AMRITA VISHWA VIDYAPEETHAM

Data Structure and Algorithm

19CSE212

PROJECT TITLE : METRO PATH

REPORT

Team Members :

Aman Bhandari (21202)

Aniket Mandal (21203)

Binod Mandal (21208)

Arun Kumar Sah(21266)

SUBMITTED TO :

Dr. R. Bhuvaneshwari

Index

Abstract :

The objective of project "METRO PATH" is to help the people in finding the shortest path between two desired metro stations using Dijkstra's algorithm which involves simplifying the metro network by representing it as a graph with nodes and edges. The nodes represent the metro stations and the edges represent the connections between them. works by starting at the source node and exploring the neighboring nodes, assigning tentative distances and updating them as it finds shorter paths continues until it has visited all nodes

Data Structures Used:

GRAPH: A graph is a common data structure used for representing networks.

MAP: A map data structure is a collection of key-value pairs, where each key is associated with a value.

VECTOR: A vector is a data structure that represents an ordered collection of elements, more dynamic than array

Functions used in the Project:

1. int main( ):

Makes graph of the metro stations and inputs the source and destination Metro station name. Calls the other function of the class Graph which execute the other functionalities.

2. void dijsktraSSSP(string,string,map):

Calculates the shortest distance between the source station and destination station.

3. void DijkstraGetShortestPathTo(string,map):

Finds the stations in between the source and destination lying on the shortest path.

4. void makedotfile():

Uses the data of stations being traversed in the shortest path and creates a dot file which will be used to make the resulting png file.

5. void calcPrice(string,string):

Uses the data of the actual Metro Fares stored in a csv file and stores them into a 2d matrix then finds the fare of the input stations.

6.void split( ):

Used to split the line-by-line tuple of the extracted file into words and store these words in a vector.

7. Void check( ):

Returns true if the extracted station from the file belongs to the stations lying in the shortest path else false

Algorithms Implemented:

Dijkstra's Algorithm:

Dijkstra's algorithm to find the shortest path between a and b. It picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. Mark visited (set to red) when done with neighbors.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by

computer scientist Edsger W. Dijkstra in 1956 and published three years later. For a given source node in the graph, the algorithm finds the shortest path between that node and every other. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined

CODE:

