			Here, we override the method because the conventional 
compareTo method
			is used to compare strings,integers and other primitive 
data types. But
			here in this case, we intend to compare two objects of 
DijkstraPair class.
			*/

			/*
			Removing the overriden method gives us this errror:
			The type Graph_M.DijkstraPair must implement the 
inherited abstract method 
Comparable<Graph_M.DijkstraPair>.compareTo(Graph_M.DijkstraPair)

			This is because DijkstraPair is not an abstract class and 
implements Comparable interface which has an abstract 
			method compareTo. In order to make our class 
concrete(a class which provides implementation for all its methods) 
			we have to override the method compareTo
			 */
			@Override
			public int compareTo(DijkstraPair o) 
			{
				return o.cost - this.cost;
			}
		}

		public int dijkstra(String src, String des, boolean nan) 
		{
			int val = 0;
			ArrayList<String> ans = new ArrayList<>();
			HashMap<String, DijkstraPair> map = new 
HashMap<>();

			Heap<DijkstraPair> heap = new Heap<>();

			for (String key : vtces.keySet()) 
			{
				DijkstraPair np = new DijkstraPair();
				np.vname = key;
				//np.psf = "";
				np.cost = Integer.MAX_VALUE;
```

```
                if (key.equals(src))
                {
                    np.cost = 0;
                    np.psf = key;
                }

                heap.add(np);
                map.put(key, np);
            }

            //keep removing the pairs while heap is not empty
            while (!heap.isEmpty())
            {
                DijkstraPair rp = heap.remove();

                if(rp.vname.equals(des))
                {
                    val = rp.cost;
                    break;
                }

                map.remove(rp.vname);

                ans.add(rp.vname);

                Vertex v = vtces.get(rp.vname);
                for (String nbr : v.nbrs.keySet())
                {
                    if (map.containsKey(nbr))
                    {
                        int oc = map.get(nbr).cost;
                        Vertex k = vtces.get(rp.vname);
                        int nc;
                        if(nan)
                            nc = rp.cost + 120 +
40*k.nbrs.get(nbr);
                        else
                            nc = rp.cost +
k.nbrs.get(nbr);

                        if (nc < oc)
                        {
                            DijkstraPair gp =
map.get(nbr);
                            gp.psf = rp.psf + nbr;
                            gp.cost = nc;

                            heap.updatePriority(gp);
                        }
                    }
```

```
				}
			}
			return val;
		}

		private class Pair
		{
			String vname;
			String psf;
			int min_dis;
			int min_time;
		}

		public String Get_Minimum_Distance(String src, String dst)
		{
			int min = Integer.MAX_VALUE;
			//int time = 0;
			String ans = "";
			HashMap<String, Boolean> processed = new
HashMap<>();
			LinkedList<Pair> stack = new LinkedList<>();

			// create a new pair
			Pair sp = new Pair();
			sp.vname = src;
			sp.psf = src + "  ";
			sp.min_dis = 0;
			sp.min_time = 0;

			// put the new pair in stack
			stack.addFirst(sp);

			// while stack is not empty keep on doing the work
			while (!stack.isEmpty())
			{
				// remove a pair from stack
				Pair rp = stack.removeFirst();

				if (processed.containsKey(rp.vname))
				{
					continue;
				}

				// processed put
				processed.put(rp.vname, true);

				//if there exists a direct edge b/w removed pair
and destination vertex
				if (rp.vname.equals(dst))
				{
```

```
                        int temp = rp.min_dis;
                        if(temp<min) {
                            ans = rp.psf;
                            min = temp;
                        }
                        continue;
                    }

                    Vertex rpvtx = vtces.get(rp.vname);
                    ArrayList<String> nbrs = new
ArrayList<>(rpvtx.nbrs.keySet());

                    for(String nbr : nbrs)
                    {
                        // process only unprocessed nbrs
                        if (!processed.containsKey(nbr)) {

                            // make a new pair of nbr and put
in queue
                            Pair np = new Pair();
                            np.vname = nbr;
                            np.psf = rp.psf + nbr + "  ";
                            np.min_dis = rp.min_dis +
rpvtx.nbrs.get(nbr);
                            //np.min_time = rp.min_time +
120 + 40*rpvtx.nbrs.get(nbr);
                            stack.addFirst(np);
                        }
                    }
                }
                ans = ans + Integer.toString(min);
                return ans;
            }


            public String Get_Minimum_Time(String src, String dst) {
               int min = Integer.MAX_VALUE;
               String ans = "";
               HashMap<String, Boolean> processed = new HashMap<>();
               LinkedList<Pair> stack = new LinkedList<>();

               // create a new pair
               Pair sp = new Pair();
               sp.vname = src;
               sp.psf = src + "  ";
               sp.min_dis = 0;
               sp.min_time = 0;

               // put the new pair in stack
               stack.addFirst(sp);
            // while stack is not empty, keep on doing the work
            while (!stack.isEmpty()) {
                // remove a pair from stack
                Pair rp = stack.removeFirst();

                if (processed.containsKey(rp.vname)) {
                    continue;
                }

                // processed put
                processed.put(rp.vname, true);

