A Mini Project Report

on

**METRO MAP**

**DESIGN AND ANALYSIS OF ALGORITHMS LAB**

II B.E IT-A, IV SEMESTER

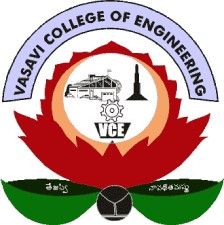
**By**

**Vootla Krishna Sai Srinivas**

**(1602-21-737-026)**

**UNDER THE GUIDANCE OF**

**PRASANNA MA’AM**

****

**Department of INFORMATION TECHNOLOGY**

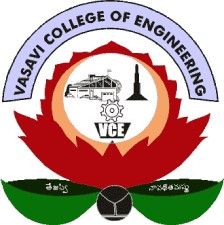
**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Ibrahimbagh, Hyderabad-31**

**(2022-23)**

**Department of Information Technology**

****

**DECLARATION BY THE CANDIDATE**

Vootla Krishna Sai Srinivas bearing hall ticket number, 1602-20-737-026, hereby declare that the project report entitled.

**“Metro Map”** Department of Information Technology, VCE, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Information technology.**

This is a record of bonafide work carried out by me and the results

embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

**Vootla Krishna Sai Srinivas**

**(1602-21-737-026)**

**ACKNOWLEDGEMENT**

The satisfaction and euphoria that accompanies the successful completion if any task would be incomplete without the mention of the people made it possible and whose encouragement and guidance have made our efforts with

success**.**

                  We are indebted to the Internal Guide, **PRASANNA MA’AM** Information Technology, VASAVI  COLLEGE OF ENGINEERING, Ibrahimbagh , Hyd-31 for her support and guidance throughout the project.

                  We are also indebted to the Head of the Department, **Dr. RAM MOHAN RAO, PROFESSOR**, Information Technology, VASAVI COLLEGE OF    ENGINEERING, Ibrahimbagh, hyd-31 for her support and guidance throughout the project.

                 We extend our deep sense of gratitude to the **Principal, S. V. RAMANA RAO** VASAVI COLLEGE OF ENGINEERING, Ibrahimbagh, Hyd-31 permitting us to undertake this project.

**ABSTRACT**

The main objective of this project is to create a replica of the Hyderabad metro map and understand how the shortest path between two metro stations can be found out using Dijkstra’s Algorithms since this map contains many routes(metro lanes) and junction metros (i.e; A metro station which is common to two or more lanes of the metro map) .

IMPLEMENTATION

1. Graph\_H.java

import java.util.\*;

import java.io.\*;

public class Graph\_H

{

public class Vertex

{

HashMap<String, Integer> nbrs = new HashMap<>();

}

static HashMap<String, Vertex> vtces;

public Graph\_H()

{

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 Hyderabad 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;

@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;

stack.addFirst(sp);

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 (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)) {

Pair np = new Pair();

np.vname = nbr;

np.psf = rp.psf + nbr + " ";

np.min\_dis = rp.min\_dis + 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\_H g)

{

g.addVertex("Miyapur~B");

g.addVertex("KPHB~B");

g.addVertex("Erragadda~B");

g.addVertex("Ameerpet~BY");

g.addVertex("Panjagutta~B");

g.addVertex("Osmania Medical College~B");

g.addVertex("MG BS~BO");

g.addVertex("LB Nagar~B");

g.addVertex("Raidurg~Y");

g.addVertex("Madhapur~Y");

g.addVertex("Yusufguda~Y");

g.addVertex("Begumpet~Y");

g.addVertex("Parade Ground~YO");

g.addVertex("Nagole~Y");

g.addVertex("Secunderabad West~O");

g.addVertex("RTC X Roads~O");

g.addVertex("NarayanGuda~O");

g.addVertex("Assembly~BP");

g.addVertex("Stadium~PR");

g.addVertex("Jubilee Hills~P");

g.addEdge("Miyapur~B", "KPHB~B", 8);

g.addEdge("KPHB~B", "Erragadda~B", 10);

g.addEdge("Erragadda~B", "Panjagutta~B", 8);