Metro works.cpp

```
#include<bits/stdc++.h>
//#include <graphviz/gvc.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
#include <stdio.h>
#include <sys/wait.h>
#include <stdlib.h>
using namespace std;
template<typename T>
class Graph
{

   map<T, list<pair<T,float> > > m;
public:
   list<T> path;
   void addEdge(T u,T v,float dist,bool bidir=true)
   {
      m[u].push_back(make_pair(v,dist));
      if(bidir)
      {
         m[v].push_back(make_pair(u,dist));
      }
   }

   void DijkstraGetShortestPathTo(T destination, map<T,T> &prev)
   {
```

```
        for( ; destination!=""; destination = prev[destination])
        {
            path.push_back(destination);
        }
        path.reverse();
        cout<<"\t\t\t";
        copy(path.begin(), path.end(), ostream_iterator<T>(cout, "\n\t\t\t"));
    }
    bool printAdj(string,string);
    void dijsktraSSSP(T,map<T,float> &dist, map<T,T> &prev);
    void makedotfile();
    vector<string> split (const string &s, char delim)
    {
        vector<string> result;
        stringstream ss (s);
        string item;

        while (getline (ss, item, delim))
        {
            result.push_back (item);
        }

        return result;
    }
    bool check(string, string);
    void calcPrice(string, string);
};

// FARE CALC FUNCTION STARTS

template<typename T>
void Graph<T>::calcPrice(string srstn,string dstn)
{
    string line;
    int i=0,j=0,k=0,len,last=0;
    string num = "";
    long arr[235][235];
    ifstream infile ("fare.csv");
```

```
if(infile.is_open())
{
   while(getline(infile,line) )
   {

      // Takes complete row
      // cout << line<< '\n';

      k=0,last=0,j=0;
      len=line.length();
      while(k!=len)
      {
         if(line[k]==','||k==len-1)
         {

            //for converting string into number
            arr[i][j]=atof(num.append(line,last,k).c_str());
            //cout<<"new entry -> "<<arr[i][j]<<" endl \n";

            //Emtying string for getting next data
            num="";

            //increasing column number after saving data
            j++;

            if(k!=len-1)
               last=k+1;
         }
         k++;
      }
      //increase row number for array
      i++;
   }
   //close the file
   infile.close();
}
else cout << "Unable to open file";

string line1;
int a,b;
```

```
    a=0;
    b=0;
    ifstream file("stations.txt");
    int f=0;
    while(!file.eof())
    {
        getline(file,line1);
        vector<string> v1=split(line1,'\t');
        if(v1[1]==srstn)
        {
            f++;
            std::istringstream(v1[0]) >> a;

        }
        if(v1[1]==dstn)
        {
            f++;
            std::istringstream(v1[0]) >> b;
        }
        if(f==2)
            break;
    }
    //cout<<"Id of "<<srstn<<" is "<<a<<endl;
    //cout<<"Id of "<<dstn<<" is "<<b<<endl;
    cout<<endl<<"\t\t\t";
    cout<<"--> Fare is: ₹"<<arr[a-1][b-1]<<endl;
}

// FARE CALCULATION ENDS HERE

template<typename T>
bool Graph<T>::check(string src, string dest)
{
    int f=0;
    list <string> :: iterator it;
    for (it = path.begin(); it != path.end(); ++it)
    {
        if(*it==src)
        {
            f++;
```

```
        }
        else if(*it==dest)
        {
            f++;
        }
    }
    if(f==2)
        return true;
    else
        return false;
}



template<typename T>
void Graph<T>::makedotfile()
{
    string a,b,clr;
    string labl;
    int f;
    f=0;
    clr="red";
    char filename[] = "finalmap.dot";
    string delimiter=",";
    ofstream fout(filename);
    fout<<"graph G {"<<endl;
    fout<<"node [shape=rect,dpi=600] margin=0.75"<<endl;
    fout<<"\n//"<<clr<<endl;
    string x;
    ifstream file("data.txt");
    while (!file.eof())
    {
        getline(file,x);
        vector<string> v = split(x,',');
        a=v[0];
        b=v[1];
        labl=v[2];
        if(f==1)
        {
            fout<<"\n//"<<clr<<endl;
```

```
            f=0;
        }

        if(check(a,b)==true)
            fout<<"\""<<a<<"\""<<" -- "<<"\""<<b<<"\""<<"
"<<"[label=\""<<labl<<"\",color="<<clr<<" "<<",penwidth=\"8\"];"<<endl;
        else
            fout<<"\""<<a<<"\""<<" -- "<<"\""<<b<<"\""<<"
"<<"[label=\""<<labl<<"\",color="<<clr<<" "<<",penwidth=\"2\"];"<<endl;
        if(a=="Seelampur" && f==0)
        {
            f=1;
            clr="blue";
        }
        else if(a=="Golf Course" && f==0)
        {
            f=1;
