

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 learn adjacency list representation of graph.

Theory:-

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

Adjacency List:

An array of lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the *i*th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is adjacency list representation of the above graph.

Algorithm:-

1.BFS :-

- **Step 1:** SET STATUS = 1 (ready state)
for each node in G
- **Step 2:** Enqueue the starting node A
and set its STATUS = 2
(waiting state)
- **Step 3:** Repeat Steps 4 and 5 until
QUEUE is empty

- **Step 4:** Dequeue a node N. Process it and set its STATUS = 3 (processed state).
- **Step 5:** Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)
[END OF LOOP]
- **Step 6:** EXIT

2.DFS :-

- **Step 1:** SET STATUS = 1 (ready state) for each node in G
- **Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting state)
- **Step 3:** Repeat Steps 4 and 5 until STACK is empty
- **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
- **Step 5:** Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)
[END OF LOOP]
- **Step 6:** EXIT

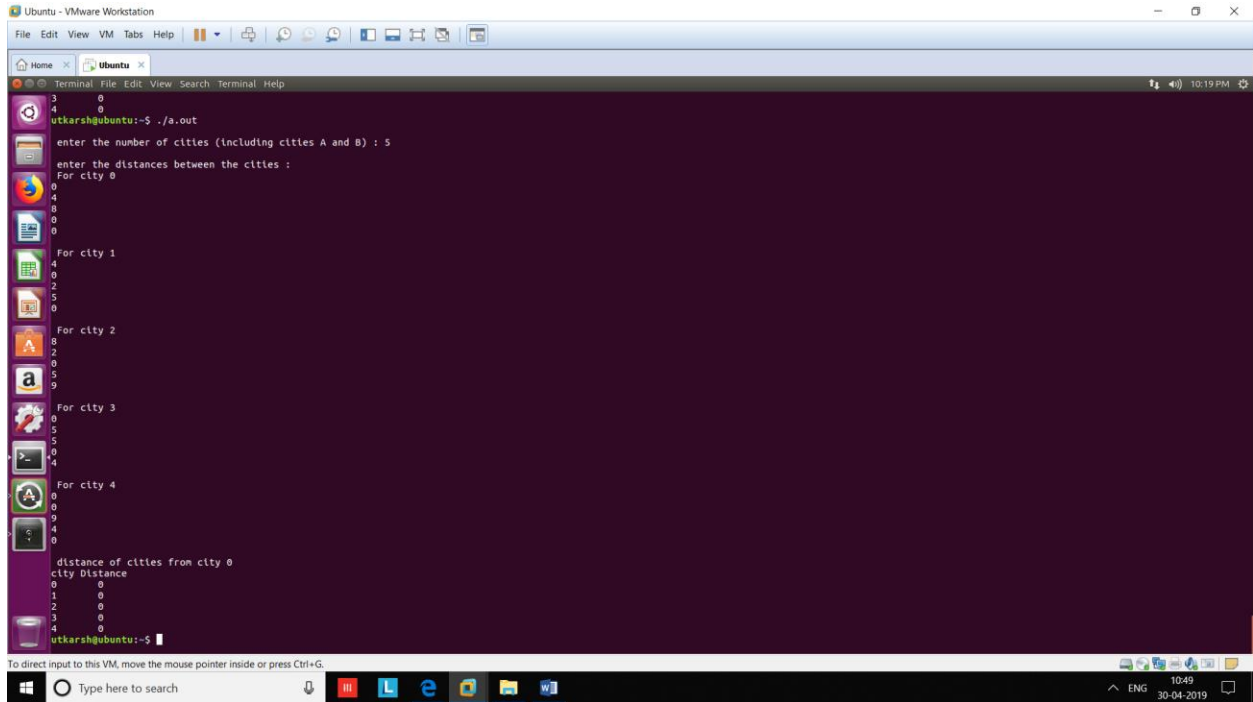
Program Code:-

```
#include<iostream>
#define MAX 20
using namespace std;
class dijkstra
{
int city;
int distance[MAX][MAX];
int d[MAX];
```

```
int visited[MAX];
public:
void city_no();
int minvertex();
void matrix_fill();
void dijkstra_code();
void display();
};
void dijkstra::city_no()
{
cout<<"\n enter the number of cities (including cities A and B) : ";
cin>>city;
}
int dijkstra::minvertex()
{
int mvertex=-1;
for(int i=0;i<city;i++)
{
if(visited[i]==0 && (mvertex==-1 || d[i]<d[mvertex]))
mvertex=i;
}
return mvertex;
}
void dijkstra::matrix_fill()
{
cout<<"\n enter the distances between the cities : ";
for(int i=0;i<city;i++)
{
cout<<"\n For city "<<i<<endl;
for(int j=0;j<city;j++)
{
if(i==j)
distance[i][j]=0;
cin>>distance[i][j];
}
d[i]=INT_MAX;
visited[i]=0;
}
}
```

```
void dijkstra::dijkstra_code()
{
d[0]=0;
for(int i=0;i<city-1;i++)
{
int mvertex=minvertex();
visited[mvertex]=1;
for(int j=0;j<city;j++)
{
if((distance[mvertex][j]!=0)&&(visited[j]==0))
{
int dist=d[mvertex]+distance[mvertex][j];
if(dist<d[j])
d[j]=dist;
}
}
}
}
void dijkstra::display()
{
cout<<"\n distance of cities from city 0 \n";
cout<<"city Distance\n";
for(int i=0;i<city;i++)
cout<<i<<"\t"<<d[i]<<endl;
}
int main()
{
dijkstra sp;
sp.city_no();
sp.matrix_fill();
sp.dijkstra_code();
sp.display();
return 0;
}
```

Output Screenshots:-



The screenshot shows a terminal window titled 'Ubuntu - VMware Workstation'. The terminal displays the execution of a C program. The user enters the number of cities as 5. The program then prompts for distances between cities for each city (0 to 4). The output shows the distances from city 0 to cities 1, 2, 3, and 4, which are 0, 1, 2, 3, and 4 respectively.

```
utkarsh@ubuntu:~$ ./a.out
enter the number of cities (including cities A and B) : 5
enter the distances between the cities :
For city 0
0
4
0
0
0
For city 1
4
0
2
5
0
For city 2
8
0
5
9
0
For city 3
6
0
5
9
0
For city 4
0
9
4
0
0
distance of cities from city 0
city Distance
0 0
1 1
2 2
3 3
4 4
utkarsh@ubuntu:~$
```

Conclusion:-

Thus,we have studied adjacency graph representation.