JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY,NOIDA

Data Structures

Mini Project

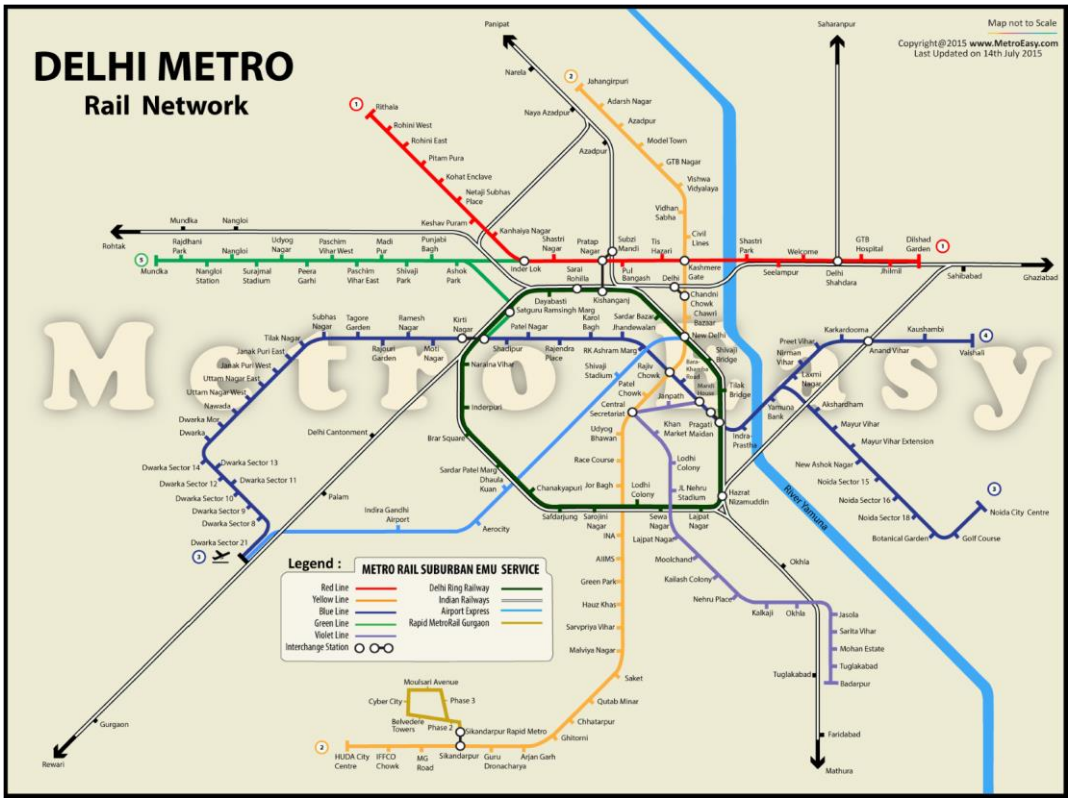
Topic - Metro Route Finder

Problem Statement:

There are millions of people who find it difficult to find the right route for their destination in the Delhi Metro. They face a lot of queries like which is the shortest route or the economical path for their journey. So here we are with the solution to this problem with a program named Metro Route Finder.

Introduction:

Metro Route Finder is a program written in C++ language, which can be beneficial for the people in their day-to-day metro life. It helps in finding out the shortest and the most economical path between two travel destinations on Delhi Metro. This program is also useful for the tourists as it helps to find the nearest metro station to popular tourist destinations like Lotus Temple, India gate, etc. One more feature added in this program is the recharge of the metro card.



List of Data Structures used in the project:

The language used for the completion and implementation of the aimed topic is C++.
Data structures used in the program are –

Maps- A Map is a type of fast key lookup data structure that offers a flexible means of indexing into its individual elements These keys, along with the data values associated with them, are stored within the Map. Each entry of a Map contains exactly one unique key and its corresponding value. Here in this project, it is used for mapping stations and distances.