            clr="green";
        }
        else if(a=="Sant Guru Ram Singh Marg" && f==0)
        {
            f=1;
            clr="violet";
        }
        else if(a=="JL Nehru Stadium" && f==0)
        {
            f=1;
            clr="yellow";
        }
    }
    fout<<"}";
}

template<typename T>
bool Graph<T>::printAdj(string src,string dest)
{
    int f;f=0;
    //Let try to print the adj list
    //Iterate over all the key value pairs in the map
    for(auto j:m)
```

```
    {
    if(j.first==src)
        f++;
        //Iterater over the list of cities
        for(auto l:j.second)
        {

            if(l.first==dest)
            {
                f++;
            }
        }
    }
    if(f>1)
        return true;
    else
        return false;

}

template<typename T>
void Graph<T>::dijsktraSSSP(T src, map<T,float> &dist, map<T,T> &prev)
{

    set<pair<float, T> > s;
    //Set all distance to infinity
    prev.clear();
    for(auto j:m)
    {
        dist[j.first] = numeric_limits<float>::infinity();
        prev[j.first] = "";
    }

    //Make a set to find a out node with the minimum distance

    dist[src] = 0;
    s.insert(make_pair(0,src));

    while(!s.empty())
    {
```

```
        //Find the pair at the front.
        auto p =    *(s.begin());
        T node = p.second; // NODE

        float nodeDist = p.first; //NODEDIST
        s.erase(s.begin());
        //Iterate over neighbours/children of the current node
        for(auto childPair: m[node])
        {
           T dest = childPair.first;
           float weight = childPair.second;
           float distance_through_node = nodeDist + childPair.second;

           if(distance_through_node < dist[childPair.first])
           {
              //In the set updation of a particular is not possible
              // we have to remove the old pair, and insert the new pair to simulation
updation

              auto f = s.find( make_pair(dist[dest],dest));
              if(f!=s.end())
              {
                 s.erase(f);
              }
              //Insert the new pair
              dist[dest] = distance_through_node;
              prev[dest] = node;
              s.insert(make_pair(dist[dest],dest));
           }
        }
     }
     //Lets print distance to all other node from src
     /*for(auto d:dist)
     {
        cout<<d.first<<",is located at distance of  "<<d.second<<endl;
     }*/
}

int main()
```

```
{   system("notify-send -t 5000 -i /home/nightwarrior-
xxx/Documents/MetroWorksDS/train1.png \"Metro Works\"");
    system("gnome-terminal -x sh -c \"fim --autowindow graph.png\"");
    system("clear");
    //system("printf '\e[45;5;196m'");
    //system("echo  \"\e[96m\"");
    system("sl -alfe");
    system("clear");

    cout<<"\t\t"<<" __  __      _                __          __          _ "<<endl;
    cout<<"\t\t"<<"|  \\/  |___| |_ _ __ ___    \\ \\        / /__  _ __| | _____"<<endl;
    cout<<"\t\t"<<"| |\\/| |/ _ \\ __| '__/ _ \\   \\ \\ /\\ / / _ \\| '__| |/ / __|"<<endl;
    cout<<"\t\t"<<"| |  | |  __/ |_| | | (_) |   \\ V  V / (_) | |  |   <\\__ \\ " <<endl;
    cout<<"\t\t"<<"|_|  |_|\\___|\\__|_|  \\___/     \\_/\\_/ \\___/|_|  |_|\\_\\___/"<<endl;
    system("echo  \"\e[96m\"");
    //system("figlet Metro Works");
    string source, destination;
    Graph<string> Metro;
    //red
    Metro.addEdge("Rithala","Netaji Subhash Place",5.2);
    Metro.addEdge("Netaji Subhash Place","Keshav Puram",1.2);
    Metro.addEdge("Keshav Puram","Kanhaiya Nagar",0.8);
    Metro.addEdge("Kanhaiya Nagar","Inderlok",1.2);
    Metro.addEdge("Inderlok","Shastri Nagar",1.2);
    Metro.addEdge("Shastri Nagar","Pratap Nagar",1.7);
    Metro.addEdge("Pratap Nagar","Pulbangash",0.8);
    Metro.addEdge("Pulbangash","Tis Hazari",0.9);
    Metro.addEdge("Tis Hazari","Kashmere Gate",1.1);
    Metro.addEdge("Kashmere Gate","Shastri Park",2.2);
    Metro.addEdge("Shastri Park","Seelampur",1.6);
    Metro.addEdge("Seelampur","Welcome",1.1);
    //blue
    Metro.addEdge("Rajouri Garden","Ramesh Nagar",1);
    Metro.addEdge("Ramesh Nagar","Moti Nagar",1.2);
    Metro.addEdge("Moti Nagar","Kirti Nagar",1);
    Metro.addEdge("Kirti Nagar","Shadipur",0.7);
    Metro.addEdge("Shadipur","Patel Nagar",1.3);
    Metro.addEdge("Patel Nagar","Rajender Place",0.9);
    Metro.addEdge("Rajender Place","Karol Bagh",1);
    Metro.addEdge("Karol Bagh","Rajiv Chowk",3.4);
```

