DSA Project

Quickest Route to your Destination
on Delhi Metro

Submitted by:

Chinmay Rahul Ghan (19102063)

Aryan Chauhan (19102071)

Hariom Kalra (19102082)

Suman Saurav (19102083)

Submitted to:

Mrs. Deepti

Introduction

The objective of project “Metro Works” is to help the people in finding the shortest path between two desired metro stations & calculate the corresponding fare.

In Delhi NCR, metro is one of the major modes of transportation which on average carry 2.3 million people dailyaccording to DMRC. One of the problems which many people including us face on daily basis is finding shortest and cheapest route between two stations when there is multiple lines and routes to consider. So, we thought of solving this problem by showing shortest route and fair between two metro stations.

We’ll be using different data structures and algorithm which
is mentioned in below pages to find the shortest route betweentwo desired metro stations where users have to enter station name which he/she have to board and destination station
which he/she have to go and we’ll show shortest route and
price.

BACKGROUND

Networks are necessary for the movement of people, transportation of goods, communicate informationand control of the flow of matter and energy. Network application is quite vast. Phenomena that are represented and analyzed as networks are roads, railways, cables, and pipelines. In networking, the cost,time, and complex nature of network increases in different kinds of network-based systems, e.g. Television cable networks, Telephone networks, Electricity supply networks, Gas pipe network and water supply system. Therefore, the cost, time, and complexity of network are considered greatly in solving networking problems. A graph is a mathematical abstraction that is useful for solving differentnetworking problems. Finding the shortest paths plays an important role in solving network based systems. In graph theory, a number of algorithms can be applied for finding the shortest path in a graph-based network system. This reduces the complexity of the network path, the cost, and the time to build and maintain the network-based systems. In recent times, planning efficient routes is very essential for business and industry with applications as varied as product distribution. It is essential for products or services to be delivered on time at the best price using the shortest available route. The shortest route network model is an efficient route that can be used in planning. This network model is applied in telecommunications and transportation planning. The shortest path problem involves finding the shortest possible path or route from a starting point to a final point.

Networks are used in general to represent shortest path problem. A graph which is used to solve such problems contains sets of vertices and edges. Pairs of vertices are connected by edges, while movement from one vertex to other vertices can be done along the edges. A graph can be directed or undirected depending on the movement along the edges, either walking on both sides or on only one side. Lengths of edges are often called weights which are normally used to calculate the shortest path from a particular point to another point. In the real world, the graph theory can be applied to different scenarios. A practical example is map representation using a graph, where vertices and edges represent cities and routes that connect the cities respectively. One-way routes are directed graphs, while routes that are not one-way are undirected. There are different types of algorithms that are used to solve shortest path problems. However, only the Dijkstra’s algorithm will be discussed in this project.

Shortest Path problems are inevitable in road network applications such as city emergency handling and drive guiding system, in where the optimal routings have to be found. As the traffic condition among a city changes from time to time and there are usually a huge amounts of requests occur at any moment, it needs to quickly find the solution.

Therefore, the efficiency of the algorithm is very important [1, 3 and5]. Some approaches take advantage of preprocessing that compute results before demanding. These results are saved in memory and could be used directly when a new request comes up. This can be inapplicable if the devices have limited memory and external storage. This project aims only at investigate the single source shortest path problems and intends to obtain some general conclusions by examining three approaches, Dijkstra's shortest path algorithm, Restricted search algorithm and A* algorithm. To verify the three algorithms, a program was developed under Microsoft Visual C++ environment. The three algorithms were implemented and visually demonstrated. The road network example is a graph data file containing partial transportation data of the Ottawa city.

APPLICATIONS

Shortest path problems form the foundation of an entire class of optimization problems that can be solved by a technique called *column generation*. Examples include vehicle routing problem, survivable network design problem, amongst others. In such problems there is a *master problem* (usually a linear program) in which each column represents a path(think of a path in a vehicle routing problem as one candidate route that a vehicle can take,think of a path in a network design problem as a possible route over which a commodity can be sent between a source and a destination). The master problem is repeatedly solved. Each time, using the metrics of the solution, a separate problem (called the column-generation subproblem) is solved. This problem turns out to be a shortest path problem (usually with side constraints or negative arc lengths rendering the problem NP-Hard). If anew useful path is obtained, it is added to the original master problem which is now re-solved over a larger subset of paths leading to increasingly better (lower cost, usually) solutions.