Vector- Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. In vectors, data is inserted at the end.

String- Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'. It is used to input the names of the stations.

Graph- 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 represent networks that is the path between two stations. Graphs involves the usage of the adjacency matrix which is use to link the stations with each other.

Pair- The pair container is a simple container defined in <utility> header consisting of two data elements or objects. The first element is referenced as 'first' and the second element as 'second' and the order is fixed (first, second). Pair is used to combine together two values which may be different in type.

Stack- Stack is a linear data structure which follows a particular order in which the operations are performed. It follows the order LIFO(Last in First Out). It is used here for storing the input stations in the order they appear from source to the destination.

Priority queue- It is an abstract data type similar to a regular queue or stack data structure in which each element additionally has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. It is used here to arrange the stations in the priority of distances.

Breadth first search- It is an algorithm that is used to graph data or searching tree or traversing structures. It starts at the tree root and explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level. So, it is used to find the shortest path between two stations.

Dijkstra's algorithm- Used to find shortest paths from source to all vertices in the given graph. Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Here it is used to calculate the most economical path between the source and destination station.

Source code:

```cpp
#include<bits/stdc++.h>
#include<fstream>
#define ll long long
#define pb push_back
#define fi first
#define se second
#define mp make_pair
using namespace std;


map<string,ll>M;

char color[200][200]={'\0'};


class comparedis
{
    public:bool operator()(pair<ll,ll> &p,pair<ll,ll> &q)
        {
            return (p.se > q.se); // For min heap use > sign
        }
};


vector< pair<ll,ll> > v[100010];//Adjacency matrix
ll N;// N is no of vertices M is edges
string station[200];
map <string,string> tourm;


void recharge()
{
    fstream f;
    ll amt,ini,cid,fin,x;
    ll c_id,amount;
    f.open("paisa.txt",ios::in|ios::out);
    if(!f)
        cout<<"Not Found\n"<<endl;
    f.seekg(0);
    cout<<endl;
    cout<<"Enter Card Id\n";
    cin>>c_id;
```

```
    cout<<"Enter Amount\n";
    cin>>amount;
    f.clear();
    while(!f.eof())
    {
        ini=f.tellg();
        f.ignore();
        f>>cid;
        f>>amt;
        fin=f.tellg();
        if(cid==c_id)
        {
            x=amt+amount;
            f.seekg(ini);
            f<<endl<<cid<<endl<<x;
            cout<<"Recharge Details\n";
            cout<<"\nCard Id: "<<cid<<endl;
            cout<<"Initial Balance: "<<amt<<endl;
            cout<<"Recharge Amount: "<<amount<<endl;
            cout<<"Total Balance: "<<x<<endl;
            break;
        }
    }
    f.close();
}



void gettour()
{
    ifstream fin;
    string s1,s2;
    fin.open("tourplace.txt",ios::in);
    if(!fin)
        cout<<"Not Found\n";
    fin.seekg(0);
    fin.clear();
    while(!fin.eof())
    {
        getline(fin,s1);
        getline(fin,s2);
        tourm[s1]=s2;
```

```
87      fin.close();
88  }
89
90
91  //Given below code will print the path
92  void disp(ll src,ll dest,ll par[])
93  {
94      ll i,x,y,cn=0,ci=0;
95      stack<ll> st;
96      st.push(dest);
97      i=dest;
98      while(par[i]!=-1)
99      {
100         i=par[i];
101         st.push(i);
102     }
103     char col='\0';
104     while(!st.empty())
105     {
106         x=st.top();
107         st.pop();
108         if(!st.empty())
109             y=st.top();
110         cout<<station[x]<<" ";
111         cn++;
112         if(col=='\0')
113             col=color[x][y];
114         else if(col!='\0'&&col!=color[x][y])
115         {
116             char c=color[x][y];
117             ci++;
118             if(c=='b')
119                 cout<<"\t\tChange to blue line";
120             else if(c=='y')