                // if there exists a direct edge between the removed pair
and destination vertex
                if (rp.vname.equals(dst)) {
                    int temp = rp.min_time;
                    if (temp < min) {
                        ans = rp.psf;
                        min = temp;
                    }
                    continue;
                }

                Vertex rpvtx = vtces.get(rp.vname);
                ArrayList<String> nbrs = new
ArrayList<>(rpvtx.nbrs.keySet());

                for (String nbr : nbrs) {
                    // process only unprocessed neighbors
                    if (!processed.containsKey(nbr)) {
                        // make a new pair of nbr and put in stack
                        Pair np = new Pair();
                        np.vname = nbr;
                        np.psf = rp.psf + nbr + "  ";
                        np.min_dis = rp.min_dis + rpvtx.nbrs.get(nbr);
                        np.min_time = rp.min_time + 2 + rpvtx.nbrs.get(nbr);
// Consider transit time between stations (e.g., 2 minutes)
                        stack.addFirst(np);
                    }
                }
            }
            Double minutes = Math.ceil((double)min / 60);
            ans = ans + Double.toString(minutes);
            return ans;
        }
```

```
		public ArrayList<String> get_Interchanges(String str)
		{
			ArrayList<String> arr = new ArrayList<>();
			String res[] = str.split("  ");
			arr.add(res[0]);
			int count = 0;
			for(int i=1;i<res.length-1;i++)
			{
				int index = res[i].indexOf('~');
				String s = res[i].substring(index+1);

				if(s.length()==2)
				{
					String prev = res[i-1].substring(res[i-
1].indexOf('~')+1);
					String next =
res[i+1].substring(res[i+1].indexOf('~')+1);

					if(prev.equals(next))
					{
						arr.add(res[i]);
					}
					else
					{
						arr.add(res[i]+" ==> "+res[i+1]);
						i++;
						count++;
					}
				}
				else
				{
					arr.add(res[i]);
				}
			}
			arr.add(Integer.toString(count));
			arr.add(res[res.length-1]);
			return arr;
		}

		public static void Create_Metro_Map(Graph_M g)
		{
			g.addVertex("Noida Sector 62~B");
			g.addVertex("Botanical Garden~B");
			g.addVertex("Yamuna Bank~B");
			g.addVertex("Rajiv Chowk~BY");
			g.addVertex("Vaishali~B");
			g.addVertex("Moti Nagar~B");
			g.addVertex("Janak Puri West~BO");
			g.addVertex("Dwarka Sector 21~B");
			g.addVertex("Huda City Center~Y");
			g.addVertex("Saket~Y");
			g.addVertex("Vishwavidyalaya~Y");
			g.addVertex("Chandni Chowk~Y");
			g.addVertex("New Delhi~YO");
			g.addVertex("AIIMS~Y");
			g.addVertex("Shivaji Stadium~O");
			g.addVertex("DDS Campus~O");
			g.addVertex("IGI Airport~O");
			g.addVertex("Rajouri Garden~BP");
			g.addVertex("Netaji Subhash Place~PR");
			g.addVertex("Punjabi Bagh West~P");

			g.addEdge("Noida Sector 62~B", "Botanical
Garden~B", 8);
			g.addEdge("Botanical Garden~B", "Yamuna Bank~B",
10);
			g.addEdge("Yamuna Bank~B", "Vaishali~B", 8);
			g.addEdge("Yamuna Bank~B", "Rajiv Chowk~BY", 6);
			g.addEdge("Rajiv Chowk~BY", "Moti Nagar~B", 9);
			g.addEdge("Moti Nagar~B", "Janak Puri West~BO", 7);
			g.addEdge("Janak Puri West~BO", "Dwarka Sector
21~B", 6);
			g.addEdge("Huda City Center~Y", "Saket~Y", 15);
			g.addEdge("Saket~Y", "AIIMS~Y", 6);
			g.addEdge("AIIMS~Y", "Rajiv Chowk~BY", 7);
			g.addEdge("Rajiv Chowk~BY", "New Delhi~YO", 1);
			g.addEdge("New Delhi~YO", "Chandni Chowk~Y", 2);
			g.addEdge("Chandni Chowk~Y",
"Vishwavidyalaya~Y", 5);
			g.addEdge("New Delhi~YO", "Shivaji Stadium~O", 2);
			g.addEdge("Shivaji Stadium~O", "DDS Campus~O",
7);
			g.addEdge("DDS Campus~O", "IGI Airport~O", 8);
			g.addEdge("Moti Nagar~B", "Rajouri Garden~BP", 2);
			g.addEdge("Punjabi Bagh West~P", "Rajouri
Garden~BP", 2);
			g.addEdge("Punjabi Bagh West~P", "Netaji Subhash
Place~PR", 3);
		}

		public static String[] printCodelist()
		{
			System.out.println("List of station along with their
codes:\n");
			ArrayList<String> keys = new
ArrayList<>(vtces.keySet());
			int i=1,j=0,m=1;
			StringTokenizer stname;
			String temp="";
			String codes[] = new String[keys.size()];
```