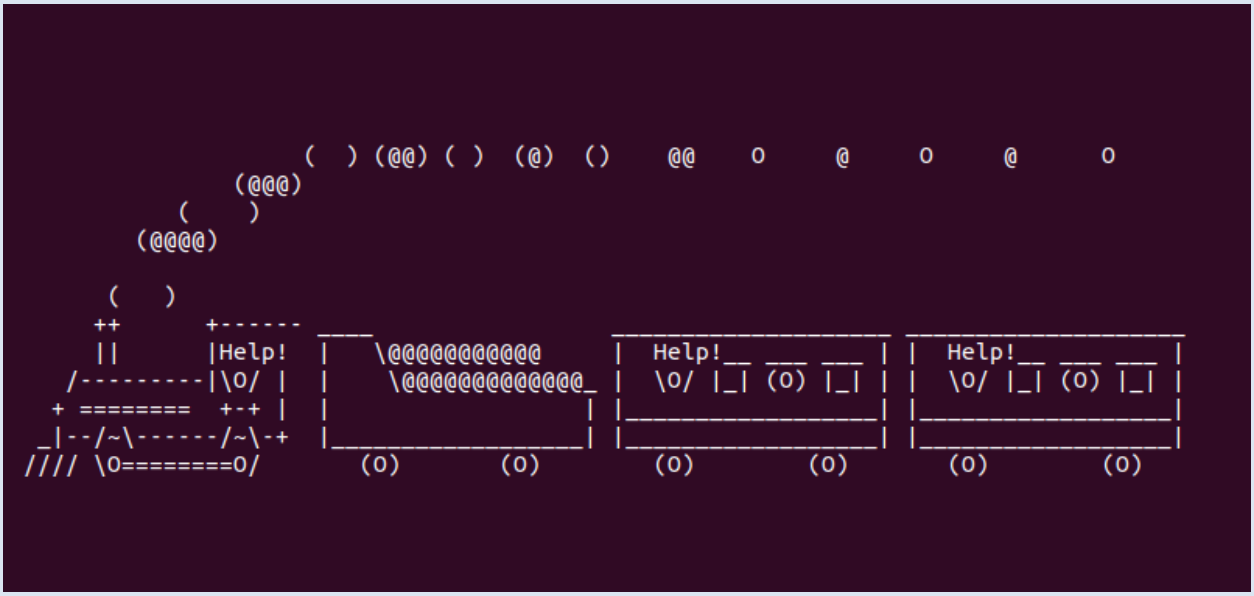
```
Metro.addEdge("Rajiv Chowk","Barakhamba Road",0.7);
Metro.addEdge("Barakhamba Road","Mandi House",1);
Metro.addEdge("Mandi House","Pragati Maiden",0.8);
Metro.addEdge("Pragati Maiden","Inderprastha",0.8);
Metro.addEdge("Inderprastha","Yamuna Bank",1.8);
Metro.addEdge("Yamuna Bank","Laxmi Nagar",1.3);
Metro.addEdge("Laxmi Nagar","Nirman Vihar",1.1);
Metro.addEdge("Nirman Vihar","Preet Vihar",1.0);
Metro.addEdge("Preet Vihar","Karkar Duma",1.2);
Metro.addEdge("Karkar Duma","Anand Vihar",1.1);
Metro.addEdge("Anand Vihar","Kaushambi",0.8);
Metro.addEdge("Kaushambi","Vaishali",1.6);
Metro.addEdge("Yamuna Bank","Akshardham",1.3);
Metro.addEdge("Akshardham","Mayur Vihar Phase-1",1.8);
Metro.addEdge("Mayur Vihar Phase-1","Mayur Vihar Extention",1.2);
Metro.addEdge("Mayur Vihar Extention","New Ashok Nagar",0.9);
Metro.addEdge("New Ashok Nagar","Noida Sector-15",1.0);
Metro.addEdge("Noida Sector-15","Noida Sector-16",1.1);
Metro.addEdge("Noida Sector-16","Noida Sector-18",1.1);
Metro.addEdge("Noida Sector-18","Botanical Garden",1.1);
Metro.addEdge("Botanical Garden","Golf Course",1.2);
Metro.addEdge("Golf Course","Noida City Center",1.3);
//green
Metro.addEdge("Madipur","Shivaji Park",1.1);
Metro.addEdge("Shivaji Park","Punjabi Bagh",1.6);
Metro.addEdge("Punjabi Bagh","Ashok Park",0.9);
Metro.addEdge("Ashok Park","Inderlok",1.4);
Metro.addEdge("Ashok Park","Sant Guru Ram Singh Marg",1.1);
Metro.addEdge("Sant Guru Ram Singh Marg","Kirti Nagar",1);
Metro.addEdge("Kashmere Gate","Lal Qila",1.5);
Metro.addEdge("Lal Qila","Jama Masjid",0.8);
Metro.addEdge("Jama Masjid","Delhi Gate",1.4);
Metro.addEdge("Delhi Gate","ITO",1.3);
Metro.addEdge("ITO","Mandi House",0.8);
Metro.addEdge("Mandi House","Janptah",1.4);
Metro.addEdge("Janptah","Central Secretariat",1.3);
Metro.addEdge("Central Secretariat","Khan Market",2.1);
Metro.addEdge("Khan Market","JL Nehru Stadium",1.4);
Metro.addEdge("JL Nehru Stadium","Jangpura",0.9);
//yellow
```

```
Metro.addEdge("Vishwavidyalaya","Vidhan Sabha",1);
Metro.addEdge("Vidhan Sabha","Civil Lines",1.3);
Metro.addEdge("Civil Lines","Kashmere Gate",1.1);
Metro.addEdge("Kashmere Gate","Chandni Chowk",1.1);
Metro.addEdge("Chandni Chowk","Chawri Bazar",1);
Metro.addEdge("Chawri Bazar","New Delhi",0.8);
Metro.addEdge("New Delhi","Rajiv Chowk",1.1);
Metro.addEdge("Rajiv Chowk","Patel Chowk",1.3);
Metro.addEdge("Patel Chowk","Central Secretariat",0.9);
