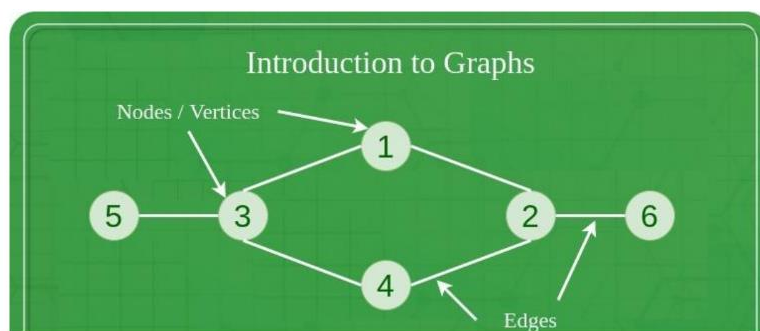


DS-UNIT 3

1. What is graph? Explain it.

ANS:

A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices (V) and a set of edges (E). The graph is denoted by $G(E, V)$.



2. What is shortest path algorithm.

ANS:

- By using graphs, we can easily find the shortest path, neighbours of the nodes, and many more.
- Graphs are used to implement algorithms like DFS and BFS.
- It is used to find minimum spanning tree which has many practical applications.
- It helps in organizing data.
- Because of its non-linear structure, helps in understanding complex problems and their visualization.

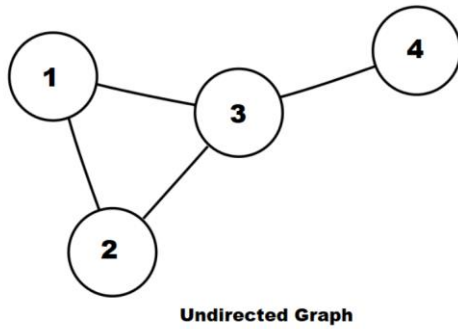
3. List and explain different types of graphs.

Ans:

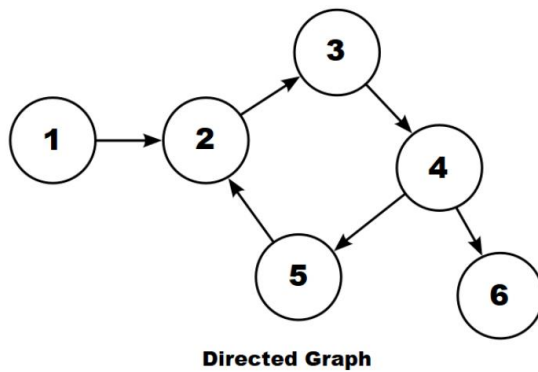
Types of Graphs in Data Structure

The most common types of graphs in the data structure are mentioned below:

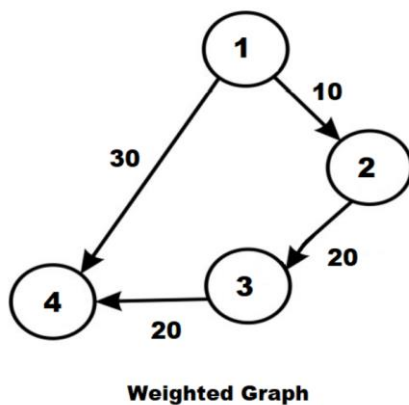
1. Undirected: A graph in which all the edges are bi-directional. The edges do not point in a specific direction.



2. Directed: A graph in which all the edges are uni-directional. The edges point in a single direction.



3. Weighted Graph: A graph that has a value associated with every edge. The values corresponding to the edges are called weights. A value in a weighted graph can represent quantities such as cost, distance, and time, depending on the graph. Weighted graphs are typically used in modelling computer networks. An edge in a weighted graph is represented as (u, v, w) , where:
- u is the source vertex
 - v is the destination vertex
 - w represents the weight associated to go from u to v

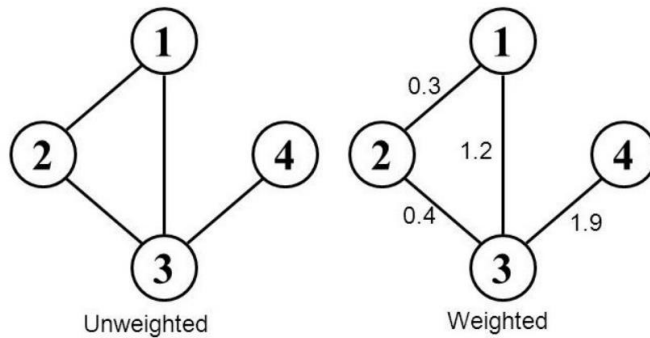


4. Unweighted Graph: A graph in which there is no value or weight associated with the edge.

All the graphs are unweighted by default unless there is a value associated.

