INTERNSHIP -
ROUTE PLANNING APPLICATION

A report submitted in partial fulfillment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY By

BONU PUJITHA

21B91A5426

Under Supervision of Mr. Rafikh (Trainer name)

Henotic IT Solutions Pvt Ltd, Hyderabad

(Duration: 5th July 2023 to 5th September 2023)

DEPARTMENT OF

INFORMATION TECHNOLOGY

S.R.K.R. ENGINEERING COLLEGE(Autonomous)

SRKR MARG, CHINNA AMIRAM, BHIMAVARAM-534204, A.P(Recognized by A.I.C.T.E New Delhi) (Accredited by NBA & NAAC) (Affiliated to JNTU, KAKINADA)

SAGI RAMA KRISHNAM RAJU ENGINEERING COLLEGE

(Autonomous)

DEPARTMENT OF

INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the Summer Internship Report titled **"ROUTE PLANNING APPLICATION"** is the bonafide work done by B.Pujitha bearing 21B91A5426 at the end of second year second semester at Henotic IT Solutions Pvt Ltd, Hyderabad from 5th July 2023 to 5th September 2023 in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in INFORMATION TECHNOLOGY

"

Department Internship coordinator **Dean -T & P Cell** **Head of the Department**

ACKNOWLEDGEMENT

I would like to extend my heartfelt gratitude to **Mr. Rafikh** working in Henotic IT Solutions Pvt Ltd in Hyderabad, for affording me the valuable opportunity to embark on an enriching internship journey within the organization. His belief in me and the chance he provided will forever be appreciated.

I am also deeply indebted to the incredible team at Henotic IT Solutions Pvt Ltd Hyderabad, who worked alongside me during this internship. Their unwavering patience, openness, and collaborative spirit not only made the workplace enjoyable but also fostered an environment conducive to growth and learning. Their support was instrumental in making this experience truly remarkable.

I wish to express my profound gratitude to Principal **Prof. M. Jagapathi Raju** for the facilities and resources provided, which were instrumental in the successful completion of my internship.

I would like to acknowledge and thank my Head of the Department **Prof. Bh.V.S.R.K.Raju**, for his invaluable constructive criticism and guidance throughout my internship, which played a pivotal role in enhancing my skills and knowledge.

I am also indebted to **Sri. K N V S Varma**, the internship coordinator of the Department of IT, for their unwavering support and valuable advice that not only helped me secure this internship but also guided me in successfully completing it.

I am extremely greatful to my department staff members and friends who helped me in successful completion of this internship.

B.Pujitha

(21B91A5426)

Internship Objectives

- Internships are generally thought of to be reserved for college students looking to gain experience in a particular field.

- However, a wide array of people can benefit from Training Internships in order to receive real world experience and develop their skills.

- An objective for this position should emphasize the skills you already possess in the area and your interest in learning more.

- Internships are utilized in a number of different career fields, including architecture, engineering, healthcare, economics, advertising and many more.

- Some internship is used to allow individuals to perform scientific research while others are specifically designed to allow people to gain first-hand experience working.

- Utilizing internships is a great way to build your resume and develop skills that can be emphasized in your resume for future jobs.

- When you are applying for a Training Internship, make sure to highlight any special skills or talents that can make you stand apart from the rest of the applicants so that you have an improved chance of landing the positions.

WEEKLY OVERVIEW OF INTERNSHIP ACTIVITIES

	DATE	DAY	NAME OF THE TOPIC/MODULE COMPLETED
1st WEEK	05/07/23	Wednesday	Introduction to java
	06/07/23	Thursday	Array and Loops concepts
	07/07/23	Friday	Eclipse (IDE) installation
	08/07/23	Saturday	OOPs concepts
	09/07/23	Sunday	Modifiers and Interfaces

	DATE	DAY	NAME OF THE TOPIC/MODULE COMPLETED
2nd WEEK	10/07/23	Monday	Abstraction and Debugging
	11/07/23	Tuesday	Special keywords in java
	12/07/23	Wednesday	Basics of DS
	13/07/23	Thursday	Stack and Linked Lists
	14/07/23	Friday	Double linked List
	15/07/23	Saturday	Queue and circular queues