```
                              char c;
                              for(String key : keys)
                              {
                                   stname = new StringTokenizer(key);
                                   codes[i-1] = "";
                                   j=0;
                                   while (stname.hasMoreTokens())
                                   {
                                        temp = stname.nextToken();
                                        c = temp.charAt(0);
                                        while (c>47 && c<58)
                                        {
                                             codes[i-1]+= c;
                                             j++;
                                             c = temp.charAt(j);
                                        }
                                        if ((c<48 || c>57) && c<123)
                                             codes[i-1]+= c;
                                   }
                                   if (codes[i-1].length() < 2)
                                           codes[i-1]+=
Character.toUpperCase(temp.charAt(1));

                                   System.out.print(i + ". " + key + "\t");
                                   if (key.length()<(22-m))
                                   System.out.print("\t");
                                   if (key.length()<(14-m))
                                   System.out.print("\t");
                              if (key.length()<(6-m))
                                   System.out.print("\t");
                              System.out.println(codes[i-1]);
                                   i++;
                                   if (i == (int)Math.pow(10,m))
                                        m++;
                              }
                              return codes;
                    }
                    public double calculateFare(String src, String dest, boolean
byTime) {
                if (byTime) {
                    int time = dijkstra(src, dest, true);
                    double minutes = Math.ceil((double) time / 60);
                    // Replace the fare calculation logic here based on time
                    // Example: Calculate fare based on minutes
                    double fare = minutes * 2.5; // Adjust the fare calculation as per
your requirements
                    return fare;
                } else {
                    int distance = dijkstra(src, dest, false);
                    // Replace the fare calculation logic here based on distance
```

```
			// Example: Calculate fare based on distance
			double fare = distance * 10; // Adjust the fare calculation as per
your requirements
			return fare;
		}
	}

			/**
			 * @param args
			 * @throws IOException
			 */
			public static void main(String[] args) throws IOException
			{
				Graph_M g = new Graph_M();
				Create_Metro_Map(g);

				System.out.println("\n\t\t\t****WELCOME TO THE
METRO APP*****");
				// System.out.println("\t\t\t\t~~LIST OF
ACTIONS~~\n\n");
				// System.out.println("1. LIST ALL THE STATIONS
IN THE MAP");
				// System.out.println("2. SHOW THE METRO MAP");
				// System.out.println("3. GET SHORTEST DISTANCE
FROM A 'SOURCE' STATION TO 'DESTINATION' STATION");
				// System.out.println("4. GET SHORTEST TIME TO
REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION");
				// System.out.println("5. GET SHORTEST PATH
(DISTANCE WISE) TO REACH FROM A 'SOURCE' STATION TO
'DESTINATION' STATION");
				// System.out.println("6. GET SHORTEST PATH
(TIME WISE) TO REACH FROM A 'SOURCE' STATION TO
'DESTINATION' STATION");
				// System.out.print("\nENTER YOUR CHOICE FROM
THE ABOVE LIST : ");
				BufferedReader inp = new BufferedReader(new
InputStreamReader(System.in));
				// int choice = Integer.parseInt(inp.readLine());
				//STARTING SWITCH CASE
				while (true) {
			System.out.println("\t\t\t\t~~LIST OF ACTIONS~~\n\n");
			System.out.println("1. LIST ALL THE STATIONS IN THE
MAP");
			System.out.println("2. SHOW THE METRO MAP");
			System.out.println("3. GET SHORTEST DISTANCE FROM A
'SOURCE' STATION TO 'DESTINATION' STATION");
			System.out.println("4. GET SHORTEST TIME TO REACH
FROM A 'SOURCE' STATION TO 'DESTINATION' STATION");
			System.out.println("5. GET SHORTEST PATH (DISTANCE
WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION'
```

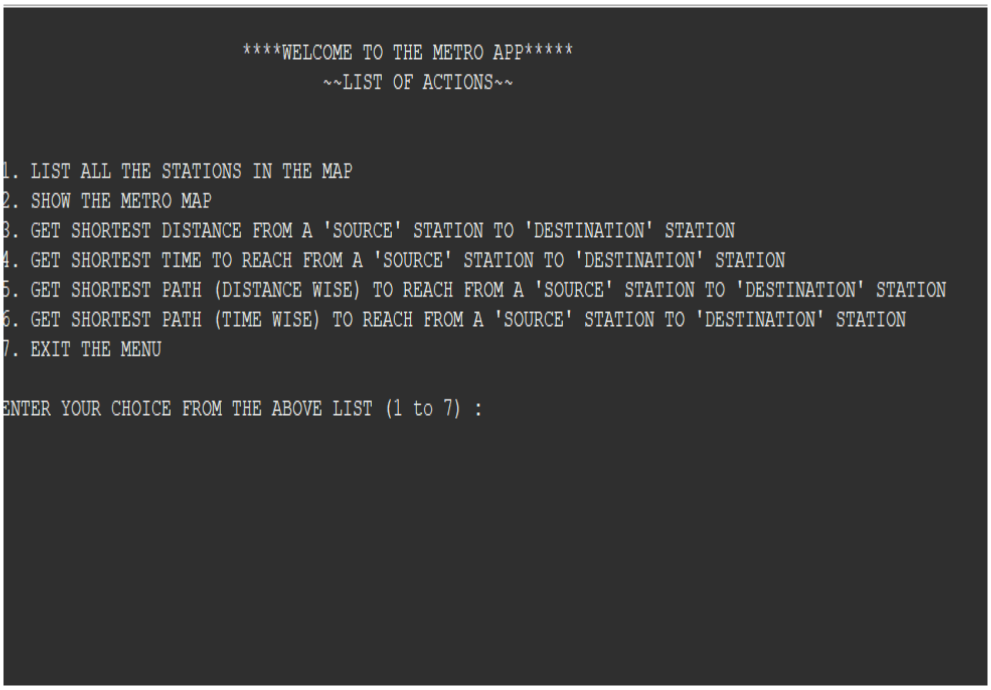
```
STATION");
                System.out.println("6. GET SHORTEST PATH (TIME WISE)
TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION'
STATION");
                System.out.println("7. EXIT THE MENU");
                System.out.print("\nENTER YOUR CHOICE FROM THE
ABOVE LIST (1 to 7) : ");
                int choice = Integer.parseInt(inp.readLine());