An edge of an unweighted graph is represented as (u, v) , where:

- u represents the source vertex
- v is the destination vertex



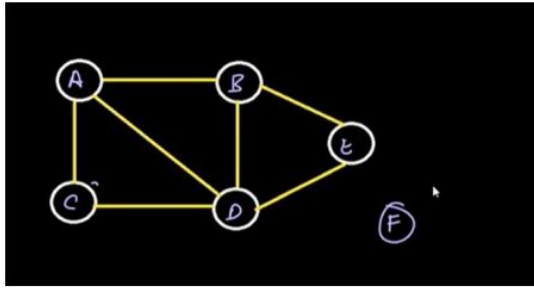
4.Explain different operation of graph.

ANS:

Operations on Graph in Data Structure

Following are the basic graph operations in data structure:

- Add/Remove Vertex – Add or remove a vertex in a graph.
- Add/Remove Edge – Add or remove an edge between two vertices.
- Check if the graph contains a given value.
- Find the path from one vertex to another vertex.



0	A
1	B
2	C
3	D
4	E
5	F

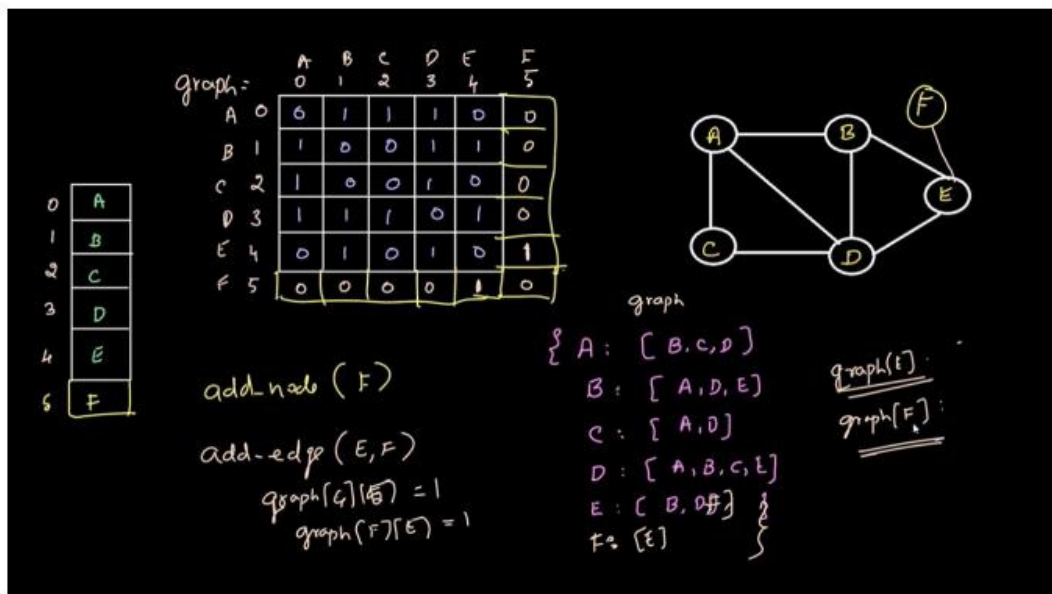
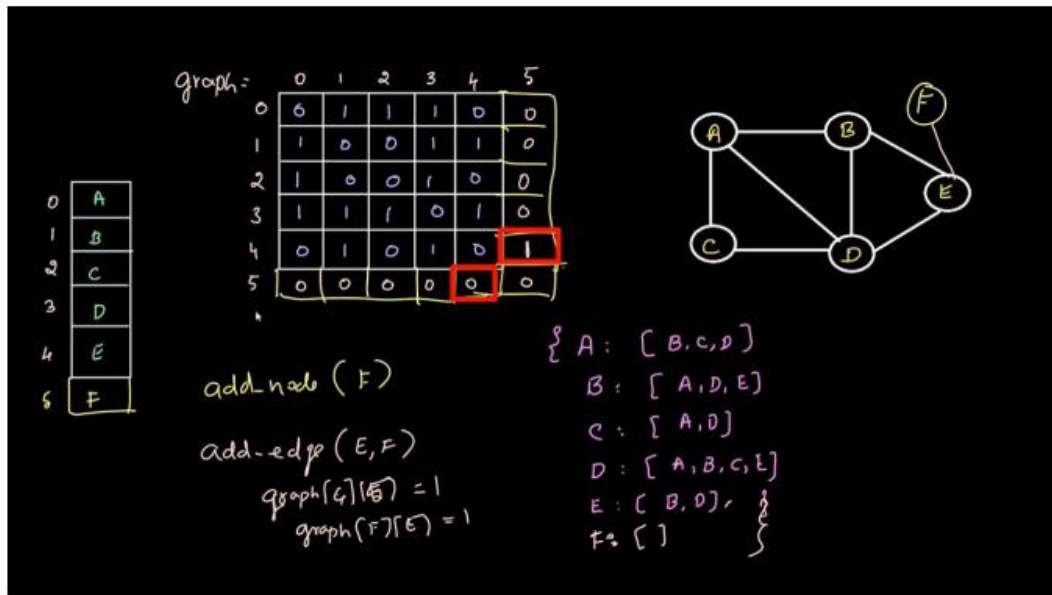
	0	1	2	3	4	5
0	0	1	1	1	0	0
1	1	0	0	1	1	0
2	1	0	0	1	0	0
3	1	1	1	0	1	0
4	0	1	0	1	0	0
5	0	0	0	0	0	0

```

graph LR
    A --- B
    B --- E
    E --- D
    D --- C
    C --- A
    A --- D
    F
  
```

add_node(F)

{ A: [B, C, D]
 B: [A, D, E]
 C: [A, D]
 D: [A, B, C, E]
 E: [B, D]
 F: []



5. Explain BFS.

ANS:

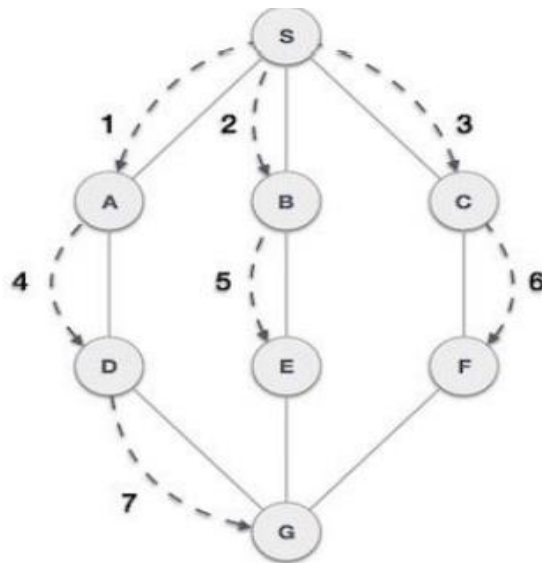
1] Breadth-First Search (BFS) –

It is a traversal operation that horizontally traverses the graph.

It traverses all the nodes at a single level before moving to the next level.

It begins at the root of the graph and traverses all the nodes at a single depth level before moving on to the next depth level.

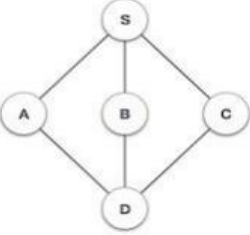
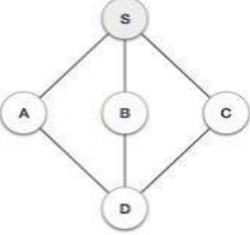
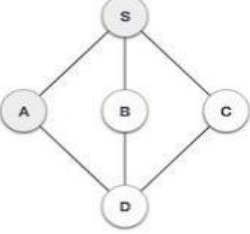
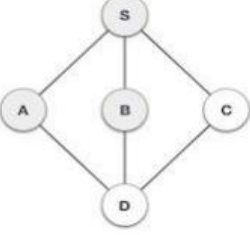
Breadth First Search (BFS) algorithm traverses a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