	DATE	DAY	NAME OF THE TOPIC/MODULE COMPLETED
3rd WEEK	17/07/23	Monday	Tree Data Structures
	18/07/23	Tuesday	Tree Traversals
	19/07/23	Wednesday	Sorting Algorthims
	20/07/23	Thursday	Array List
	21/07/23	Friday	Heap Data Structure
	22/07/23	Saturday	Hash Table and Hash Maps

	DATE	DAY	NAME OF THE TOPIC/MODULE COMPLETED
4th WEEK	24/07/23	Monday	Exploring Projects Ideas
	25/07/23	Tuesday	Discussing With Trainer
	26/07/23	Wednesday	Project Session
	27/07/23	Thursday	Finalizing the ideas
	28/07/23	Friday	Building Up the Requirements
	29/07/23	Saturday	Coding

	DATE	DAY	NAME OF THE TOPIC/MODULE COMPLETED
5th WEEK	30/07/23	Monday	Coding
	31/07/23	Tuesday	
	01/08/23	Wednesday	
	02/08/23	Thursday	Testing
	03/08/23	Friday	Enhancement
	04/08/23	Saturday	Submission

TABLE OF CONTENTS

1.ABSTRACT

The **Metro Route Optimization System** presented in this project provides an efficient and user-friendly solution for navigating a metro network. The system is implemented using Java, focusing on graph data structures and algorithms. It enables users to explore metro stations, find the shortest distance, and calculate optimal travel routes based on both distance and time constraints.

Key Features:

1. **Graph Representation:** The metro network is modeled as a graph, where stations are vertices and connections between stations are edges. This representation allows for efficient navigation and pathfinding.
2. **Dijkstra's Algorithm:** The system employs Dijkstra's algorithm to find the shortest distance and time paths between two metro stations. This algorithm ensures accurate and optimal route planning for users.
3. **Interactive User Interface:** The system provides an interactive command-line interface, allowing users to input station names, codes, or serial numbers for route planning. It also displays a list of available actions for the users to choose from.
4. **Station Codes:** To enhance user experience, each station is associated with a unique code. This feature simplifies user input and ensures a seamless interaction with the system.
5. **Interchange Detection:** The system identifies interchange stations where passengers can switch between different metro lines. It calculates the number of interchanges and provides detailed route information to the users.
6. **Optimized Path Output:** The system outputs optimized paths in a clear and structured manner. It displays the starting station, intermediate stops, and the destination station, along with the total distance or time required for the journey.

2.PROBLEM STATEMENT:

Urban metro networks often pose challenges to commuters, including complex routes, multiple interchanges, and varying travel times. Navigating through these networks efficiently and making informed decisions regarding optimal routes can be daunting for passengers. Existing metro information systems may lack real-time updates and often do not consider user preferences, leading to suboptimal travel experiences. Therefore, there is a pressing need for a comprehensive solution that addresses these challenges and empowers commuters with accurate, real-time, and user-friendly metro travel information.

Solution:

1. Graph-Based Representation:The metro network is represented as a graph, enabling efficient analysis of stations, connections, and travel times.

2. Intuitive User Interface: The system provides an interactive command-line interface, allowing users to input station names, codes, or unique identifiers. It accommodates various input preferences for user convenience.

3. Pathfinding Algorithms: Utilizing Dijkstra's algorithm, the system calculates both the minimum distance and time required for travel between two metro stations. Users can choose between optimizing for distance or time based on their preferences.

4. Station Codes and Interchange Detection:Each station is assigned a unique code, simplifying user input. The system intelligently detects interchange stations and calculates optimal routes considering the number of interchanges, ensuring efficient travel planning.

5. Dynamic Updates: The system supports real-time updates to the metro network. New stations and connections can be seamlessly integrated, ensuring that the system stays current with the evolving metro network.

3.MODULE:

- Graph_M class

4.SOFTWARE REQUIREMENTS SPECIFICATIONS

System configurations :

The software requirement specification can produce at the culmination of the analysis task. The function and performance allocated to software as part of system engineering are refined by established a complete information description, a detailed functional description, a representation of system behavior, and indication of performance and design constrain, appropriate validate criteria, and other information pertinent to requirements.

Software Requirements:

Operating system **:** Windows 11 Ultimate.

Coding Language **:** JAVA.

IDE Used **:** Eclipse(IDE) Professional.