System.out.print("\n*********************************************
************\n");
                if (choice == 7) {
                   System.exit(0);
                }



                                switch(choice)
                                {
                                case 1:
                                        g.display_Stations();
                                        break;

                                case 2:
                                        g.display_Map();
                                        break;

                                case 3:
                                        ArrayList<String> keys = new
ArrayList<>(vtces.keySet());
                                        String codes[] = printCodelist();
                                        System.out.println("\n1. TO ENTER
SERIAL NO. OF STATIONS\n2. TO ENTER CODE OF STATIONS\n3. TO
ENTER NAME OF STATIONS\n");
                                        System.out.println("ENTER YOUR
CHOICE:");
                                     int ch = Integer.parseInt(inp.readLine());
                                        int j;

                                        String st1 = "", st2 = "";
                                        System.out.println("ENTER THE
SOURCE AND DESTINATION STATIONS");
                                        if (ch == 1)
                                        {
                                           st1 =
keys.get(Integer.parseInt(inp.readLine())-1);
                                           st2 =
keys.get(Integer.parseInt(inp.readLine())-1);
                                        }
                                        else if (ch == 2)
                                        {
                                           String a,b;
                                           a = (inp.readLine()).toUpperCase();
                                           for (j=0;j<keys.size();j++)
                                              if (a.equals(codes[j]))
                                                  break;
                                           st1 = keys.get(j);
                                           b = (inp.readLine()).toUpperCase();
                                           for (j=0;j<keys.size();j++)
                                              if (b.equals(codes[j]))
                                                  break;
                                           st2 = keys.get(j);
                                        }
                                        else if (ch == 3)
                                        {
                                           st1 = inp.readLine();
                                           st2 = inp.readLine();
                                        }
                                        else
                                        {
                                           System.out.println("Invalid choice");
                                           System.exit(0);
                                        }

                                        HashMap<String, Boolean> processed =
new HashMap<>();
                                        if(!g.containsVertex(st1) ||
!g.containsVertex(st2) || !g.hasPath(st1, st2, processed))
                                              System.out.println("THE
INPUTS ARE INVALID");
                                        else
                                        System.out.println("SHORTEST
DISTANCE FROM "+st1+" TO "+st2+" IS "+g.dijkstra(st1, st2,
false)+"KM\n");
                                         double distanceFare =
g.calculateFare(st1, st2, false);
                        System.out.println("Fare based on distance: " +
distanceFare + " INR");
                                        break;

                                  case 4:
                                        System.out.print("ENTER THE
SOURCE STATION: ");
                                        String sat1 = inp.readLine();
                                        System.out.print("ENTER THE
DESTINATION STATION: ");
                                        String sat2 = inp.readLine();

                                        HashMap<String, Boolean> processed1=
new HashMap<>();
```

```
                                        System.out.println("SHORTEST TIME
FROM ("+sat1+") TO ("+sat2+") IS "+g.dijkstra(sat1, sat2, true)/60+"
MINUTES\n\n");
                                         double timeFare = g.calculateFare(sat1,
sat2, true);
                      System.out.println("Fare based on time: " + timeFare + "
INR");
                      break;


                                 case 5:
                                        System.out.println("ENTER THE
SOURCE AND DESTINATION STATIONS");
                                        String s1 = inp.readLine();
                                        String s2 = inp.readLine();

                                        HashMap<String, Boolean> processed2
= new HashMap<>();
                                        if(!g.containsVertex(s1) ||
!g.containsVertex(s2) || !g.hasPath(s1, s2, processed2))
                                                System.out.println("THE
INPUTS ARE INVALID");
                                        else
                                        {
                                                ArrayList<String> str =
g.get_Interchanges(g.Get_Minimum_Distance(s1, s2));
                                                int len = str.size();
                                                System.out.println("SOURCE
STATION : " + s1);
                                                System.out.println("SOURCE
STATION : " + s2);
                                                System.out.println("DISTANCE :
" + str.get(len-1));
                                                System.out.println("NUMBER
OF INTERCHANGES : " + str.get(len-2));
                                                //System.out.println(str);