The shortest path is often used in SNA (Social Network Analysis) to calculate betweennesscentrality among others. Usually, people with the strongest bonds tend to communicate through the shortest path. Knowing in a graph the shortest path between people (nodes) can let you know hidden strong bonds.

Shortest path algorithms are applied to automatically find directions between physical locations,such as driving directions on web mapping websites like MapQuest or Google Maps. For this application fast specialized algorithms are available.

If one represents a nondeterministic abstract machine as a graph where vertices describe states andedges describe possible transitions, shortest path algorithms can be used to find an optimal sequence of choices to reach a certain goal state, or to establish lower bounds on the time needed to reach a given state. For example, if vertices represent the states of a puzzle like a Rubik's Cube and each directed edge corresponds to a single move or turn, shortest path algorithms can beused to find a solution that uses the minimum possible number of moves.

In a networking or telecommunications mindset, this shortest path problem is sometimes called the min-delay path problem and usually tied with a widest path problem. For example, the algorithm mayseek the shortest (min-delay) widest path, or widest shortest (min-delay) path.

A more lighthearted application is the games of "six degrees of separation" that try to find theshortest path in graphs like movie stars appearing in the same film.

Other applications, often studied in operations research, include plant and facilitylayout, robotics, transportation, and VLSI design.

A road network can be considered as a graph with positive weights. The nodes represent road junctions and each edge of the graph is associated with a road segment between two junctions. The weight of an edge may correspond to the length of the associated road segment, the time needed to traverse the segment, or the cost of traversing the segment. Using directed edges, it is also possible to model one-way streets. Such graphs are special in the sense that some edges are more important than others for long-distance travel (e.g., highways). This property has been formalized using the notion of highway dimension. There are a great number of algorithms that exploit this property and are therefore able to compute the shortest path a lot quicker than would be possible on general graphs.

All of these algorithms work in two phases. In the first phase, the graph is preprocessed without knowing the source or target node. The second phase is the query phase. In this phase, source and target node are known. The idea is that the road network is static, so the preprocessing phase can be done once and used for a large number of queries on the same road network.

The algorithm with the fastest known query time is called hub labeling and is able to compute shortest path on the road networks of Europe or the US in a fraction of a microsecond.

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.

Data Structure Used:

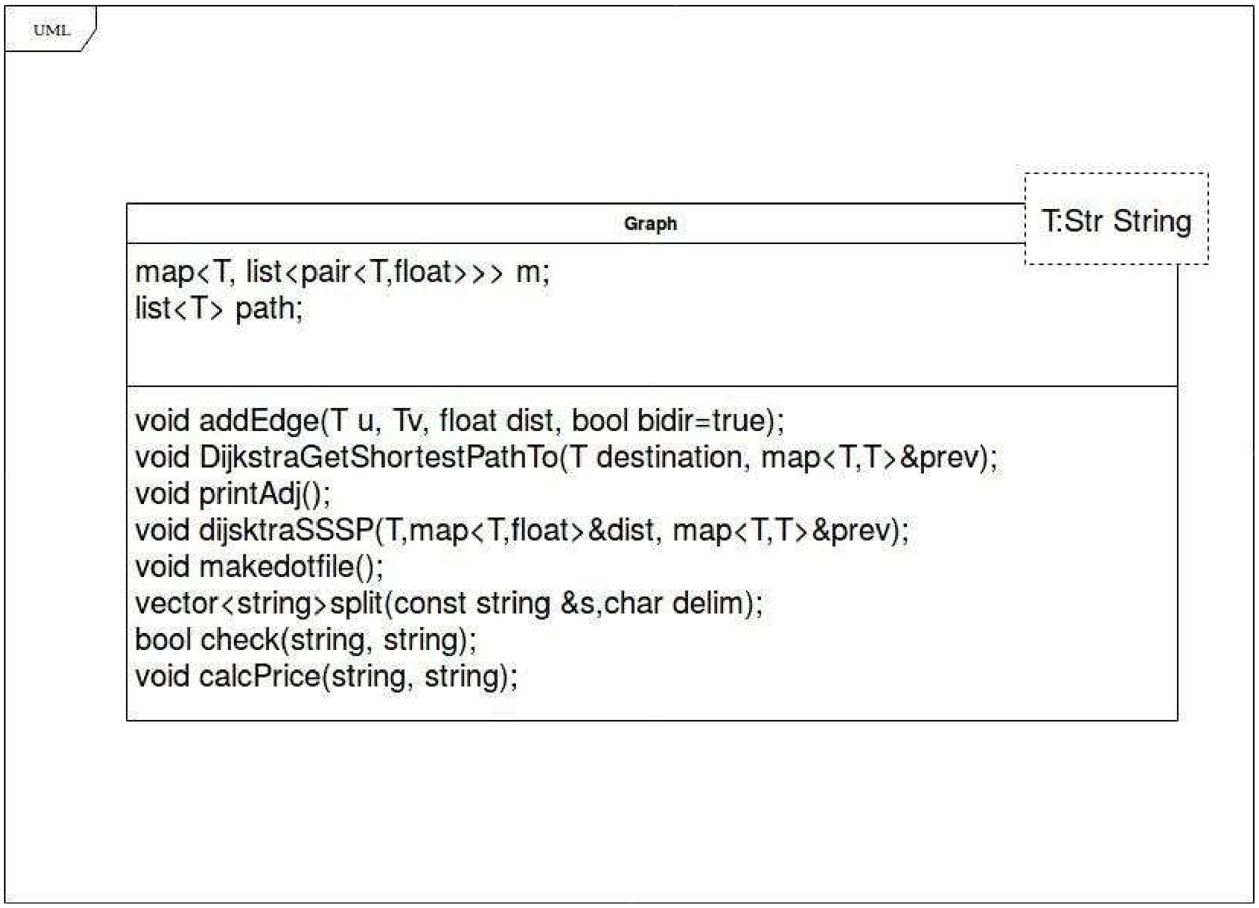
- Graphs: A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like LinkedIn, Facebook.

