**Aim:** Implementation of graph and its traversals

**Objective:**

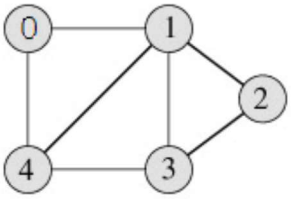
1. Understanding representation of Graph and its traversal.

**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.




Representations of graph:

1. Adjacency Matrix
2. Adjacency List

1) 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:

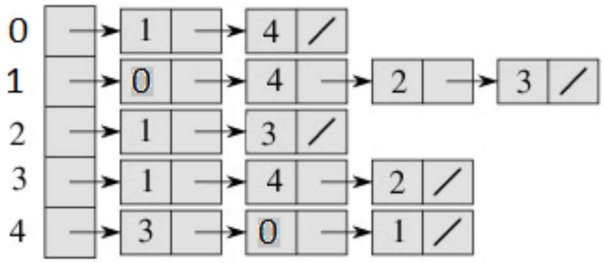
| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 1 | 0 | 1 | 0 |

**Adjacency Matrix Representation of the above graph**

**2) Adjacency List:**

An array of linked lists is used. Size of the array is equal to number of vertices. Let the array be array[]. An entry array[i] represents the linked 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 stored in nodes of linked lists. Following is adjacency list representation of the above graph.




Adjacency List Representation of the above Graph

**Graph Traversal**

Graph traversal is technique used for visiting each vertex of the graph exactly once. There are two graph traversal techniques:

1. DFS (Depth First Search)
2. BFS (Breadth First Search)

**1) DFS (Depth First Search):**

**Algorithm:**

1) Define a Stack whose size is number of vertices in the graph.
2) Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.
3) Visit any one of the adjacent vertex of the vertex which is at top of the stack which is not visited and push it on to the stack.
4) Repeat step 3 until there are no new vertex to be visited from the vertex on top of the stack.
5) When there is no new vertex to be visited then use back tracking and pop one vertex from the stack.
6**)** Repeat steps 3, 4 and 5 until stack becomes Empty.

**2) BFS (Breath First Search)**

1) Define a Queue of size equal to total number of vertices in the graph.
2) Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.
3) Visit all the adjacent vertices of the vertex which is at front of the Queue which is not visited and insert them into the Queue.

4) When there is no new vertex to be visited from the vertex which is at the front of the Queue then delete that vertex from the Queue.
5) Repeat step 3 and 4 until queue becomes empty.
6) When queue becomes Empty, then produce final spanning tree by removing unused edges from the graph

**Source code for the implementation:**

**(Write only important functions)**

Post Lab Assignment:

1) Discuss real world applications of graph.