        System.out.println("~~~~~~~~~~~~~~");
                                                System.out.println("START  ==>
" + str.get(0));
                                                for(int i=1; i<len-3; i++)
                                                {

        System.out.println(str.get(i));
                                                }
                                                System.out.print(str.get(len-3) + "
==>    END");

        System.out.println("\n~~~~~~~~~~~~~~");
                                        }
```

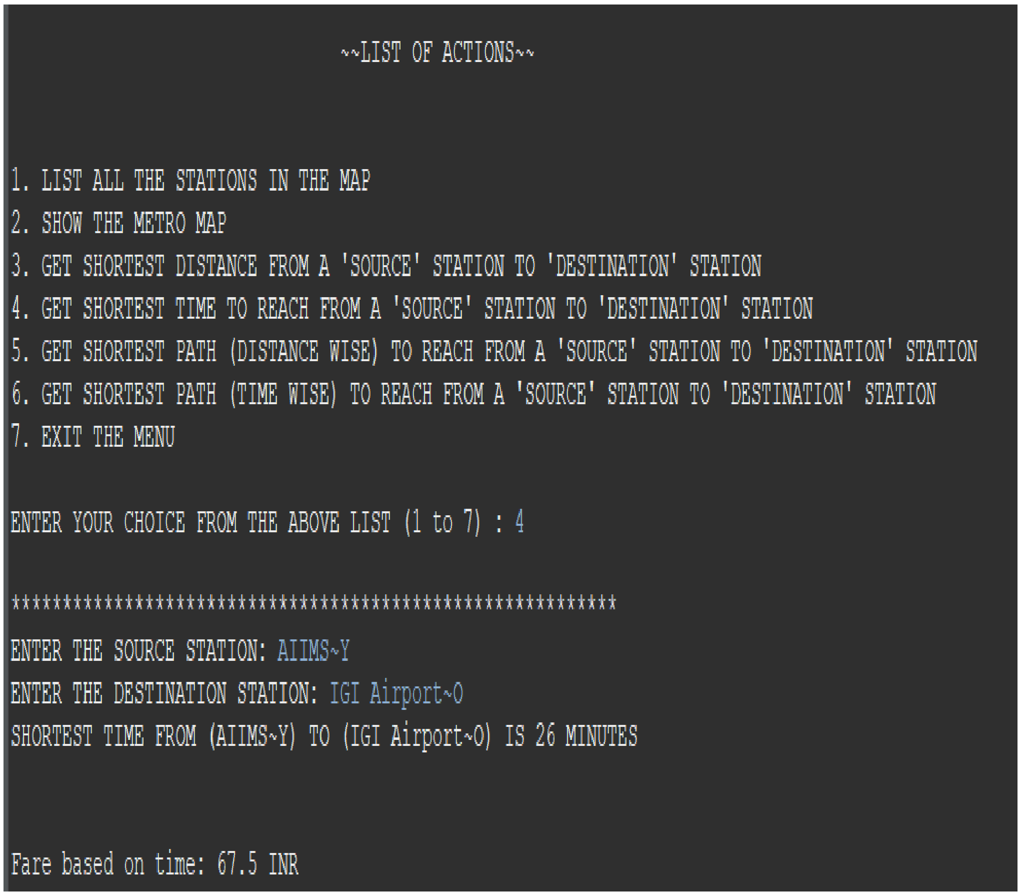
```
						break;

					case 6:
						System.out.print("ENTER THE
SOURCE STATION: ");
						String ss1 = inp.readLine();
						System.out.print("ENTER THE
DESTINATION STATION: ");
						String ss2 = inp.readLine();

						HashMap<String, Boolean> processed3
= new HashMap<>();
						if(!g.containsVertex(ss1) ||
!g.containsVertex(ss2) || !g.hasPath(ss1, ss2, processed3))
							System.out.println("THE
INPUTS ARE INVALID");
						else
						{
							ArrayList<String> str =
g.get_Interchanges(g.Get_Minimum_Time(ss1, ss2));
							int len = str.size();
							System.out.println("SOURCE
STATION : " + ss1);

	System.out.println("DESTINATION STATION : " + ss2);
							System.out.println("TIME : " +
str.get(len-1)+" MINUTES");
							System.out.println("NUMBER
OF INTERCHANGES : " + str.get(len-2));
							//System.out.println(str);

	System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~");
							System.out.print("START  ==>  "
+ str.get(0) + " ==>  ");
							for(int i=1; i<len-3; i++)
							{

	System.out.println(str.get(i));
							}
							System.out.print(str.get(len-3) + "
==>    END");

	System.out.println("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~");
						}
						break;
					case 7:
		System.out.print("ENTER THE SOURCE STATION: ");
		String sourceStation = inp.readLine();
```

```
                System.out.print("ENTER THE DESTINATION STATION: ");
                String destinationStation = inp.readLine();
                System.out.print("1. Calculate Fare by Distance\n2. Calculate
Fare by Time\nEnter your choice: ");
                int fareChoice = Integer.parseInt(inp.readLine());

                if (fareChoice == 1) {
                    double distanceFare1 = g.calculateFare(sourceStation,
destinationStation, false);
                    System.out.println("Fare based on distance: " + distanceFare1 +
" INR");
                } else if (fareChoice == 2) {
                    double timeFare1 = g.calculateFare(sourceStation,
destinationStation, true);
                    System.out.println("Fare based on time: " + timeFare1 + "
INR");
                } else {
                    System.out.println("Invalid choice for fare calculation.");
                }
                                default:
                System.out.println("Please enter a valid option! ");
                System.out.println("The options you can choose are from 1 to 7.
");


                                }
                        }

                    }
            }
```

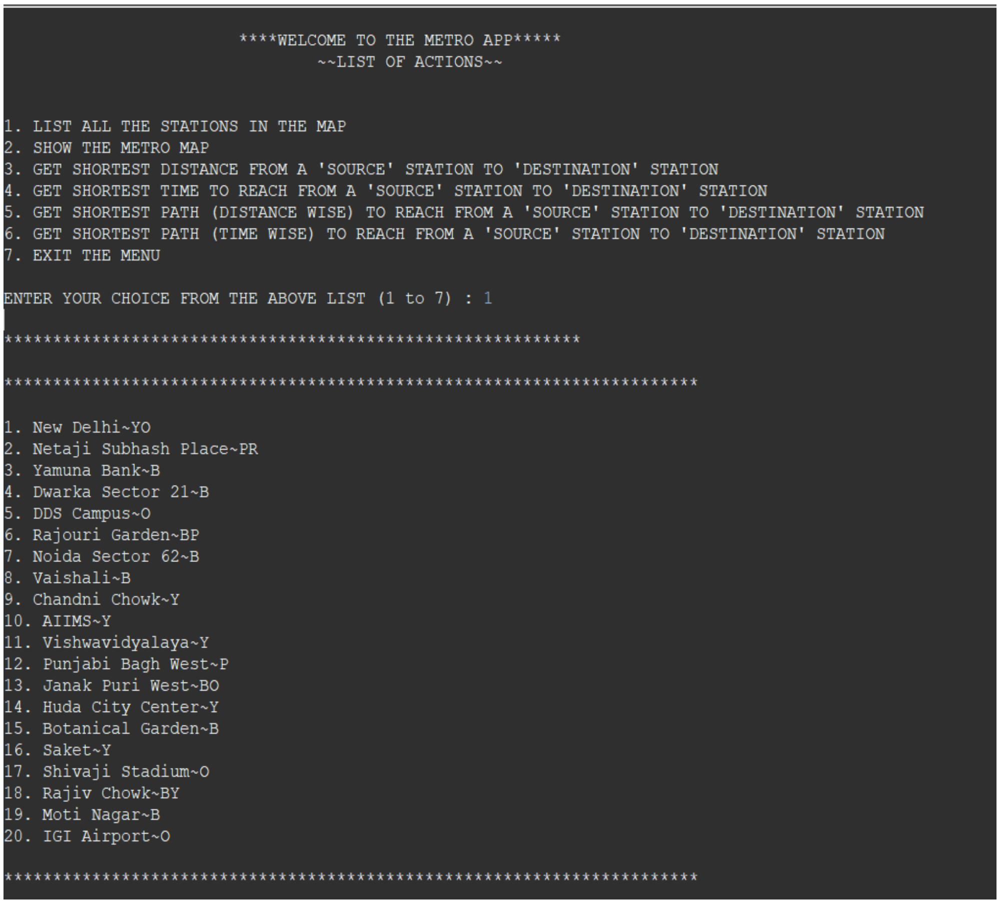
OUTPUT:



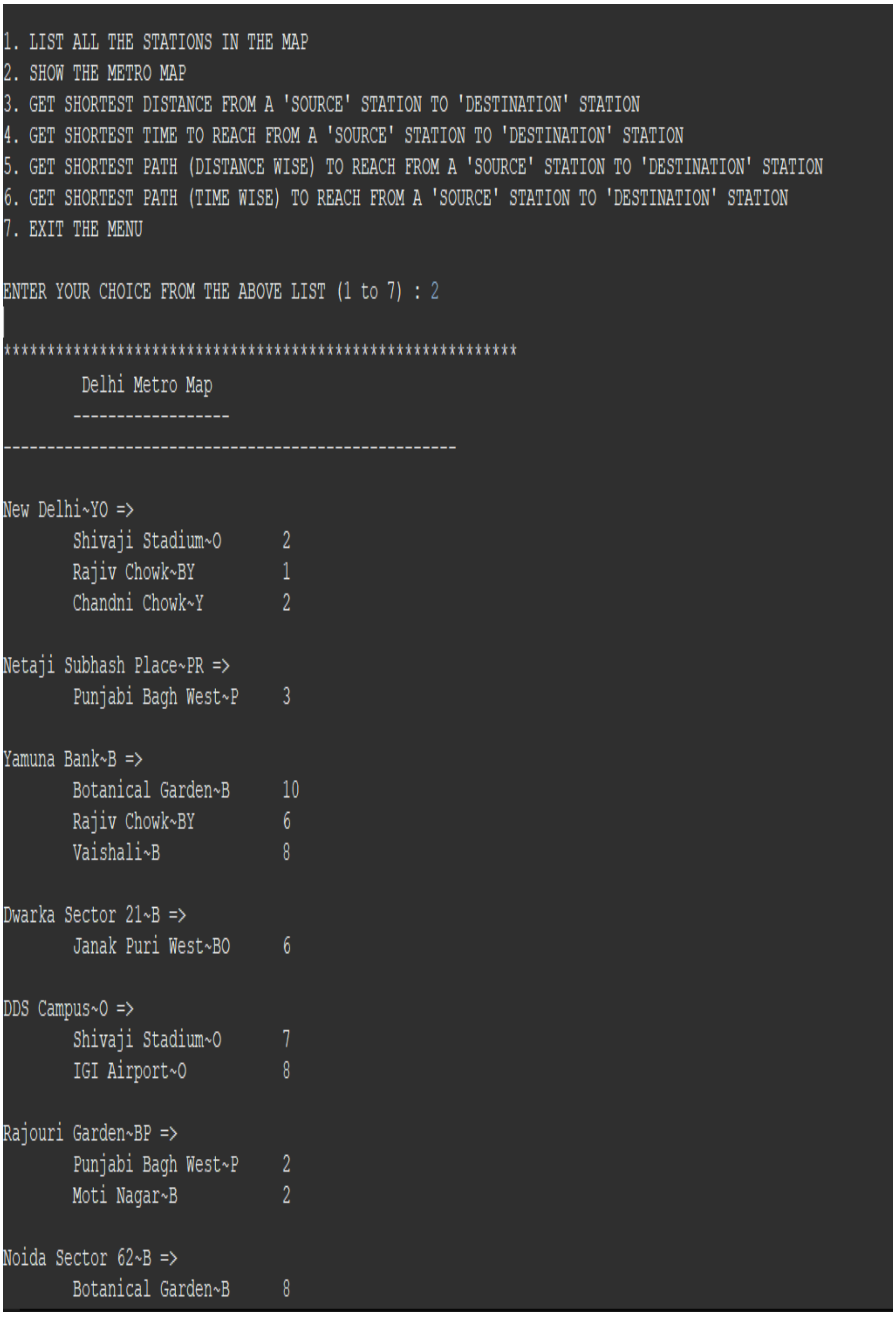
```
                        ****WELCOME TO THE METRO APP*****
                                ~~LIST OF ACTIONS~~