- Standard Template Library Containers: A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows a great flexibility in the types supported as elements. The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers).

Algorithms Used:

Dijkstra Algorithm: 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.

The algorithm exists in many variants. Dijkstra's original algorithm found the shortest path between two given nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.


UML
Graph
T:Str String
map<T, list<pair<T,float>>> m;
list<T> path;
void addEdge(T u, Tv, float dist, bool bidir=true);
void DijkstraGetShortestPathTo(T destination, map<T,T>&prev);
void printAdj();
void dijsktraSSSP(T,map<T,float>&dist, map<T,T>&prev);
void makedotfile();
vector<string>split(const string &s,char delim);
bool check(string, string);
void calcPrice(string, string);

CODE

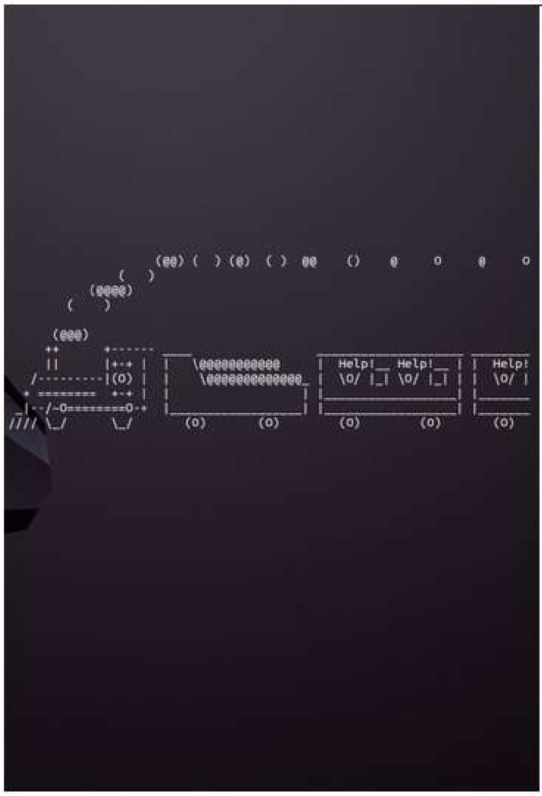
```
//run using g++ -std=c++11 Dijkstras.cpp
#include<bits/stdc++.h>
//#include <graphviz/gvc.h>
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
#include <stdio.h>
//#include <sys/wait.h>
#include <stdlib.h>
using namespace std;
template<typename T>
class Graph
{

    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)
```