121                 cout<<"\t\tChange to yellow line";
122             else if(c=='o')
123                 cout<<"\t\tChange to orange line";
124             else if(c=='g')
125                 cout<<"\t\tChange to green line";
126             else if(c=='r')
127                 cout<<"\t\tChange to red line";
128             else if(c=='v')
```

```
                cout<<"\t\tChange to Violet line";
            col=c;
        }
        cout<<endl;
    }
    cout<<endl<<"No of stations ="<<cn<<endl;
    cout<<"No of interchange stations ="<<ci-1<<endl;
    cout<<endl;
}


//To find shotest path
void bfs(ll src,ll dest)
{
    bool vis[100010]={false};
    ll par[100010];
    for(ll i=0;i<N;i++)
        par[i]=-1;
    queue<ll> q;
    q.push(src);
    vis[src]=true;
    while(!q.empty())
    {
        ll x=q.front();
        q.pop();
        ll vsz=v[x].size();
        for(ll i=0;i<vsz;i++)
        {
            ll y=v[x][i].fi;
            if(!vis[y])
            {
                par[y]=x;
                vis[y]=true;
                q.push(y);
            }
        }
        v[x].clear();
    }
    disp(src,dest,par);
}
```

```
//To find most economical path
void dijkstra(ll src,ll dest)
{
    bool vis[100010]={false};
    ll dist[100010], par[100010];
    for(ll i=0;i<N;i++)
    {
        dist[i]=LLONG_MAX;
        par[i]=-1;
    }
    priority_queue< pair<ll,ll>,vector< pair<ll,ll> >,comparedis > pq;
    pq.push(mp(src,0));
    dist[src]=0;
    par[src]=-1;
    vis[src]=true;
    while(!pq.empty())
    {
        pair<ll,ll> k=pq.top();
        pq.pop();
        ll x=k.fi;
        //if(x==dest)
        //  break;
        ll vsz=v[x].size();
        for(ll i=0;i<vsz;i++)
        {
            ll y=v[x][i].fi;
            ll w=v[x][i].se;
            if(dist[x]+w < dist[y])
            {
                par[y]=x;
                dist[y]=dist[x]+w;
            }
            if(!vis[y])
            {
                vis[y]=true;
                pq.push(mp(y,dist[y]));
            }
        }
        v[x].clear();
    }
    disp(src,dest,par);
```

```
213  void consmap()//To assign values to metro stations
214  {
215      ifstream fin;
216      string s;
217      fin.open("list.txt",ios::in);
218      ll l=0;
219      fin.seekg(0);
220      fin.clear();
221      while(!fin.eof())
222      {
223          getline(fin,s);
224          M[s]=l;
225          station[l]=s;
226          l++;
227      }
228      N=l-1;
229      fin.close();
230      map<string,ll> ::iterator it;
231      //for(it=M.begin();it!=M.end();it++)
232      //  cout<<it->se<<" "<<it->fi<<endl;
233  }
234
235
236  void addedge(char fname[],ll w)//To add edges
237  {
238      ifstream fin;
239      string s;
240      ll x,y;
241      fin.open(fname,ios::in);
242      fin.seekg(0);
243      getline(fin,s);
244      x=M[s];
245      char c=fname[0];
246      fin.clear();
247      while(!fin.eof())
248      {
249          getline(fin,s);
250          y=M[s];
251          v[x].pb(mp(y,w));
252          v[y].pb(mp(x,w));
253          color[x][y]=c;
254          color[y][x]=c;
```

```
255         x=y;
256     }
257     fin.close();
258 }
259
260 void consgraph()//To construct edges
261 {
262     string s;
263     addedge("blueline.txt",0);
264     addedge("yellowline.txt",0);
265     addedge("redline.txt",0);
266     addedge("greenline.txt",0);
267     addedge("violetline.txt",0);
268     addedge("bluext.txt",0);
269     addedge("orangeline.txt",1);
270 }
271
272
273 int main()
274 {
275     string source,destination;
276     ll i,x,y,w,src,dest,k,choice,dec;
277     char ch;
278     gettour();
279     consmap();
280     do
281     {
282         cout<<endl;
283         cout<<"1.To Route between two stations\n";
284         cout<<"2.To check nearest metro station to a tourist place\n";
285         cout<<"3.To Recharge your Smart Card\n";
286         cin>>dec;
287         switch(dec)
288         {
289             case 1:
290                     do
291                     {
292                         consgraph();//To build the adjacency matrix
293                         cout<<"Enter station 1\n";
294                         getline(cin,source);
295                         getline(cin,source);
296                         //cout<<source<<endl;
```

