

## Experiment No. 9

### Aim:

To represent flight paths between cities using a **graph** data structure with adjacency matrix or adjacency list representation, and to check whether the graph is **connected or not**.

### Objective:

The objective of this experiment is to:

1. Represent cities and their flight connections using a **graph data structure**.
2. Implement the graph using **adjacency matrix** or **adjacency list** representation.
3. Assign a **cost to each edge**, such as flight time or fuel consumption.
4. Traverse the graph to check whether it is **connected or not** (i.e., whether there exists a path between every pair of cities).

### Problem Statement:

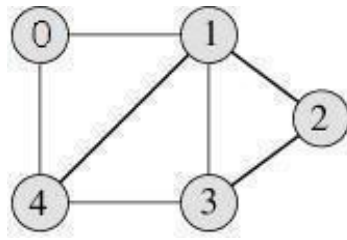
There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight takes to reach city B from A or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Check whether the graph is connected or not.

### 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.

The pair is ordered because  $(u, v)$  is not same as  $(v, u)$  in case of directed graph(di-graph). The pair of form  $(u, v)$  indicates that there is an edge from vertex  $u$  to vertex  $v$ . The edges may contain weight/value/cost.



**undirected graph with 5 vertices.**

Following two are the most commonly used representations of 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:**

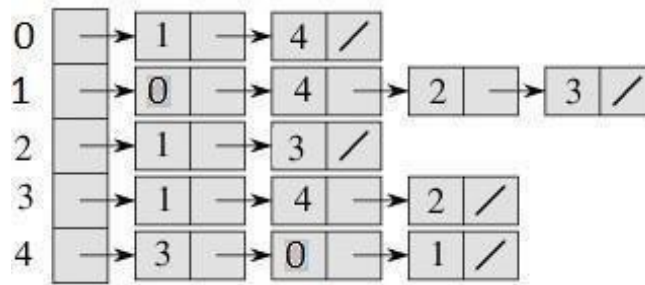
Adjacency Matrix is a 2D array of size  $V \times V$  where  $V$  is the number of vertices in a graph. Let the 2D array be  $adj[][]$ , a slot  $adj[i][j] = 1$  indicates that there is an edge from vertex  $i$  to vertex  $j$ . Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix 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  $w$ .

	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

The adjacency matrix for the above example graph is:

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.



Adjacency List Representation the above Graph

#### Algorithm:

Graph creation using Adjacency Matrix

1. Declare an array of  $M[\text{size}][\text{size}]$  which will store the graph.
2. Enter how many nodes you want in a graph.
3. Enter the edges of the graph by two vertices each, say  $V_i, V_j$  indicates some edge
4. If the graph is directed set  $M[i][j]=1$ . If graph is undirected set  $M[i][j] = 1$  and  $M[j][i] = 1$  as well.
5. When all the edges of the desired graph is entered print the graph  $M[i][j]$ .

Graph creation using Adjacency list

1. Declare node structure for creating adjacency list.
2. Initialize an array of nodes. This array will act as head nodes. The index of  $\text{head}[]$  will be the starting vertex.
3. The create function will create the adjacency list.

#### Input:

**Note: Write input you are giving to your program**

#### Output:

**Note: Write output of your program**

#### Conclusion:

Thus, the flight network was successfully represented using graph /Adjacency Matrix/List and graph connectivity was checked using BFS traversal.