g.addEdge("Erragadda~B", "Ameerpet~BY", 6);

g.addEdge("Ameerpet~BY", "Osmania Medical College~B", 9);

g.addEdge("Osmania Medical College~B", "MG BS~BO", 7);

g.addEdge("MG BS~BO", "LB Nagar~B", 6);

g.addEdge("Raidurg~Y", "Madhapur~Y", 15);

g.addEdge("Madhapur~Y", "Nagole~Y", 6);

g.addEdge("Nagole~Y", "Ameerpet~BY", 7);

g.addEdge("Ameerpet~BY", "Parade Ground~YO", 1);

g.addEdge("Parade Ground~YO", "Begumpet~Y", 2);

g.addEdge("Begumpet~Y", "Yusufguda~Y", 5);

g.addEdge("Parade Ground~YO", "Secunderabad West~O", 2);

g.addEdge("Secunderabad West~O", "RTC X Roads~O", 7);

g.addEdge("RTC X Roads~O", "NarayanGuda~O", 8);

g.addEdge("Osmania Medical College~B", "Assembly~BP", 2);

g.addEdge("Jubilee Hills~P", "Assembly~BP", 2);

g.addEdge("Jubilee Hills~P", "Stadium~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\_H g = new Graph\_H();

Create\_Metro\_Map(g);

System.out.println("\n\t\t\t\*\*\*\*WELCOME TO THE METRO APP\*\*\*\*\*");

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. ");

}

}

}

}

2. Heap.java

import java.util.ArrayList;

import java.util.HashMap;

public class Heap<T extends Comparable<T>>

{

ArrayList<T> data = new ArrayList<>();

HashMap<T, Integer> map = new HashMap<>();

public void add(T item)

{

data.add(item);

map.put(item, this.data.size() - 1);

upheapify(data.size() - 1);

}

private void upheapify(int ci)

{

int pi = (ci - 1) / 2;

if (isLarger(data.get(ci), data.get(pi)) > 0)

{

swap(pi, ci);

upheapify(pi);

}

}

private void swap(int i, int j)

{

T ith = data.get(i);

T jth = data.get(j);

data.set(i, jth);

data.set(j, ith);

map.put(ith, j);

map.put(jth, i);

}

public void display()

{

System.out.println(data);

}

public int size()

{

return this.data.size();

}

public boolean isEmpty()

{

return this.size() == 0;

}

public T remove()

{

swap(0, this.data.size() - 1);

T rv = this.data.remove(this.data.size() - 1);

downheapify(0);

map.remove(rv);

return rv;

}

private void downheapify(int pi)

{

int lci = 2 \* pi + 1;

int rci = 2 \* pi + 2;

int mini = pi;

if (lci < this.data.size() && isLarger(data.get(lci), data.get(mini)) > 0)

{

mini = lci;

}

if (rci < this.data.size() && isLarger(data.get(rci), data.get(mini)) > 0)

{

mini = rci;

}

if (mini != pi)

{

swap(mini, pi);

downheapify(mini);

}

}

public T get()

{

return this.data.get(0);

}

public int isLarger(T t, T o)

{

return t.compareTo(o);

}

public void updatePriority(T pair)

{

int index = map.get(pair);

upheapify(index);

}

OUTPUTS:

(Compilation of Files)

