ASSIGNMENT 3

AIM:-Represent a Graph using adjacency matrix.

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.

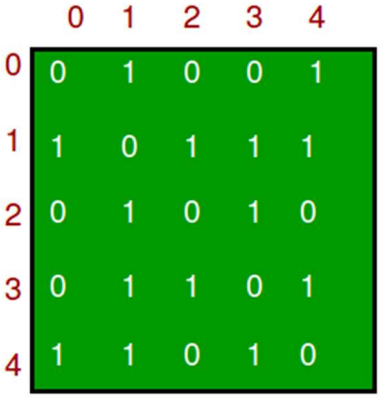
Following two are the most commonly used representations of a graph.
1. Adjacency Matrix
2. Adjacency List

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:



ALGORITHM:-

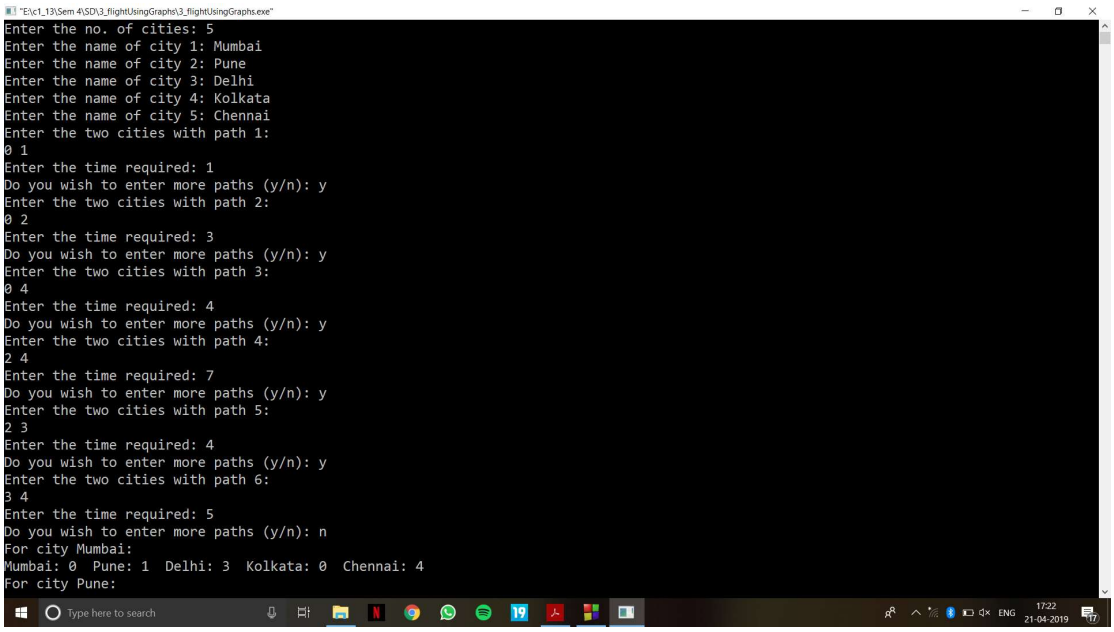
CODE:-

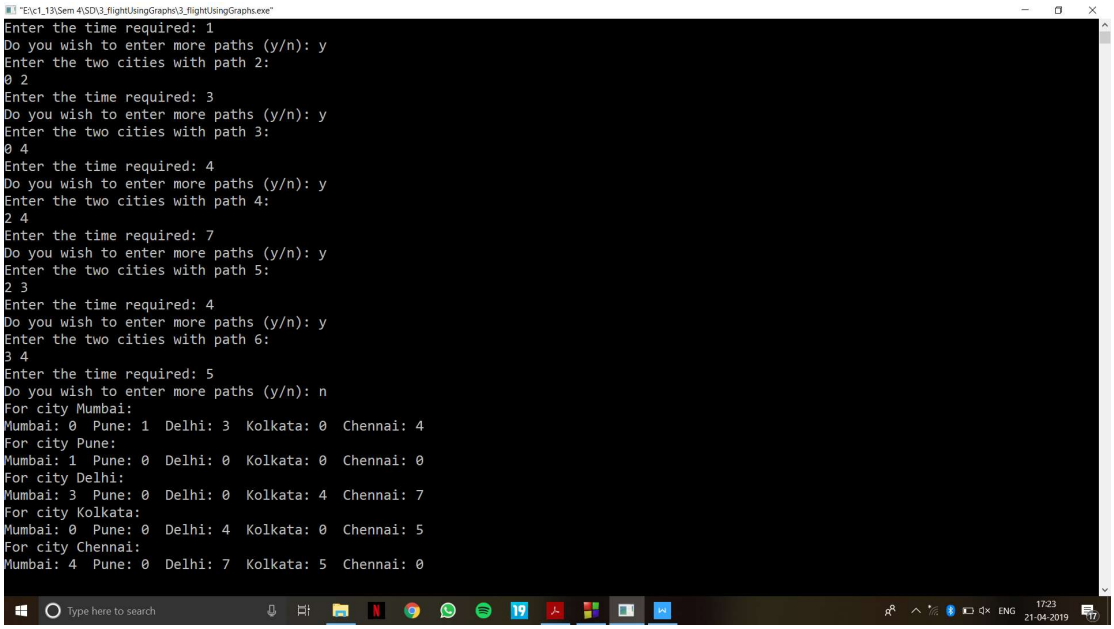
```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;

int main()
{
   int cities,paths = 0,c1,c2,time;
   char ch = 'y';
   cout<<"Enter the no. of cities: ";
   cin>>cities;
   int g[cities][cities];
   string cityName[cities];

   for(int i=0;i<cities;i++)
   {
      for(int j=0;j<cities;j++)
      {
         g[i][j] = 0;
      }
      cout<<"Enter the name of city "<<i+1<<": ";
      cin>>cityName[i];
   }
   while(ch == 'y')
   {
      cout<<"Enter the two cities with path "<<paths+1<<": "<<endl;
      cin>>c1>>c2;
      cout<<"Enter the time required: ";
      cin>>time;
      g[c1][c2] = time;
      g[c2][c1] = time;
      paths++;
      cout<<"Do you wish to enter more paths (y/n): ";
      cin>>ch;
   }
   for(int i=0;i<cities;i++)
   {
      cout<<"For city "<<cityName[i]<<": "<<endl;
      for(int j=0;j<cities;j++)
      {
         cout<<cityName[j]<<": "<<g[i][j]<<"  ";
      }
      cout<<endl;
   }
   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.