TECHNOLOGIES:-

JAVA LANGUAGE:
Java is a versatile, high-level programming language widely used in a variety of applications, Java is an object-oriented, class-based programming language renowned for its platform independence, reliability, and extensive libraries. Developed by Sun Microsystems (now owned by Oracle), Java has earned its popularity due to its "Write Once, Run Anywhere".

ECLIPSE(IDE):

An Integrated Development Environment (IDE) in Java is a software application

1.Code Editing: Java IDEs offer advanced code editing features such as syntax highlighting, auto-completion, and code formatting to enhance productivity and code quality

2. Project Management: IDEs help developers organize their projects by providing project templates, source code version control integration, and build tools for managing dependencies.

3. Debugging: Debugging tools in an IDE allow developers to identify and fix issues in their Java code. Developers can set breakpoints, step through code, inspect variables, and view stack traces.

4. Compiler and Build Tools: Java IDEs often come with integrated compilers and build tools like Ant, Maven, or Gradle, streamlining the process of building and running Java application

5.HARDWARE REQUIREMENTS

1. Processor-Intel Core I5
2. RAM-8GB

6. CODE

Before getting started , you have to do the following things–

Create a package named: Graph_M class

Create 1 classes in that package

1) graph

****Code:**

```
package assignments;
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 vtces.size();
	}
	public boolean containsVertex(String vname) {
		return 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 queue
stack.addFirst(sp);

// while queue is not empty keep on doing the work
while (!stack.isEmpty()) {

	// remove a pair from queue
	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_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 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);
                }
            }
        }
        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 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 = -1;
                try {
                        choice = Integer.parseInt(inp.readLine());
                } catch (Exception e) {
                        // default will handle
                }
        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();
```

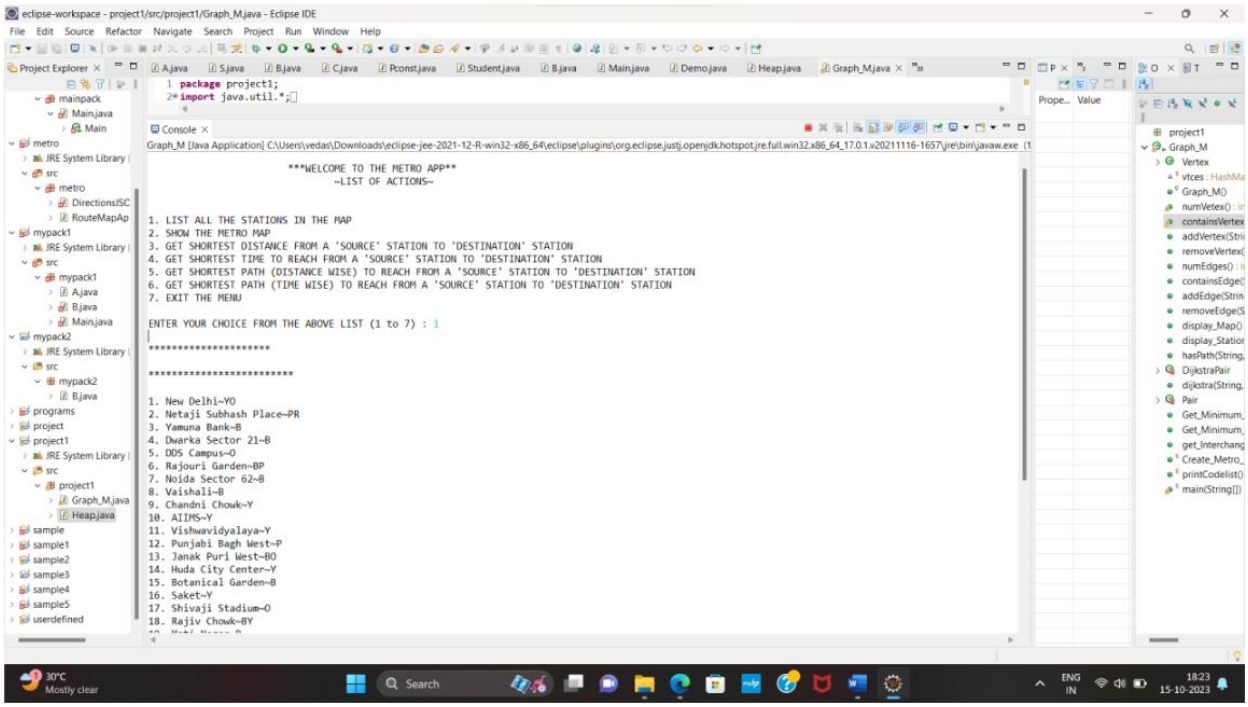
```
System.out.println(
                "\n1. TO ENTER SERIAL NO. OF STATIONS\n2. TO ENTER CODE OF STATIONS\n3. TO ENTER NAME OF STATIONS\n");
System.out.println("ENTER YOUR CHOICE:");
int ch = Integer.parseInt(inp.readLine());
int j;
String st1 = "", st2 = "";
System.out.println("ENTER THE SOURCE AND 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");
```