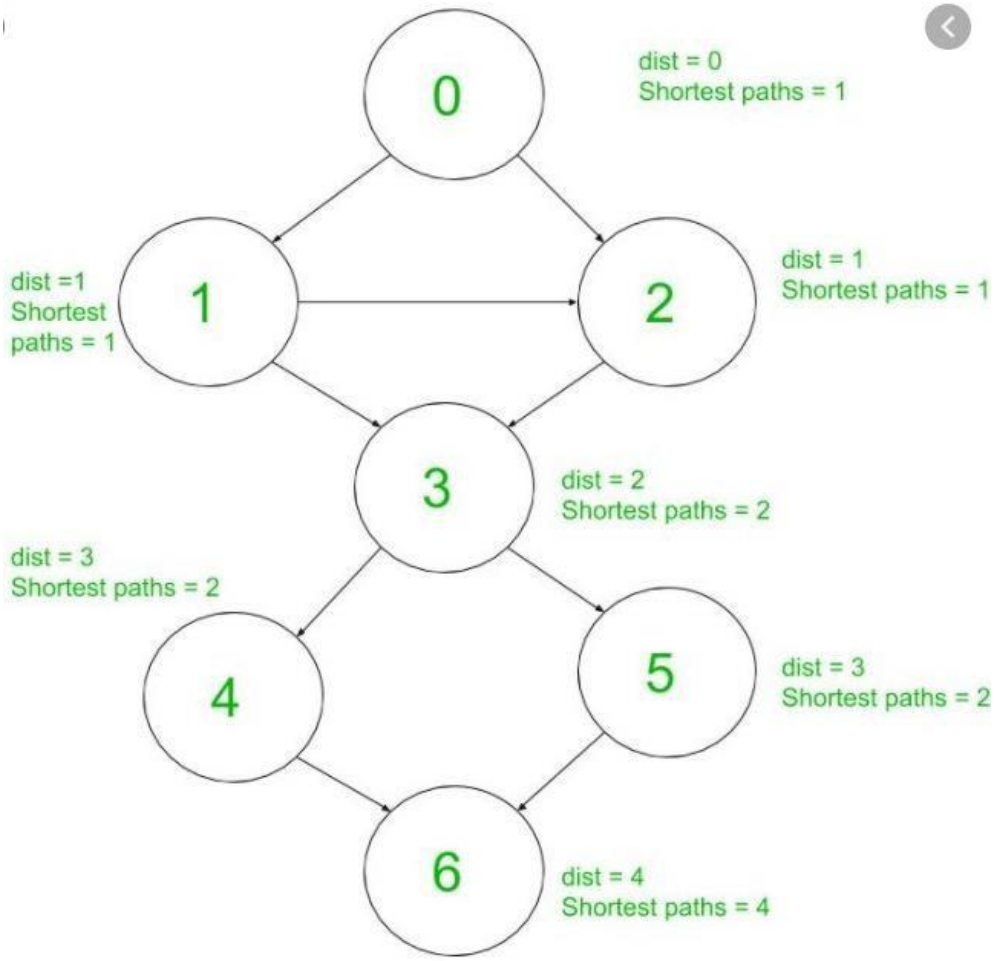
```
286         cin>>dec;
287         switch(dec)
288         {
289             case 1:
290                     do
291                     {
292                         consgraph();//To build the adjacency matrix
293                         cout<<"Enter station 1\n";
294                         getline(cin,source);
295                         getline(cin,source);
296                         //cout<<source<<endl;
297                         cout<<"Enter station 2\n";
298                         getline(cin,destination);
299                         //cout<<destination<<endl;
300                         src=M[source];
301                         dest=M[destination];
302                         cout<<"1.For most economic path\n";
303                         cout<<"2.For shortest path";
304                         cin>>choice;
305                         switch(choice)
306                         {
307                             case 1:dijkstra(src,dest);
308                                     break;
309                             case 2:bfs(src,dest);
310                                     break;
311                         }
312                         cout<<"Do you wish to check for any other station\n";
313                         cin>>ch;
314                     }while(ch=='Y'||ch=='y');
315                     break;
316             case 2:
317                     do
318                     {
319                         string place;
320                         cout<<"Enter a place\n";
321                         getline(cin,place);
322                         getline(cin,place);
323                         string st;
324                         st=tourm[place];
325                         cout<<st<<endl;
326                         cout<<"Do you wish to check for any other place\n";
327                         cin>>ch;
```

```
303                         cout<<"2.For shortest path";
304                         cin>>choice;
305                         switch(choice)
306                         {
307                             case 1:dijkstra(src,dest);
308                                     break;
309                             case 2:bfs(src,dest);
310                                     break;
311                         }
312                         cout<<"Do you wish to check for any other station\n";
313                         cin>>ch;
314                     }while(ch=='Y'||ch=='y');
315                     break;
316             case 2:
317                     do
318                     {
319                         string place;
320                         cout<<"Enter a place\n";
321                         getline(cin,place);
322                         getline(cin,place);
323                         string st;
324                         st=tourm[place];
325                         cout<<st<<endl;
326                         cout<<"Do you wish to check for any other place\n";
327                         cin>>ch;
328                     }while(ch=='Y'||ch=='y');
329                     break;
330             case 3:
331                     do
332                     {
333                         recharge();
334                         cout<<"Do you wish to recharge some other smart card\n";
335                         cin>>ch;
336                     }while(ch=='Y'||ch=='y');
337                     break;
338         }
339         cout<<"Do you wish to go back to main menu\n";
340         cin>>ch;
341     }while(ch=='Y'||ch=='y');
342     return 0;
343 }
344
```