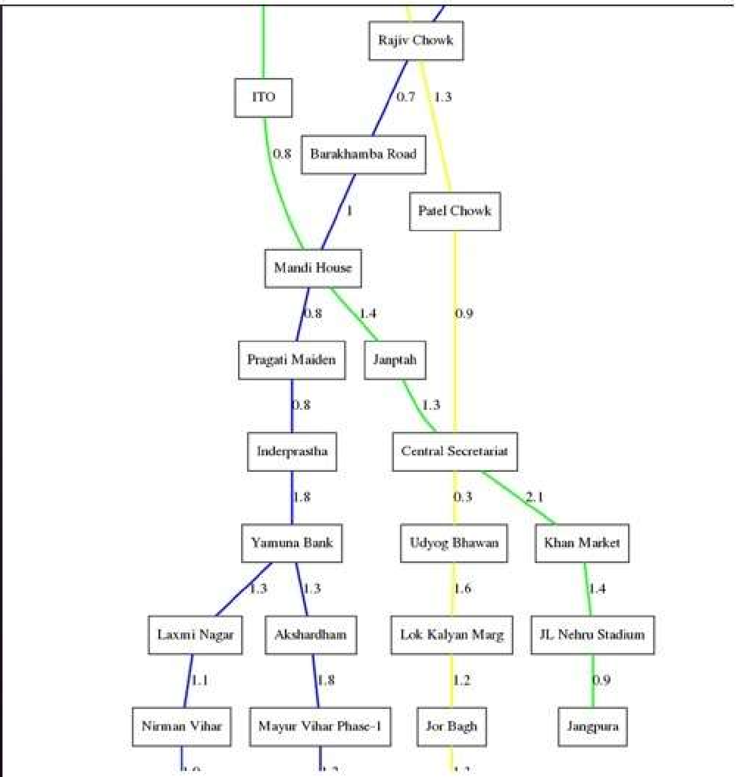
```cpp
        {
            m[v].push_back(make_pair(u,dist));
        }
    }

    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;
```

```cpp
    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));
```

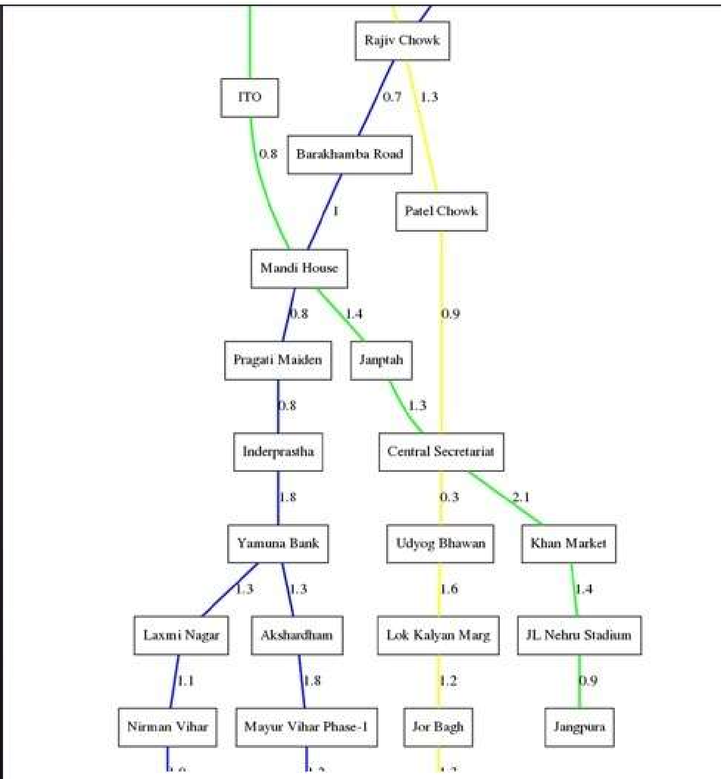
```cpp
    while(!s.empty())
    {

        //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;
}
```