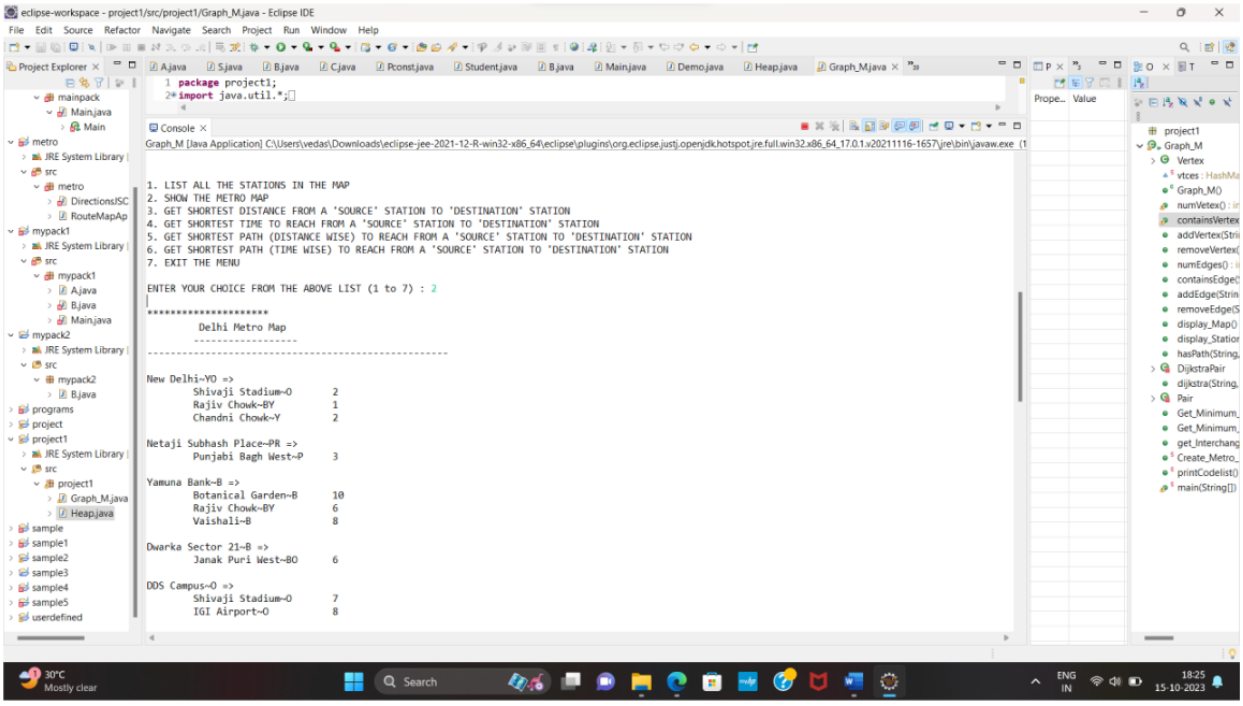
```
                    System.exit(0);
                }
                HashMap<String, Boolean> processed = new HashMap<>();
                if (!g.containsVertex(st1) || !g.containsVertex(st2) || !g.hasPath(st1, st2,
processed))
                    System.out.println("THE INPUTS ARE INVALID");
                else
                    System.out.println("SHORTEST DISTANCE FROM " + st1 + "
TO " + st2 + " IS "
                            + g.dijkstra(st1, st2, false) + "KM\n");
                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");
                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");
```