Implementation details and results:

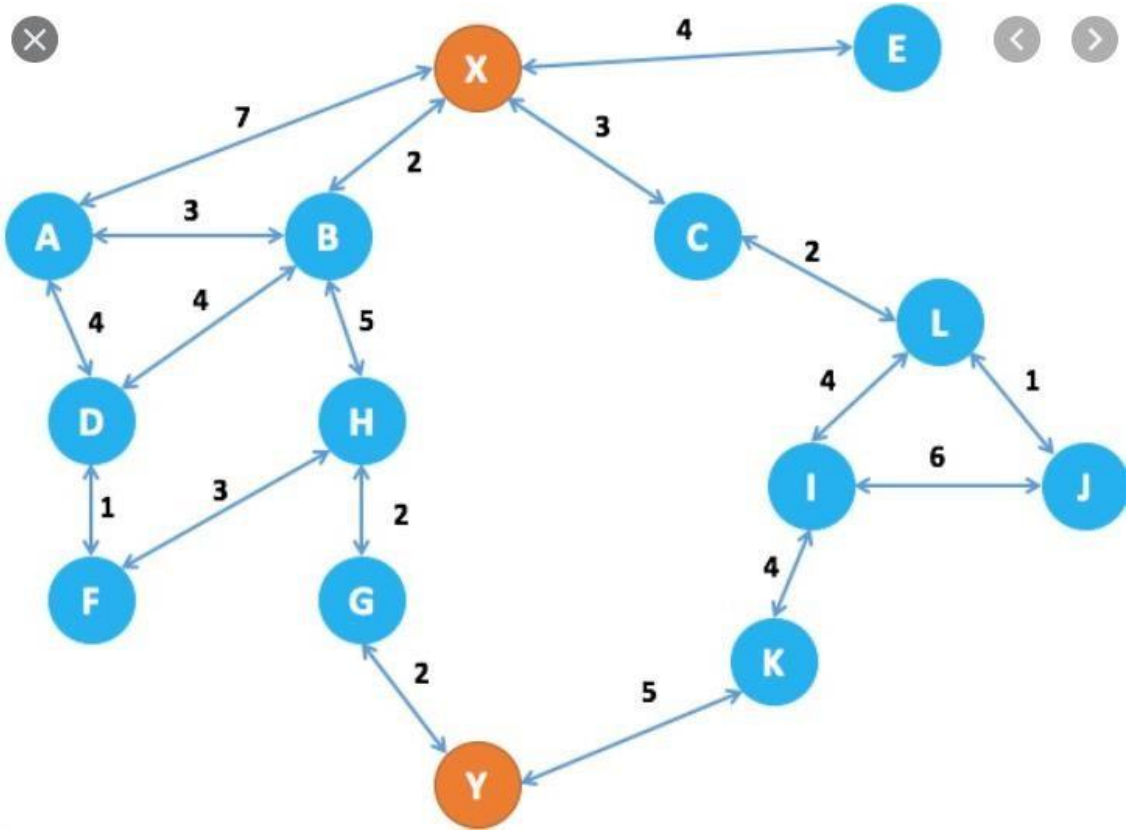
This program is built by combining mainly two important data structures – Breadth first search and Dijkstra's algorithm.