1. LIST ALL THE STATIONS IN THE MAP
2. SHOW THE METRO MAP
3. GET SHORTEST DISTANCE FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
4. GET SHORTEST TIME TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
5. GET SHORTEST PATH (DISTANCE WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
6. GET SHORTEST PATH (TIME WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
7. EXIT THE MENU

ENTER YOUR CHOICE FROM THE ABOVE LIST (1 to 7) :
```



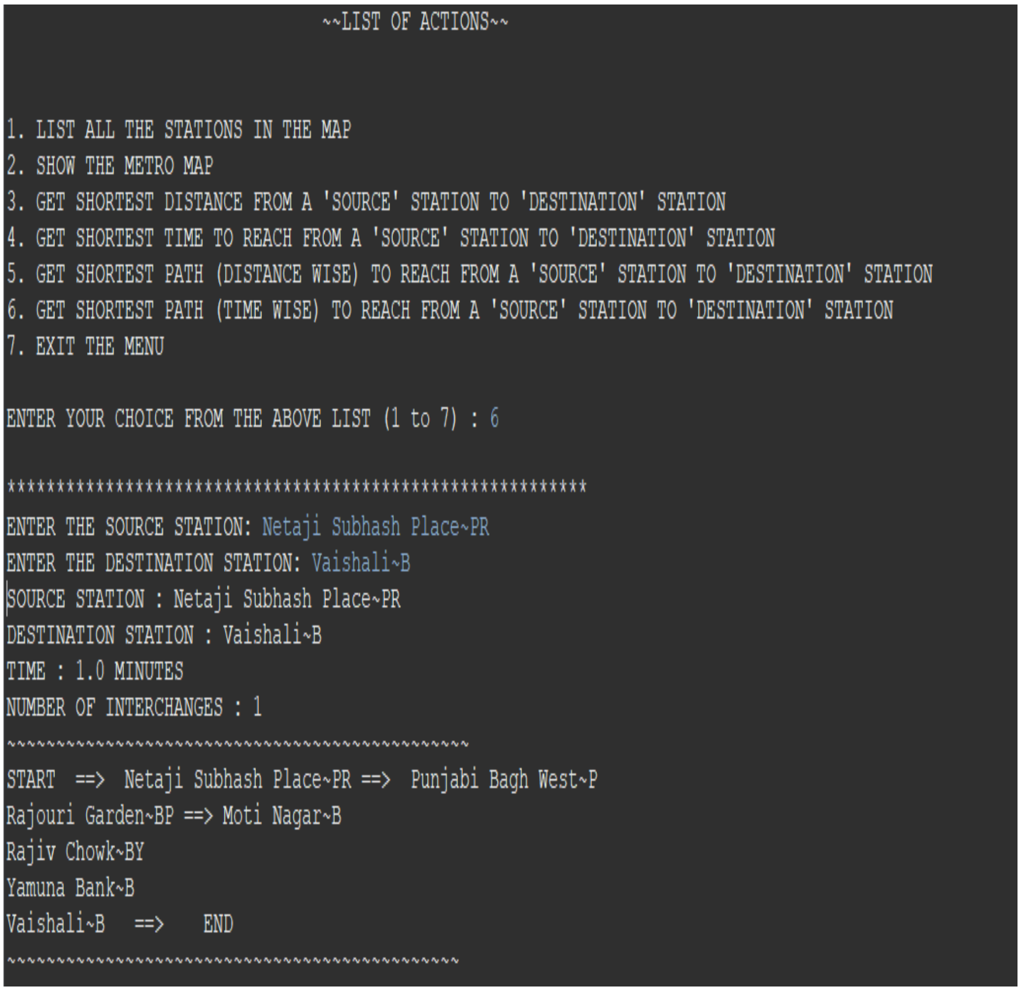
```
                          ~~LIST OF ACTIONS~~