Metro.addEdge("Central Secretariat","Udyog Bhawan",0.3);
Metro.addEdge("Udyog Bhawan","Lok Kalyan Marg",1.6);
Metro.addEdge("Lok Kalyan Marg","Jor Bagh",1.2);
Metro.addEdge("Samaypur Badli","Rohini Sector - 18",0.8);
Metro.addEdge("Rohini Sector - 18","Haiderpur Badli Mor",1.3);
Metro.addEdge("Haiderpur Badli Mor","Jahangirpuri",1.3);
Metro.addEdge("Jahangirpuri","Adarsh Nagar",1.3);
Metro.addEdge("Adarsh Nagar","Azadpur",1.5);
Metro.addEdge("Azadpur","Model Town",1.4);
Metro.addEdge("Model Town","GTB Nagar",1.4);
Metro.addEdge("GTB Nagar","Vishwavidyalaya",0.8);
Metro.addEdge("Jor Bagh","INA",1.3);
Metro.addEdge("INA","AIIMS",0.8);
Metro.addEdge("AIIMS","Green Park",1.0);
Metro.addEdge("Green Park","Hauz Khas",1.8);
Metro.addEdge("Hauz Khas","Malviya Nagar",1.7);
Metro.addEdge("Malviya Nagar","Saket",0.9);
Metro.addEdge("Saket","Qutab Minar",1.7);
Metro.addEdge("Qutab Minar","Chhattarpur",1.3);
Metro.addEdge("Chhattarpur","Sultanpur",1.6);
Metro.addEdge("Sultanpur","Ghitorni",1.3);
Metro.addEdge("Ghitorni","Arjan Garh",2.7);
Metro.addEdge("Arjan Garh","Guru Dronacharya",2.3);
Metro.addEdge("Guru Dronacharya","Sikandarpur",1.0);
Metro.addEdge("Sikandarpur","MG Road",1.2);
Metro.addEdge("MG Road","IFFCO Chowk",1.1);
Metro.addEdge("IFFCO Chowk","Huda City Centre",1.5);
map<string,float> dist;
map<string,string> prev;
string sourcestn, deststn;
cout<<endl<<endl<<endl;
```

```
    cout<<"\t\t";
    cout<<"Enter source station in capital case: ";
    getline(cin,sourcestn);
    //system("echo \"\e[92m\"");
    cout<<endl;
    cout<<"\t\t";
    cout<<"Enter destination station in capital case: ";
    //system("echo \"\e[96m\"");
    getline(cin,deststn);
    bool res=Metro.printAdj(sourcestn,deststn);
    if(res==false)
    {
      system("zenity --error --title \"Error Occured\" --text='Invalid Station
Entered'");
      system("clear");
      return 0;
    }

    //system("echo \"\e[92m\"");
    Metro.dijsktraSSSP(sourcestn, dist, prev);
    //system("echo \"\e[96m\"");
    cout<<endl<<"\t\t";
    cout<<"Distance from "<<sourcestn<<" to "<<deststn<<" - "<<dist[deststn]<<"
Kms"<<endl;
    cout<<endl<<"\t\tPath: "<<endl;
    Metro.DijkstraGetShortestPathTo(deststn,prev);
    Metro.makedotfile();
    Metro.calcPrice(sourcestn,deststn);
    system("dot -Tpng finalmap.dot -o path.png");
    system("gnome-terminal  -- sh -c \"fim --autowindow path.png\"");
    return 0;
}
```

OUTPUT:



After Metro moved out:

Shortest path: