ASSIGNMENT 3

AIM :

There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A, or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representations used.

OBJECTIVE :

To input Graph and display the graph matrix.

THEORY :

Graph is a data structure that consists of following two components:

1. A finite set of vertices also called as nodes.

2. A finite set of ordered pair of the form (u, v) called as edge.

Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.

Following two are the most commonly used representations of a graph.
1. Adjacency Matrix
2. Adjacency List

There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

Adjacency Matrix:

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:

$$\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 1 \\
4 & 1 & 1 & 0 & 1 & 0
\end{array}$$

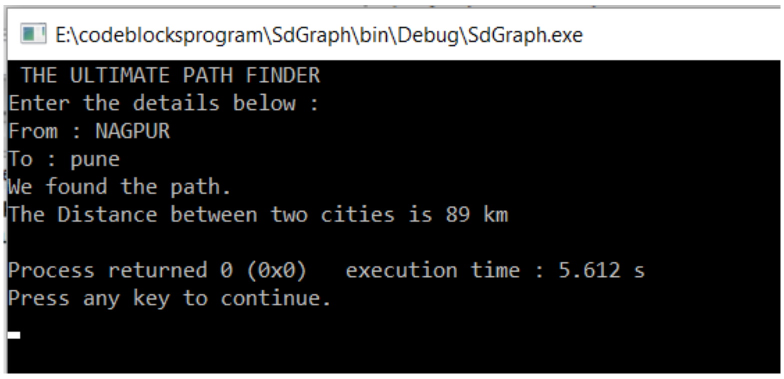
ALGORITHM :

1. Decide names of cities and assign them indices according to matrix.
2. Create matrix containing value of distance of path between to cities at particular row and column index.
3. Take input of from and destination cities from user.
4. Map the name of cities with corresponding indices.
5. Find value of distance in corresponding row and column.

CODE :

```
#include <bits/stdc++.h>

using namespace std;

const int  SIZE = 5;

string cities[SIZE] = {"PUNE", "MUMBAI", "NAGPUR", "AMRAVATI", "DELHI"};

int paths[SIZE][SIZE] = {
    {0,67,-1,-1,345},
    {67,0,-1,45,-1},
    {89,-1,0,-1,-1},
    {-1,-1,-1,0,-1},
    {-1,-1,-1,95,0}
};

int indexOfCity(string cityName){
    transform(cityName.begin(), cityName.end(), cityName.begin(), ::toupper);
    for(int i =0; i< SIZE; i++){

        if( cityName == cities[i]){
            return i;
        }
    }
        return -1;
}

void checkPath(){
    string from, to;
    cout<<" THE ULTIMATE PATH FINDER  "<<endl;
    cout<<"Enter the details below : "<<endl;
    cout<<"From : ";
    cin>>from;
    cout<<"To : ";
    cin>>to;
    //cout<<indexOfCity(from)<<"  "<<indexOfCity(to)<<endl;
    if(paths[indexOfCity(from)][indexOfCity(to)] != -1){
        if(paths[indexOfCity(from)][indexOfCity(to)] == 0)
        {
            cout<<"Please change your destination city ."<<endl;
        }
        else{
            cout<<"We found the path."<<endl;
            cout<<"The Distance between two cities is
"<<paths[indexOfCity(from)][indexOfCity(to)]<<" km"<<endl;
        }
    }
    else{
        cout<<"Sorry there is no path between these two cities"<<endl;
    }
}

int main()
{
    checkPath();
    return 0;
}
```

OUTPUT:-



```
E:\codeblocksprogram\SdGraph\bin\Debug\SdGraph.exe
 THE ULTIMATE PATH FINDER
Enter the details below :
From : NAGPUR
To : pune
We found the path.
The Distance between two cities is 89 km

Process returned 0 (0x0)   execution time : 5.612 s
Press any key to continue.
```

CONCLUSION:-

Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).

Cons: Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time.