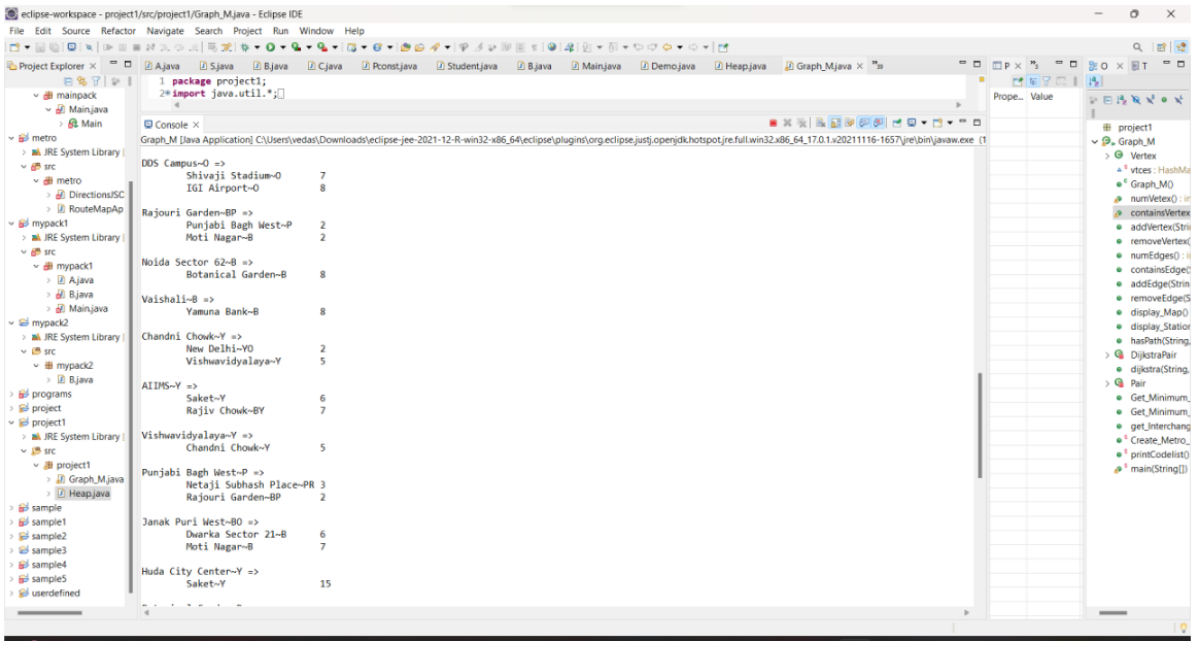
```
                else {
                    ArrayList<String> str = g.get_Interchanges(g.Get_Minimum_Distance(s1, s2));
                    int len = str.size();
                    System.out.println("SOURCE STATION : " + s1);
                    System.out.println("SOURCE STATION : " + s2);
                    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 {
```

