

ASSIGNMENT 3

AIM:-Represent a Graph using adjacency matrix.

OBJECTIVE:-

To Create Graph and Display it.

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 $V \times V$ 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:

	0	1	2	3	4
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

ALGORITHM:-

- 1} Decide name of cities and assign them indices according to matrix.
- 2} Create matrix value of distance of path between two cities at particular row and index.
- 3} Take input and destination cities from users.
- 4} Make the name of cities with corresponding indices.
- 5} find value of distance in corresponding cities.

CODE:-

```
#include<iostream>

#define MAX 10

using namespace std;

class airport
{
    string city[MAX];
    int distance[10][10];

public :
    int n;
    airport();
```

```
void read_city();
void show_graph();
};
airport::airport()
{
    n=0;
    for(int i=0;i<MAX;i++)
    {
        for(int j=0;j<MAX;j++)
            distance[i][j]=0;
    }
}
void airport::read_city()
{
    int k;
    cout<<"\nEnter the no. of cities: " ;
    cin>>n;
    cout<<"Enter city name:\n";
    for(int k=0;k<n;k++)
    {
        cout<<k+1<<"] ";
        cin>>city[k];
    }
    for(int i=0;i<n;i++)
    {
        for(int j=i+1 ; j<n ; j++)
        {
```

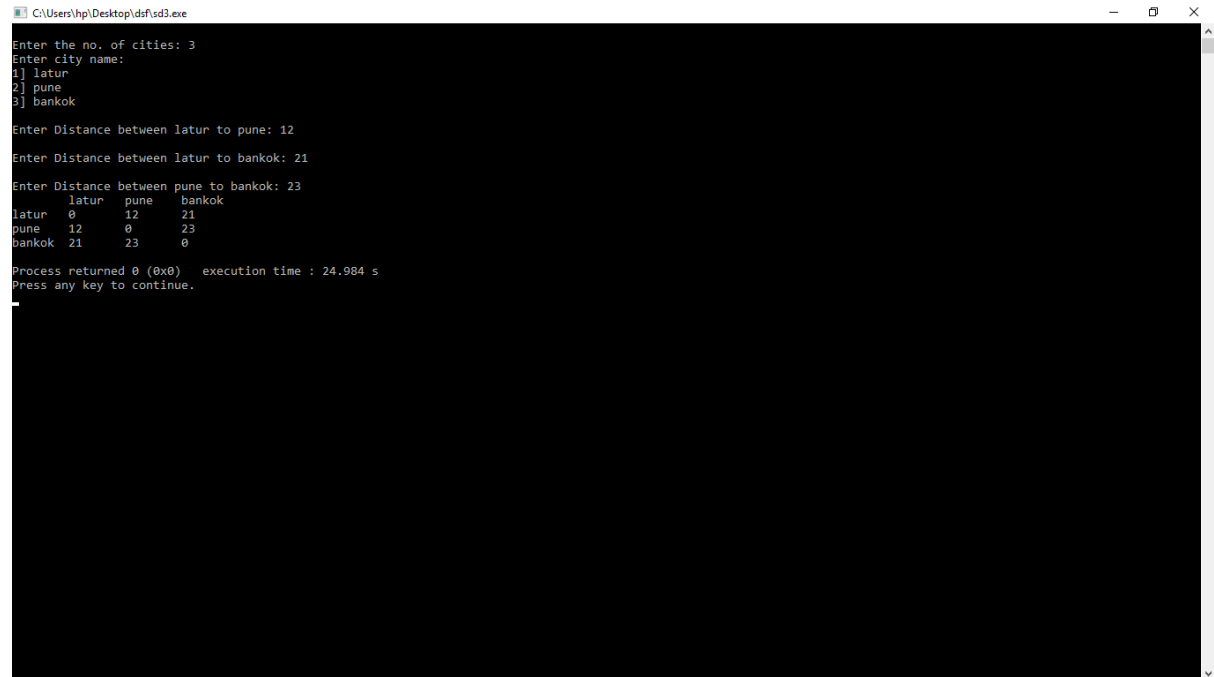
```
        cout<<"\nEnter Distance between "<<city[i]<<" to "<<city[j]<<": ";
        cin>>distance[i][j];
        distance[j][i]=distance[i][j];
    }
}

void airport::show_graph()
{
    cout<<"\t";
    for(int k=0;k<n;k++)
    {
        cout<<city[k]<<"\t";
    }
    cout<<endl;
    for(int i=0;i<n;i++)
    {
        cout<<city[i]<<"\t";
        for(int j=0;j<n;j++)
        {
            cout<<distance[i][j]<<"\t";
        }
        cout<<endl;
    }
}

int main()
{
    airport obj;
```

```
obj.read_city();  
  
obj.show_graph();  
  
}
```

OUTPUT:-



```
C:\Users\hpl\Desktop\dfs\sd3.exe  
Enter the no. of cities: 3  
Enter city name:  
1] latur  
2] pune  
3] bankok  
Enter Distance between latur to pune: 12  
Enter Distance between latur to bankok: 21  
Enter Distance between pune to bankok: 23  
latur   pune   bankok  
latur   0      12    21  
pune    12     0     23  
bankok  21    23     0  
Process returned 0 (0x0)   execution time : 24.984 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.