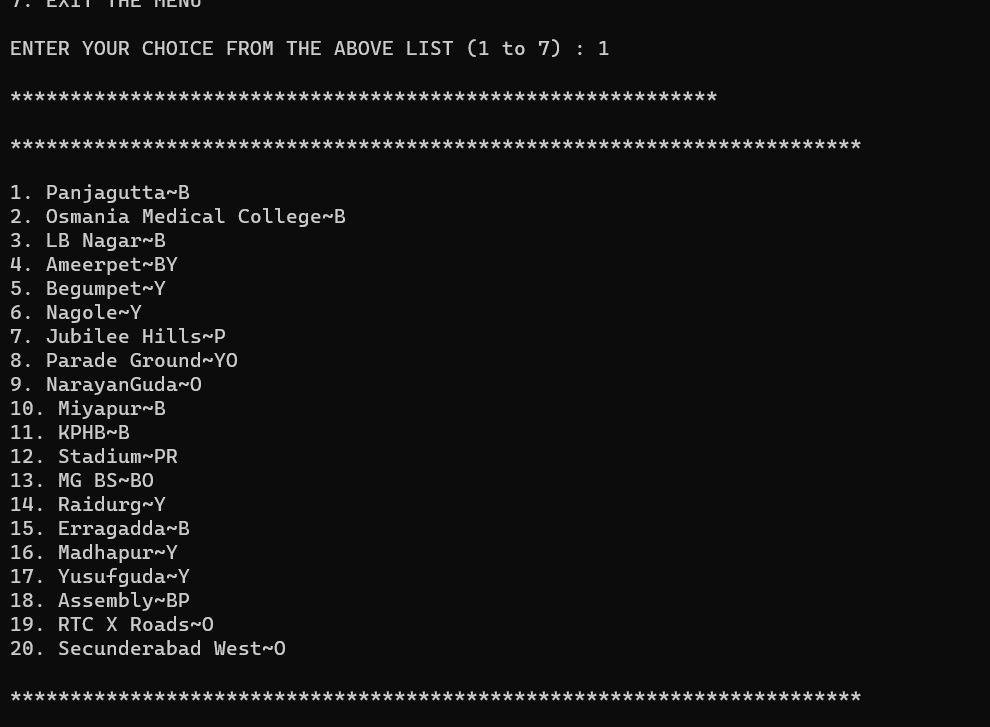
javac Graph\_H.java  
javac Heap.java

java Graph\_H

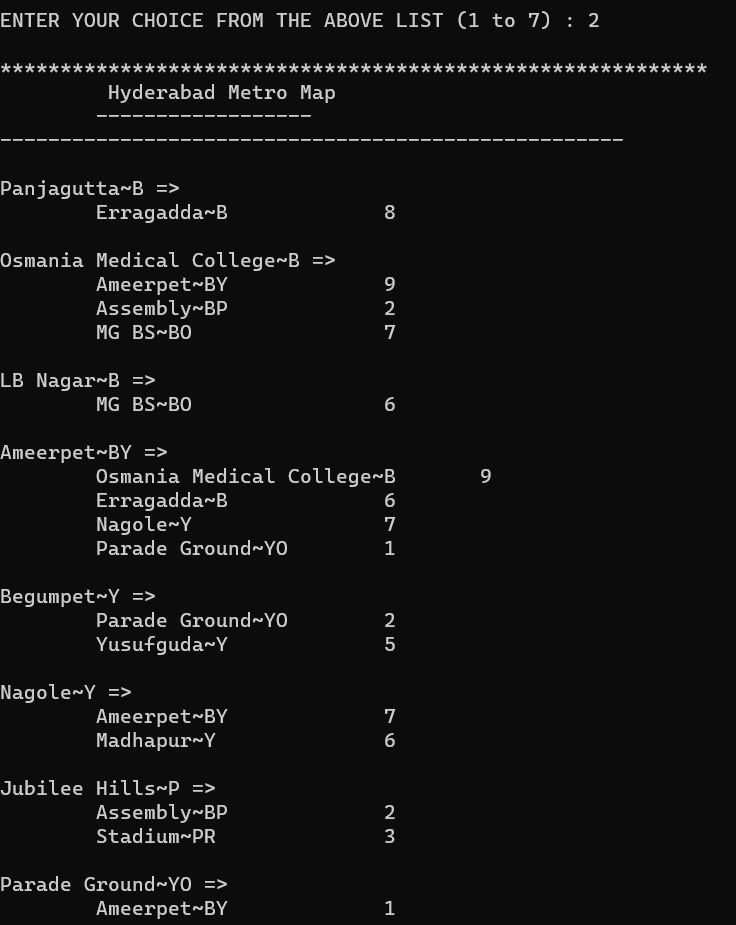