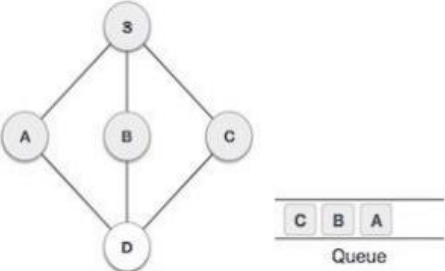
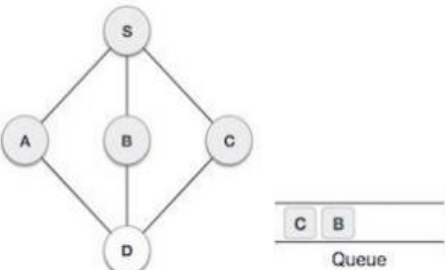
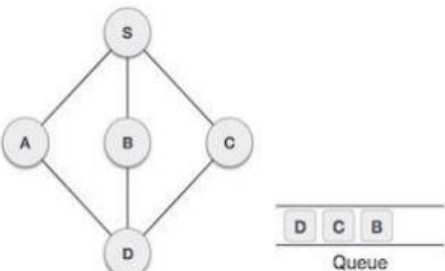


As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

- Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- Rule 2 – If no adjacent vertex is found, remove the first vertex from the queue.
- Rule 3 – Repeat Rule 1 and Rule 2 until the queue is empty.

Step	Traversal	Description
------	-----------	-------------

Step	Traversal	Description
1	 <div> <div></div> <div></div> <div>Queue</div> </div>	Initialize the queue.
2	 <div> <div></div> <div></div> <div>Queue</div> </div>	We start from visiting S (starting node), and mark it as visited.
3	 <div> <div>A</div> <div></div> <div>Queue</div> </div>	We then see an unvisited adjacent node from S . In this example, we have three nodes but alphabetically we choose A , mark it as visited and enqueue it.
4	 <div> <div>B</div> <div>A</div> <div>Queue</div> </div>	Next, the unvisited adjacent node from S is B . We mark it as visited and enqueue it.

5		<p>Next, the unvisited adjacent node from S is C. We mark it as visited and enqueue it.</p>
6		<p>Now, S is left with no unvisited adjacent nodes. So, we dequeue and find A.</p>
7		<p>From A we have D as unvisited adjacent node. We mark it as visited and enqueue it.</p>


```
graph = {
    '5': ['3','7'],
    '3': ['2', '4'],
    '7': ['8'],
    '2': [],
    '4': ['8'],
    '8': []
}

visited = [] # List for visited nodes.
queue = [] #Initialize a queue
```

```
def bfs(visited, graph, node): #function for BFS
    visited.append(node)
    queue.append(node)

    while queue: # Creating loop to visit each node
        m = queue.pop(0)
        print (m, end = " ")

        for neighbour in graph[m]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5') # function calling
```

6.Explain DFS.

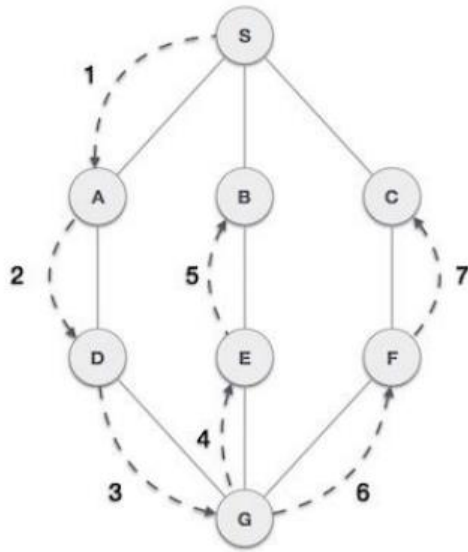
ANS:

2] Depth-First Search (DFS):

This is another traversal operation that traverses the graph vertically.