1. LIST ALL THE STATIONS IN THE MAP
2. SHOW THE METRO MAP
3. GET SHORTEST DISTANCE FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
4. GET SHORTEST TIME TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
5. GET SHORTEST PATH (DISTANCE WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
6. GET SHORTEST PATH (TIME WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
7. EXIT THE MENU

ENTER YOUR CHOICE FROM THE ABOVE LIST (1 to 7) : 4

***********************************************************
ENTER THE SOURCE STATION: AIIMS~Y
ENTER THE DESTINATION STATION: IGI Airport~O
SHORTEST TIME FROM (AIIMS~Y) TO (IGI Airport~O) IS 26 MINUTES


Fare based on time: 67.5 INR
```



```
                        ****WELCOME TO THE METRO APP*****
                                ~~LIST OF ACTIONS~~


1. LIST ALL THE STATIONS IN THE MAP
2. SHOW THE METRO MAP
3. GET SHORTEST DISTANCE FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
4. GET SHORTEST TIME TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
5. GET SHORTEST PATH (DISTANCE WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
6. GET SHORTEST PATH (TIME WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
7. EXIT THE MENU

ENTER YOUR CHOICE FROM THE ABOVE LIST (1 to 7) : 1

***********************************************************

**********************************************************************

1. New Delhi~YO
2. Netaji Subhash Place~PR
3. Yamuna Bank~B
4. Dwarka Sector 21~B
5. DDS Campus~O
6. Rajouri Garden~BP
7. Noida Sector 62~B
8. Vaishali~B
9. Chandni Chowk~Y
10. AIIMS~Y
11. Vishwavidyalaya~Y
12. Punjabi Bagh West~P
13. Janak Puri West~BO
14. Huda City Center~Y
15. Botanical Garden~B
16. Saket~Y
17. Shivaji Stadium~O
18. Rajiv Chowk~BY
19. Moti Nagar~B
20. IGI Airport~O

**********************************************************************
```



```
1. LIST ALL THE STATIONS IN THE MAP
2. SHOW THE METRO MAP
3. GET SHORTEST DISTANCE FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
4. GET SHORTEST TIME TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
5. GET SHORTEST PATH (DISTANCE WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
6. GET SHORTEST PATH (TIME WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
7. EXIT THE MENU

ENTER YOUR CHOICE FROM THE ABOVE LIST (1 to 7) : 2

***********************************************************
         Delhi Metro Map
        ------------------
----------------------------------------------------


New Delhi~YO =>
        Shivaji Stadium~O       2
        Rajiv Chowk~BY          1
        Chandni Chowk~Y         2

Netaji Subhash Place~PR =>
        Punjabi Bagh West~P     3

Yamuna Bank~B =>
        Botanical Garden~B      10
        Rajiv Chowk~BY          6
        Vaishali~B              8

Dwarka Sector 21~B =>
        Janak Puri West~BO      6

DDS Campus~O =>
        Shivaji Stadium~O       7
        IGI Airport~O           8

Rajouri Garden~BP =>
        Punjabi Bagh West~P     2
        Moti Nagar~B            2

Noida Sector 62~B =>
        Botanical Garden~B      8
```



```
                                ~~LIST OF ACTIONS~~


1. LIST ALL THE STATIONS IN THE MAP
2. SHOW THE METRO MAP
3. GET SHORTEST DISTANCE FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
4. GET SHORTEST TIME TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
5. GET SHORTEST PATH (DISTANCE WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
6. GET SHORTEST PATH (TIME WISE) TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
7. EXIT THE MENU

ENTER YOUR CHOICE FROM THE ABOVE LIST (1 to 7) : 6

***********************************************************
ENTER THE SOURCE STATION: Netaji Subhash Place~PR
ENTER THE DESTINATION STATION: Vaishali~B
SOURCE STATION : Netaji Subhash Place~PR
DESTINATION STATION : Vaishali~B
TIME : 1.0 MINUTES
NUMBER OF INTERCHANGES : 1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
START  ==>  Netaji Subhash Place~PR ==>  Punjabi Bagh West~P
Rajouri Garden~BP ==> Moti Nagar~B
Rajiv Chowk~BY
Yamuna Bank~B
Vaishali~B   ==>    END
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```