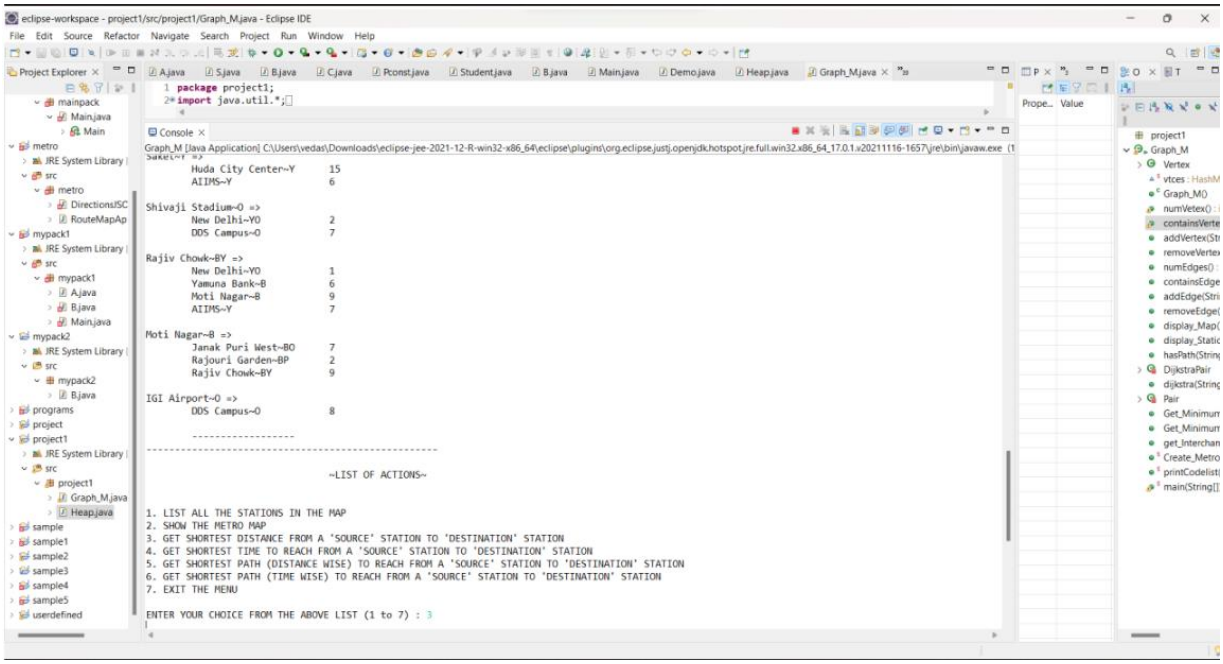
```
                                        ArrayList<String>                    str                    =
g.get_Interchanges(g.Get_Minimum_Time(ss1, ss2));
                                        int len = str.size();
                                        System.out.println("SOURCE STATION : " + ss1);
                                        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;
                        default: // If switch expression does not match with any case,
                                                // default statements are executed by the program.
                                                // No break is needed in the default case
                                System.out.println("Please enter a valid option! ");
                                System.out.println("The options you can choose are from 1 to 6. ");
                                }
                        }
                }
        }
```