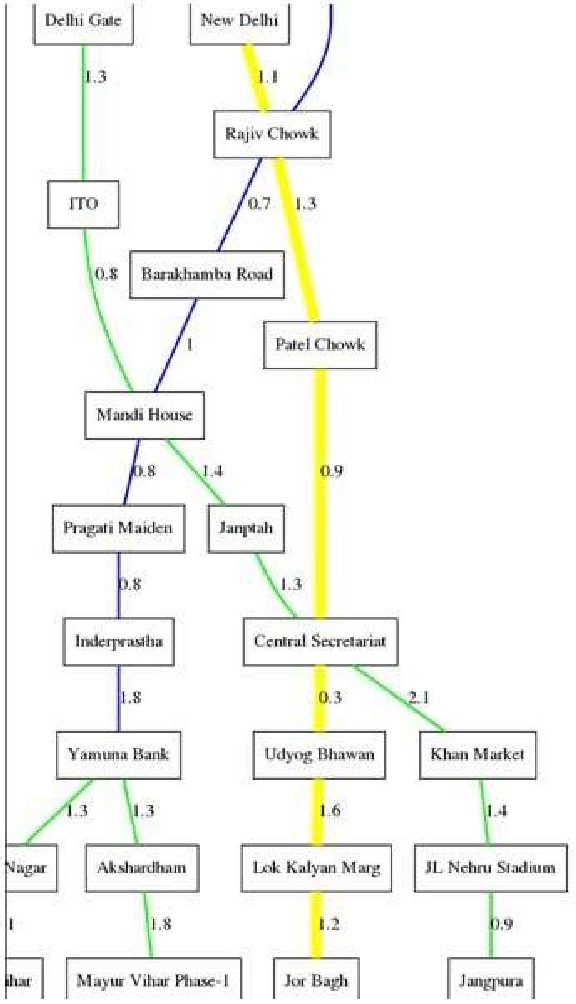
Outputs:

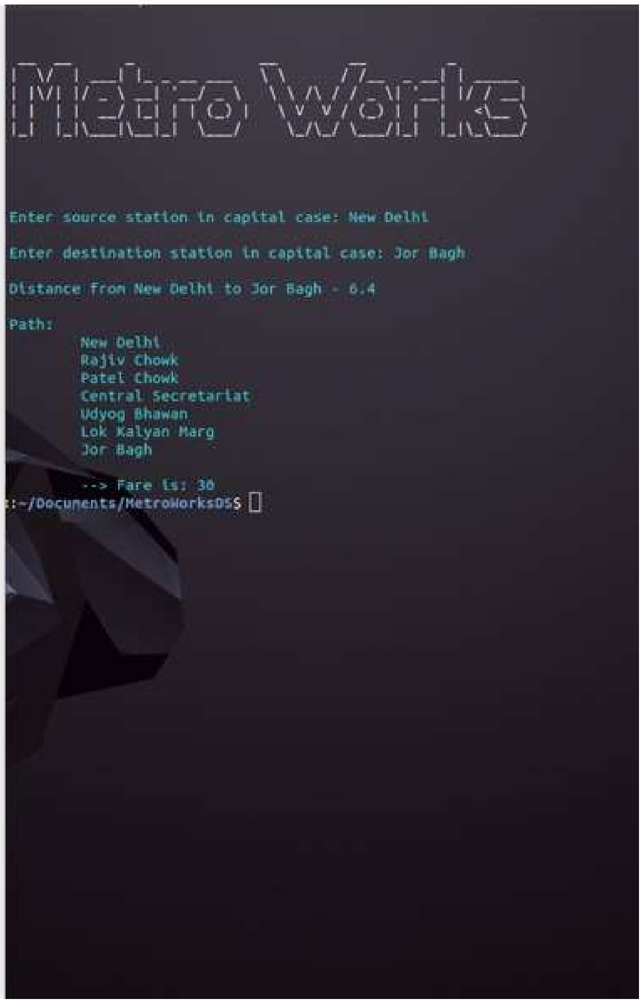




```
Enter source station in capital case:
```







```
Enter source station in capital case: New Delhi

Enter destination station in capital case: Jor Bagh

Distance from New Delhi to Jor Bagh - 6.4

Path:
        New Delhi
        Rajiv Chowk
        Patel Chowk
        Central Secretariat
        Udyog Bhawan
        Lok Kalyan Marg
        Jor Bagh

        --> Fare is: 30
:~/Documents/MetroWorksDS$
```

Time Complexities:

For Dijkstra Algorithm – O (E logV)

For Using Graph Data Structure – O (V+E)
where,
V is no. of vertices and E is no. of edges.

Future Scope:

This method can be extended to apply in various metro routes like Mumbai metro, Jaipur Metro etc.

This method can also be used to find shortest bus routes and their corresponding fares.