We say that BFS is the algorithm to use if we want to find the shortest path in an undirected, unweighted graph. The claim for BFS is that the first time a node is discovered during the traversal, that distance from the source would give us the shortest path.



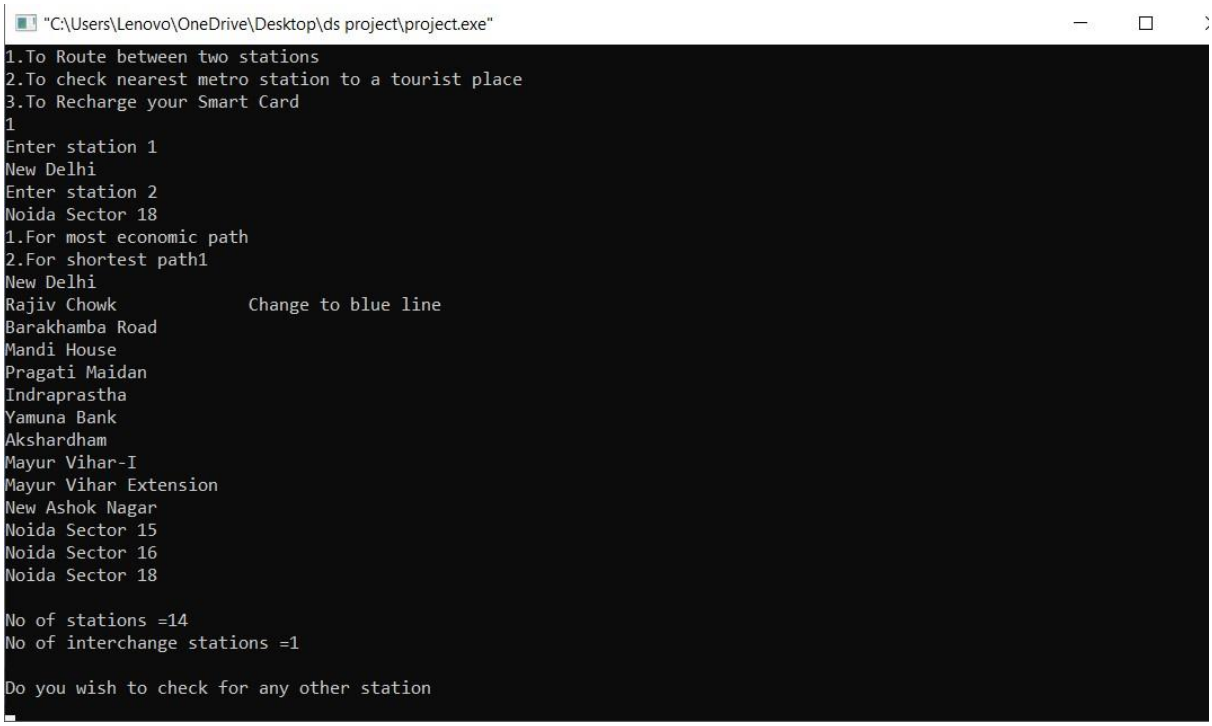
The Time complexity of BFS is O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.
Dijkstra's algorithm can be used to determine the shortest path from one node in a graph to every other node within the same