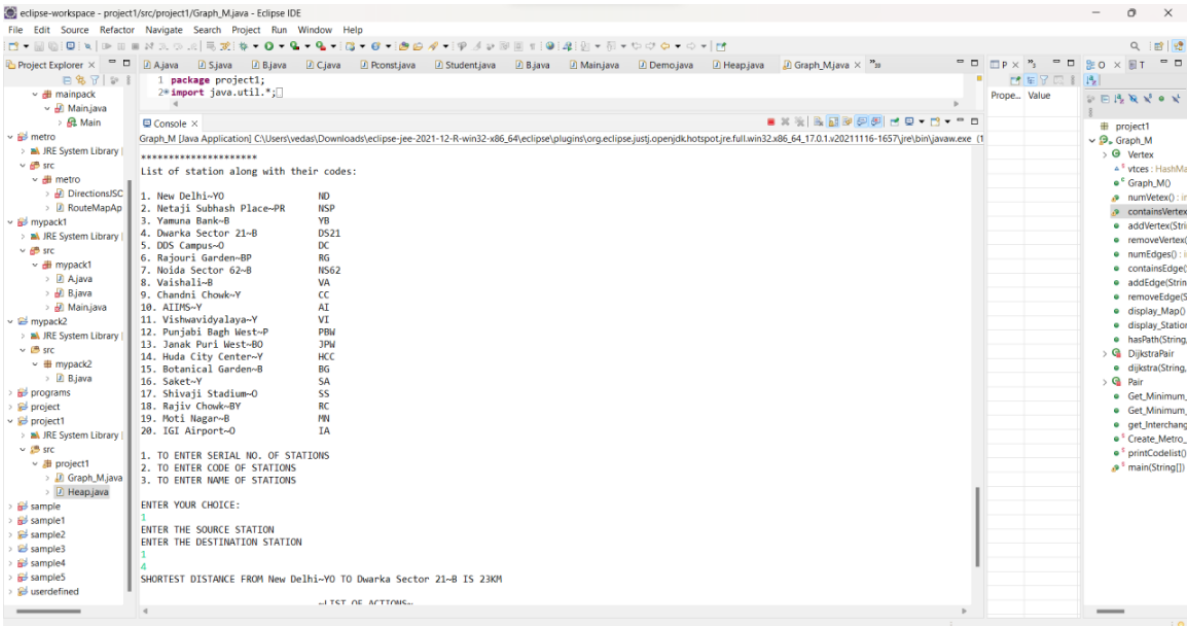
OUTPUT:-

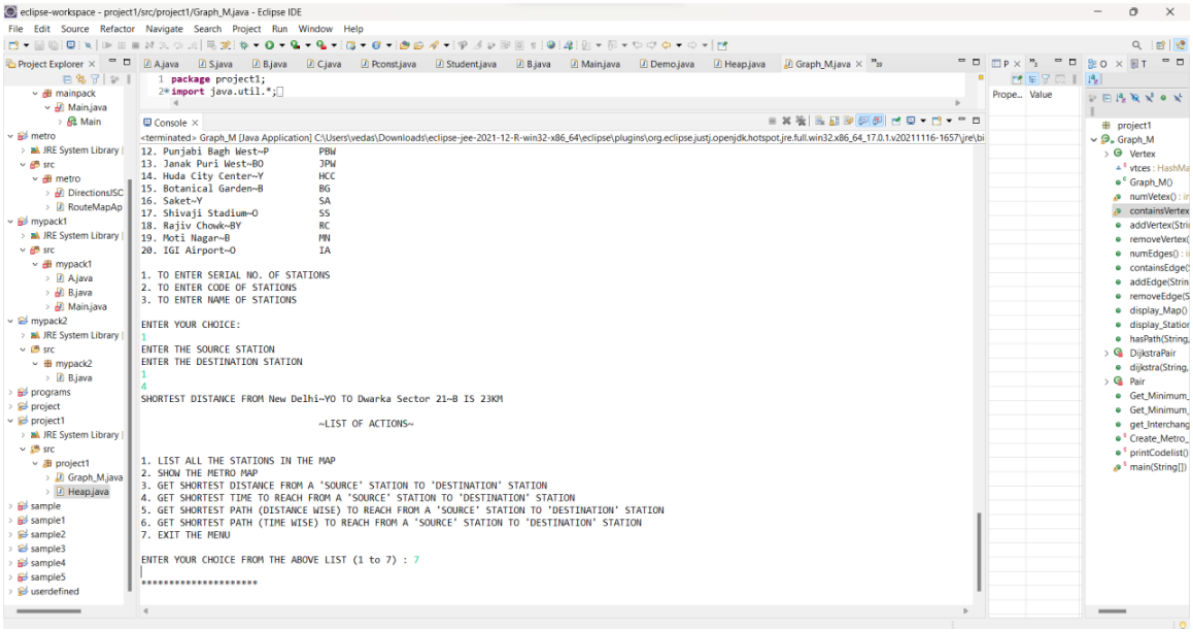

***WELCOME TO THE METRO APP**
~LIST OF ACTIONS~
1. LIST ALL THE STATIONS IN THE MAP
2. SHOW THE METRO MAP
3. GET SHORTEST DISTANCE FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
4. GET SHORTEST TIME TO REACH FROM A 'SOURCE' STATION TO 'DESTINATION' STATION
5. GET SHORTEST PATH (DISTANCE WISE) TO REACH 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) : 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


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


eclipse-workspace - project1/src/project1/Graph_M.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
A.java S.java B.java C.java Pconst.java Student.java B.java Main.java Demo.java Heap.java Graph_M.java
1 package project1;
2 import java.util.*;
Console
Graph_M [Java Application] C:\Users\vedas\Downloads\eclipse-jee-2021-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe
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
Vaishali~B =>
Yamuna Bank~B 8
Chandni Chowk~Y =>
New Delhi~YO 2
Vishwavidyalaya~Y 5
AIIMS~Y =>
Saket~Y 6
Rajiv Chowk~BY 7
Vishwavidyalaya~Y =>
Chandni Chowk~Y 5
Punjabi Bagh West~P =>
Netaji Subhash Place~PR 3
Rajouri Garden~BP 2
Janak Puri West~BO =>
Dwarka Sector 21~B 6
Moti Nagar~B 7
Huda City Center~Y =>
Saket~Y 15


