**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:-**

**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:

Adjacency Matrix Representation

**ALGORITHM:-**

**CODE:-**

#include<iostream>

using namespace std;

#include<string.h>

#include<stdlib.h>

int main()

{

int flight[10][10],src,dest;

string time[10][10];

flight[0][1]=1;

flight[0][2]=1;

flight[0][3]=1;

flight[1][0]=1;

flight[1][2]=1;

flight[1][3]=1;

flight[2][0]=1;

flight[2][1]=1;

flight[2][3]=1;

flight[3][1]=1;

flight[3][2]=1;

time[0][1]="11am to 4am" ;

time[0][2]="12pm to 8pm";

time[0][3]="3am to 7pm";

time[1][0]="4am to 12:50am";

time[1][2]="5:30pm to 9:45pm";

time[1][3]="2:15pm to 4:50pm";

time[2][0]="7:45am to 11:30pm ";

time[2][1]="6:25am to 7:45pm";

time[2][3]="3:20pm to 9:55pm ";

time[3][1]="10:20am to 7:50pm";

time[3][2]="3:40pm to 10:45pm ";

char ch='y';

while(1)

{

cout<<endl<<endl<<"-----LIST OF AIRPORTS-----"<<endl;

cout<<"1.Pune = 0 "<<"\n"<<"2.Mumbai = 1 "<<"\n"<<"3.Banglore = 2 "<<"\n"<<"4.Delhi = 3"<<endl;

cout<<"Enter the Source City(No.) and Destination(No.) : "<<endl;

cin>>src;

cin>>dest;

if(flight[src][dest]==1)

{

cout<<"Flight is available from ";

cout<<time[src][dest]<<endl;

}

else

{

cout<<"Flight is not available "<<endl;

}

cout<<"If u want to exit press 'y' else press other character "<<endl;

cin>>ch;

if(ch=='y')

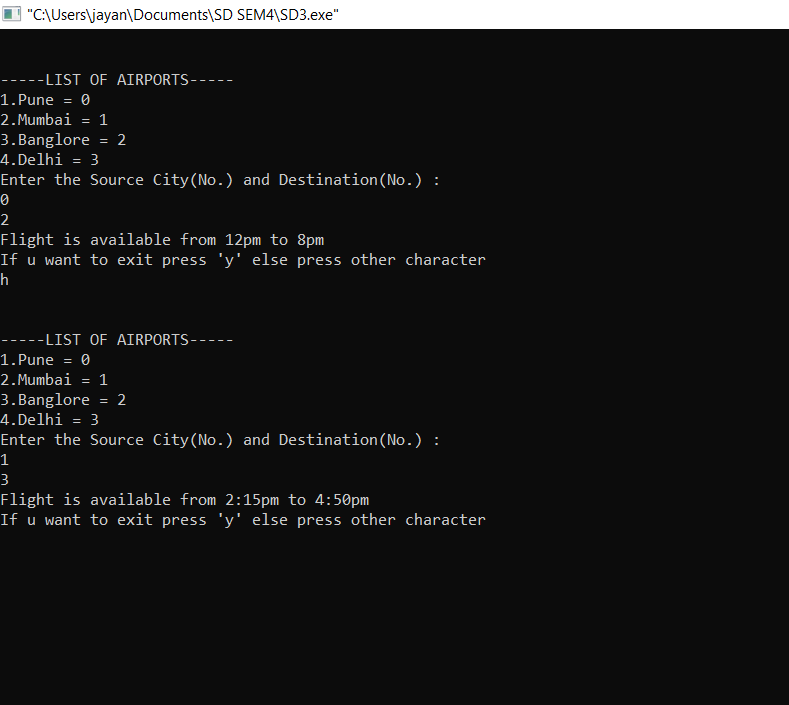
exit(0);

}

return 0;

}

**OUTPUT:-**

****

**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.