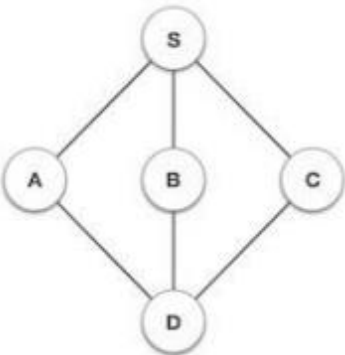

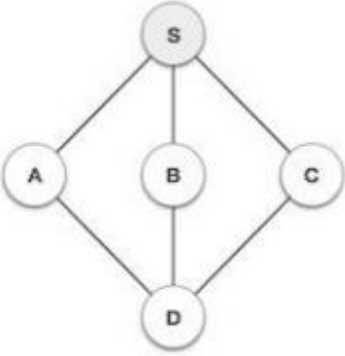

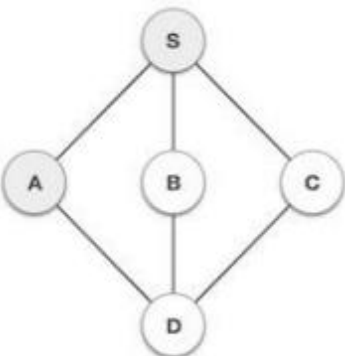
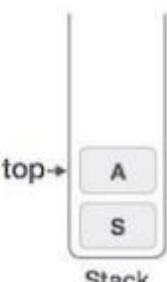
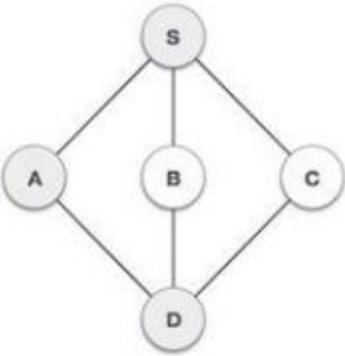
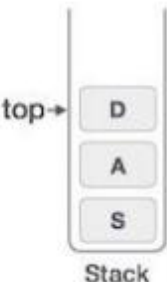
It starts with the root node of the graph and investigates each branch as far as feasible before backtracking.

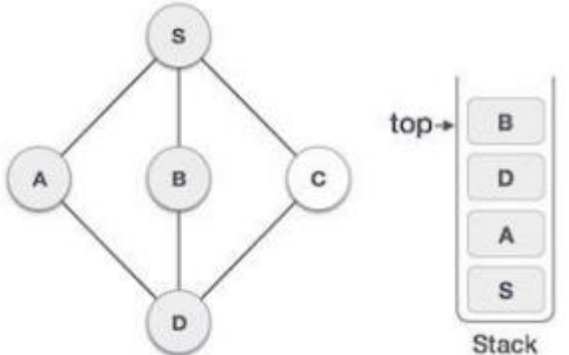
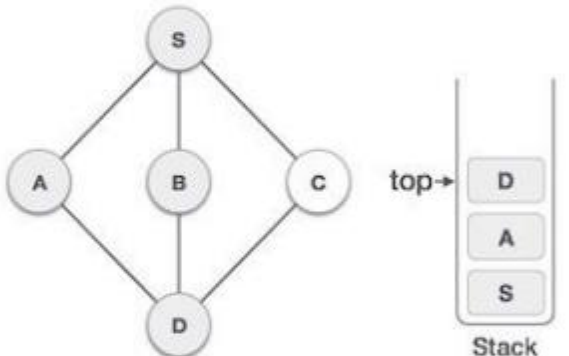
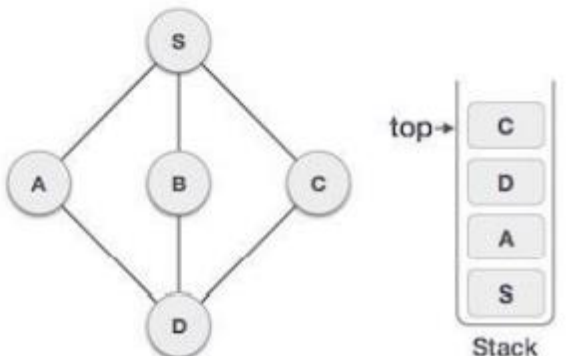
Depth First Search (DFS) algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



As in the example given above, DFS algorithm traverses from S to A to D to G to E to B first, then to F and lastly to C. It employs the following rules.

- Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
- Rule 2 – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
- Rule 3 – Repeat Rule 1 and Rule 2 until the stack is empty.