(After Compilation)  
1. Menu Options

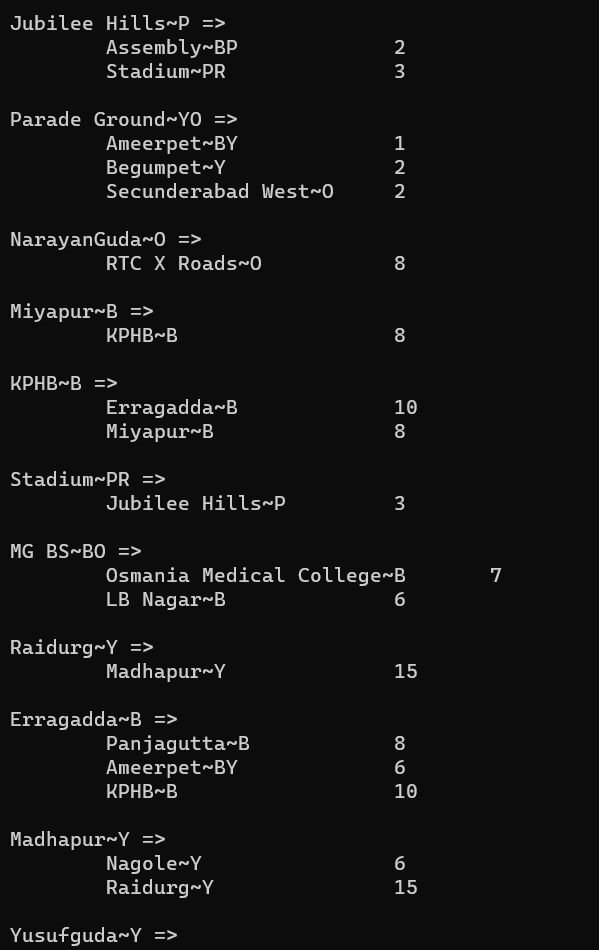


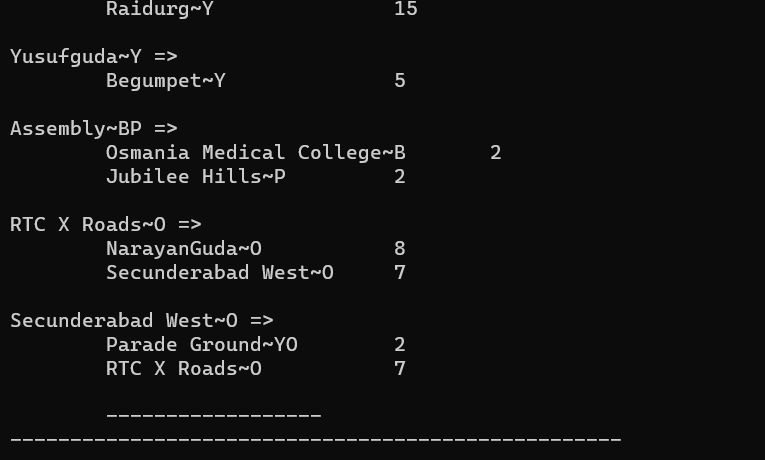
2. By selecting option-1:



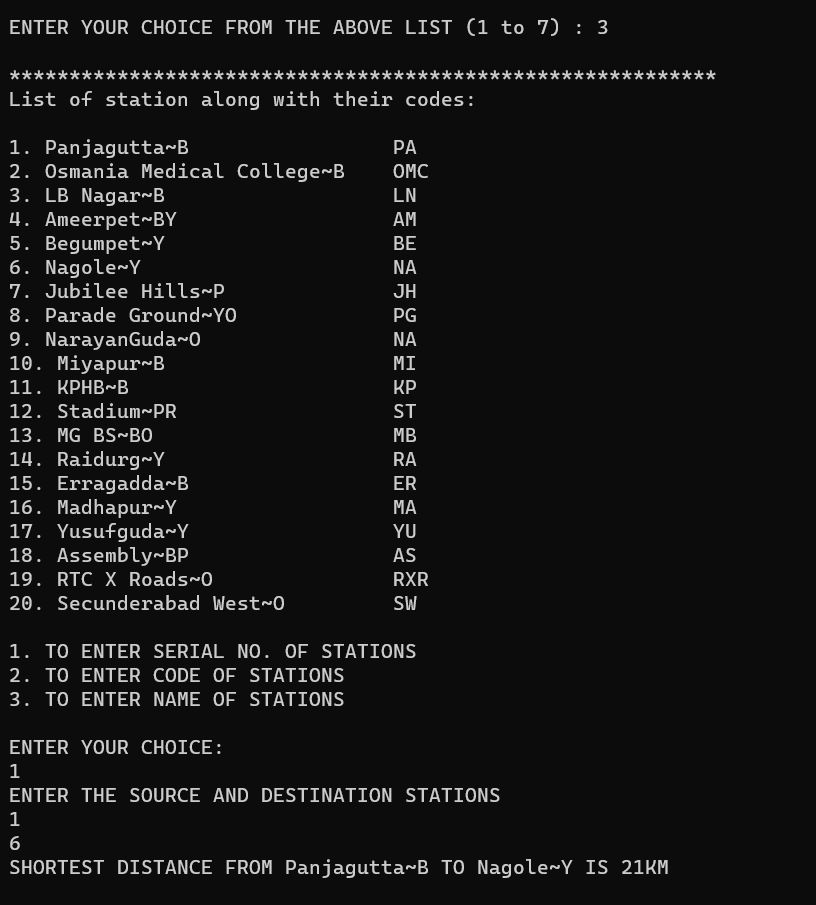
3. By selecting option-2:



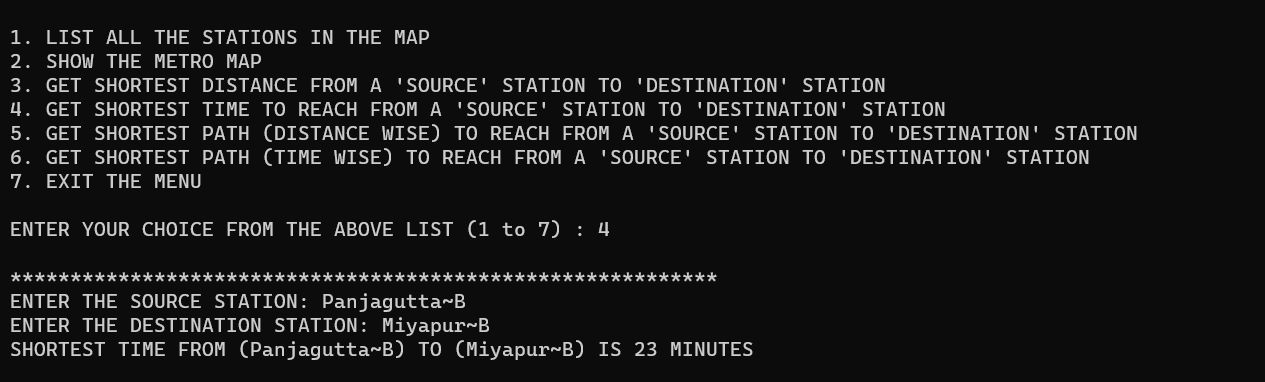




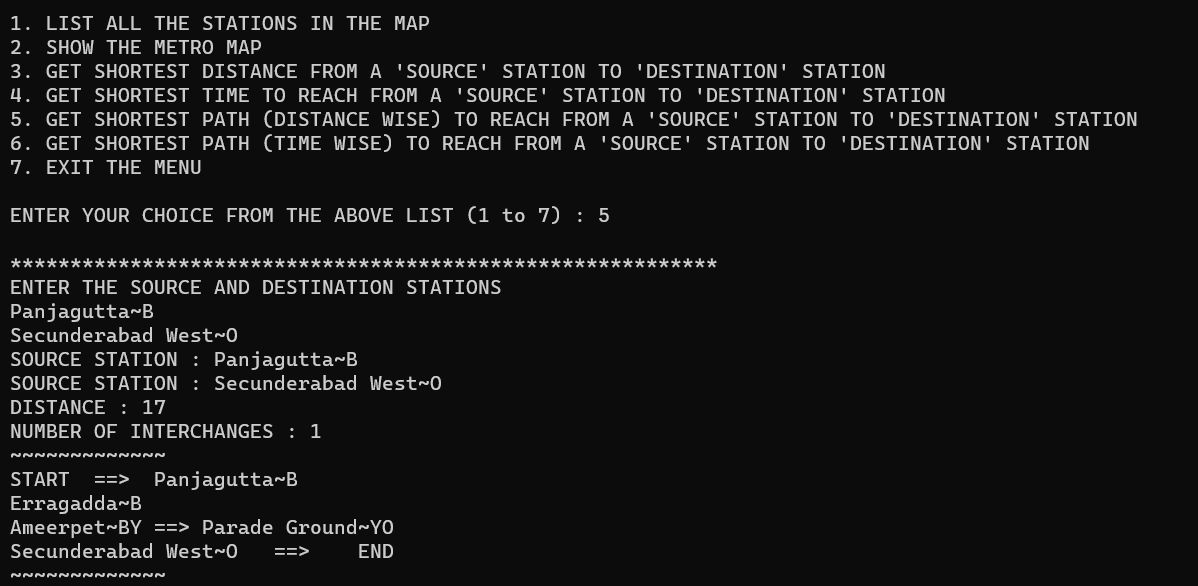
4. By selecting option-3:



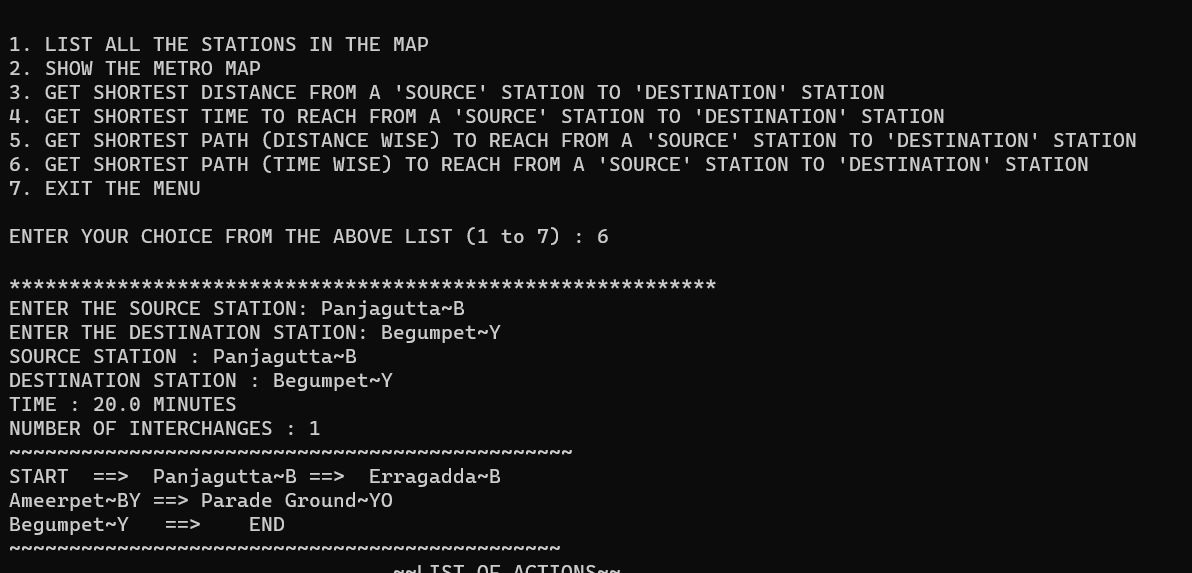
5. By selecting option-4:



6. By selecting option-5:



7. By selecting option-6:



REFERENCES

1. Ellis Horowitz, Sartaj Sahani, Sanguthevar Rajasekaran,

“Fundamentals of computer Algorithms”, Second edition (2008),

Universities Press.

2. Thomas H. Cormen, Leiserson C.E, Rivest.R.L , Stein.C,

Introduction to Algorithm, 2nd

edition (2001), MIT press, USA..

3. <https://ocw.mit.edu/courses/electrical-engineering-and>

computer-science/6-046j-design and-analysis-of-algorithms spring-2 015/index.htm