eclipse-workspace - project1/src/project1/Graph_M.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
A.java S.java B.java C.java Pconst.java Student.java B.java Main.java Demo.java Heap.java Graph_M.java
1 package project1;
2 import java.util.*;
Console
Graph_M [Java Application] C:\Users\vedas\Downloads\eclipse-jee-2021-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe
Huda City Center~Y 15
AIIMS~Y 6
Shivaji Stadium~O =>
New Delhi~YO 2
DDS Campus~O 7
Rajiv Chowk~BY =>
New Delhi~YO 1
Yamuna Bank~B 6
Moti Nagar~B 9
AIIMS~Y 7
Moti Nagar~B =>
Janak Puri West~BO 7
Rajouri Garden~BP 2
Rajiv Chowk~BY 9
IGI Airport~O =>
DDS Campus~O 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) : 3


eclipse-workspace - project1/src/project1/Graph_M.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Graph_M [Java Application] C:\Users\vedas\Downloads\eclipse-jee-2021-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bin\javaw.exe
****
List of station along with their codes:
1. New Delhi~YO ND
2. Netaji Subhash Place~PR NSP
3. Yamuna Bank~B YB
4. Dwarka Sector 21~B DS21
5. DDS Campus~O DC
6. Rajouri Garden~BP RG
7. Noida Sector 62~B NS62
8. Vaishali~B VA
9. Chandni Chowk~Y CC
10. AIIMS~Y AI
11. Vishwavidyalaya~Y VI
12. Punjabi Bagh West~P PBW
13. Janak Puri West~BO JPW
14. Huda City Center~Y HCC
15. Botanical Garden~B BG
16. Saket~Y SA
17. Shivaji Stadium~O SS
18. Rajiv Chowk~BY RC
19. Moti Nagar~B MN
20. IGI Airport~O IA
1. TO ENTER SERIAL NO. OF STATIONS
2. TO ENTER CODE OF STATIONS
3. TO ENTER NAME OF STATIONS
ENTER YOUR CHOICE:
1
ENTER THE SOURCE STATION
ENTER THE DESTINATION STATION
1
4
SHORTEST DISTANCE FROM New Delhi~YO TO Dwarka Sector 21~B IS 23KM


eclipse-workspace - project1/src/project1/Graph_M.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
<terminated> Graph_M [Java Application] C:\Users\vedas\Downloads\eclipse-jee-2021-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.1.v20211116-1657\jre\bi
12. Punjabi Bagh West~P PBW
13. Janak Puri West~BO JPW
14. Huda City Center~Y HCC
15. Botanical Garden~B BG
16. Saket~Y SA
17. Shivaji Stadium~O SS
18. Rajiv Chowk~BY RC
19. Moti Nagar~B MN
20. IGI Airport~O IA
1. TO ENTER SERIAL NO. OF STATIONS
2. TO ENTER CODE OF STATIONS
3. TO ENTER NAME OF STATIONS
ENTER YOUR CHOICE:
1
ENTER THE SOURCE STATION
ENTER THE DESTINATION STATION
1
4
SHORTEST DISTANCE FROM New Delhi~YO TO Dwarka Sector 21~B IS 23KM
~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) : 7
****

CONCLUSION

In conclusion, the Automated Metro Network Analysis and Optimization System presents a comprehensive solution to managing and optimizing a metro network efficiently. Through the implementation of graph theory algorithms like Dijkstra's algorithm, the system offers real-time solutions for commuters, ensuring they can find the shortest and most time-efficient routes between stations. Additionally, the system allows for dynamic addition and removal of stations and connections, providing flexibility to adapt to the changing needs of the metro network.

The system's modular design promotes code reusability and maintainability, making it easier to extend or modify in the future. The implementation of various data structures and algorithms demonstrates the versatility of Java programming language in solving complex problems related to network analysis and optimization.

The project not only fulfills the technical requirements but also addresses the practical needs of commuters, offering them a user-friendly interface to interact with the metro network. The inclusion of features like station code mapping and interchange information enhances the user experience, making it intuitive and efficient.

However, like any software system, there is always room for improvement. Future enhancements could focus on incorporating real-time data feeds, allowing the system to adapt to changing traffic conditions and provide dynamic route suggestions. Additionally, further optimization algorithms could be explored to improve the system's efficiency, especially for larger metro networks.

In summary, the Automated Metro Network Analysis and Optimization System showcases the power of algorithmic solutions in addressing real-world transportation challenges. By offering commuters accurate and timely information, the system contributes significantly to improving the overall metro commuting experience. With ongoing improvements and adaptability, this system stands as a robust foundation for future advancements in metro network management and optimization.

THANK YOU….!