graph data structure, provided that the nodes are reachable from the starting node.

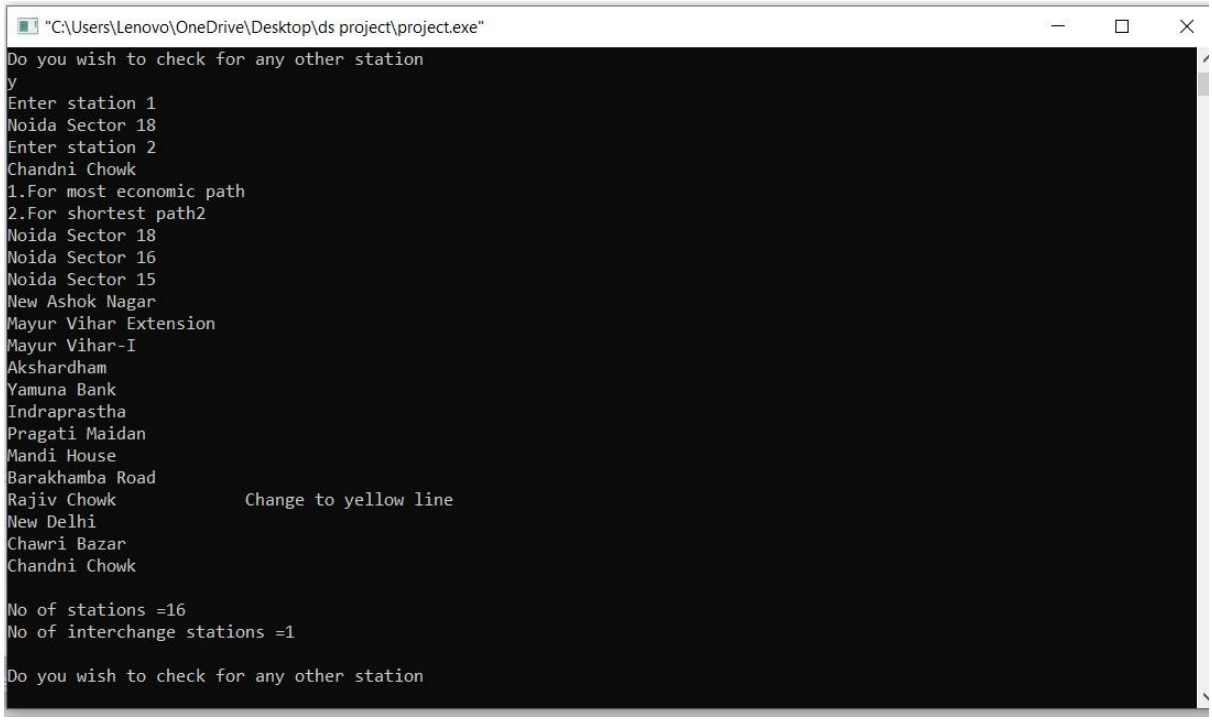


Time Complexity of Dijkstra's Algorithm is O (V 2) but with minpriority queue it drops down to O (V + E l o g V) .