Step	Traversal	Description
1	  <p style="text-align: center;">Stack</p>	Initialize the stack.
2	  <p style="text-align: center;">Stack</p>	Mark S as visited and put it onto the stack. Explore any unvisited adjacent node from S . We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.
3	  <p style="text-align: center;">Stack</p>	Mark A as visited and put it onto the stack. Explore any unvisited adjacent node from A . Both S and D are adjacent to A but we are concerned for unvisited nodes only.
4	  <p style="text-align: center;">Stack</p>	Visit D and mark it as visited and put onto the stack. Here, we have B and C nodes, which are adjacent to D and both are unvisited. However, we shall again choose in an alphabetical order.

5		<p>We choose B, mark it as visited and put onto the stack. Here B does not have any unvisited adjacent node. So, we pop B from the stack.</p>
6		<p>We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find D to be on the top of the stack.</p>
7		<p>Only unvisited adjacent node is from D is C now. So we visit C, mark it as visited and put it onto the stack.</p>

```
# Using a Python dictionary to act as an adjacency list
graph = {
    '5': ['3', '7'],
    '3': ['2', '4'],
    '7': ['8'],
    '2': [],
    '4': ['8'],
    '8': []
}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node): #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

7.Explain how to create a graph in python.

8. Explain how to perform operations on graph.

9. Explain advantages and disadvantages of graph.

ANS:

Advantages of Graph:

- By using graphs we can easily find the shortest path, neighbors of the nodes, and many more.
- Graphs are used to implement algorithms like DFS and BFS.

- It is used to find minimum spanning tree which has many practical applications.
- It helps in organizing data.
- Because of its non-linear structure, helps in understanding complex problems and their visualization.

Disadvantages of Graph:

- Graphs use lots of pointers which can be complex to handle.
- It can have large memory complexity.
- If the graph is represented with an adjacency matrix then it does not allow parallel edges and multiplication of the graph is also difficult.

10. Explain different parts of graph.

11. Explain undirected graph and its implementation.

12. Explain representation of graph and its implementation.

ANS:

Graph Representation:

Graph can be represented in the following ways:

1. Set Representation:

Set representation of a graph involves two sets:

Set of vertices $V = \{V1, V2, V3, V4\}$ and set of edges $E = \{\{V1, V2\}, \{V2, V3\}, \{V3, V4\}, \{V4, V1\}\}$.

This representation is efficient for memory but does not allow parallel edges.

2. Sequential Representation:

This representation of a graph can be represented by means of matrices: Adjacency Matrix, Incidence matrix and Path matrix.

• Adjacency Matrix:

This matrix includes information about the adjacent nodes. Here, $a_{ij} = 1$ if there is an edge from V_i to V_j otherwise 0. It is a matrix of order $V \times V$.

• Path Matrix:

This matrix includes information about the simple path between two vertices. Here, $P_{ij} = 1$ if there is a path from V_i to V_j otherwise 0. It is also called as reachability matrix of graph G.

3. Linked Representation:

This representation gives the information about the nodes to which a specific node is connected i.e. adjacency lists.

This representation gives the adjacency lists of the vertices with the help of array and linked lists.

In the adjacency lists, the vertices which are connected with the specific vertex are arranged in the form of lists which is connected to that vertex.

13. Explain linked representation of graph.

Linked Representation:

This representation gives the information about the nodes to which a specific node is connected i.e. adjacency lists.

This representation gives the adjacency lists of the vertices with the help of array and linked lists.

In the adjacency lists, the vertices which are connected with the specific vertex are arranged in the form of lists which is connected to that vertex.

14. Explain sequential representation of graph.

Sequential Representation:

This representation of a graph can be represented by means of matrices: Adjacency Matrix, Incidence matrix and Path matrix.

- **Adjacency Matrix:**

This matrix includes information about the adjacent nodes. Here, $a_{ij} = 1$ if there is an edge from V_i to V_j otherwise 0. It is a matrix of order $V \times V$.

- **Path Matrix:**

This matrix includes information about the simple path between two vertices. Here, $P_{ij} = 1$ if there is a path from V_i to V_j otherwise 0. It is also called as reachability matrix of graph G.

15. Explain insert and delete operation of graph.

Adding an edge: Adding an edge is done by inserting both of the vertices connected by that edge in each others list. For example, if an edge between **(u, v)** has to be added, then **u** is stored in **v's vector list** and **v** is stored in **u's vector list**. (push back)

Deleting an edge: To delete edge between **(u, v)**, **u's adjacency list** is traversed until **v** is found and it is removed from it. The same operation is performed for **v**. (erase)