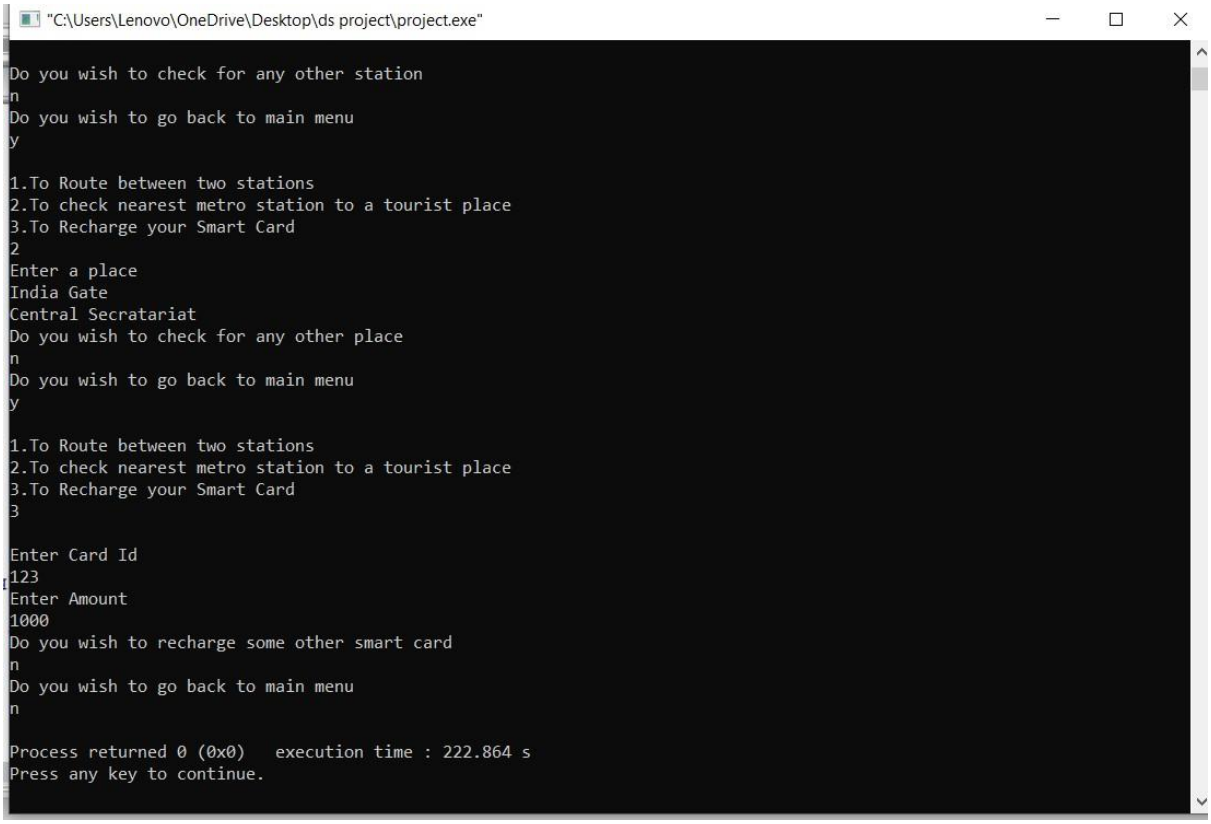
- To check the route between two stations which specifies all the metro stations needed to be covered with the additional information about the interchanging stations i.e., where the metro line will change. This gives the information about the most economic path.



- To check the route between two stations which would be the shortest path between the source and destination station.



- To check the nearest metro station to a tourist place we input the tourist spot and on entering it we get know the station name closest to it.
- Also helps to recharge the smart metro card instantly.



Conclusion:

The given project was a great spot of research and personal development in the field of Data Structures. I got to learn many new things and implemented our knowledge to build a program that could help the people in the best possible way. It could help them by suggesting the route best suited to them and helps to save their time and money. Besides providing all these facilities, it can be accessed by all.