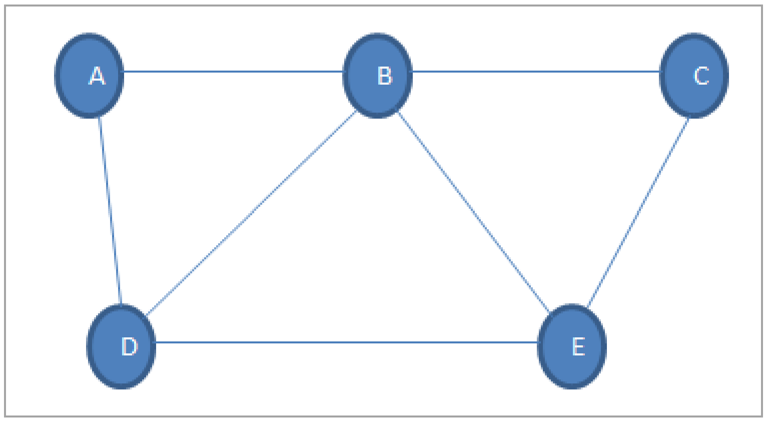
what a Graph is: It is a representation of data in a non-linear structure consisting of nodes (or vertices) and edges (or paths).

A graph is a non-linear data structure. A graph can be defined as a collection of Nodes which are also called “vertices” and “edges” that connect two or more vertices. A graph can also be seen as a cyclic tree where vertices do not have a parent-child relationship but maintain a complex relationship among them.

<u>Following is an example of a graph data structure.</u>



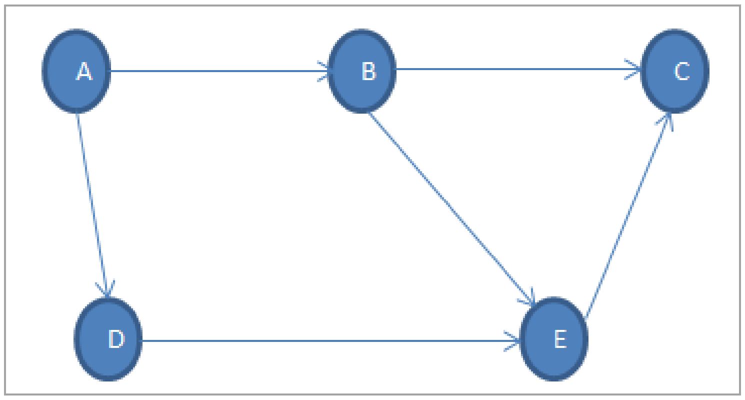
Given above is an example graph G. Graph G is a set of vertices {A,B,C,D,E} and a set of edges {(A,B),(B,C),(A,D),(D,E),(E,C),(B,E),(B,D)}.

Types of Graphs: Directed, Undirected Graph

A graph in which the edges do not have directions is called the Undirected graph. The graph shown above is an undirected graph.

A graph in which the edges have directions associated with them is called a Directed graph.

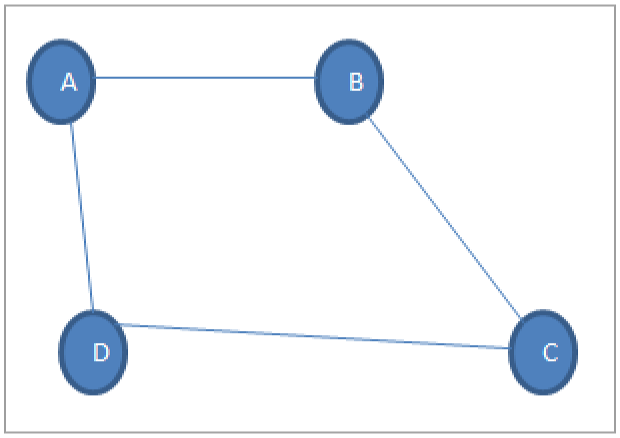
<u>Given below is an example of a directed graph.</u>



In the directed graph shown above, edges form an ordered pair wherein each edge represents a specific path from one vertex to another vertex. The vertex from which the path initiates is called "**Initial Node**" while the vertex into which the path terminates is called the "**Terminal Node**".

Thus in above graph, the set of vertices is {A, B, C, D, E} and the set of edges is {(A,B),(A,D),(B,C),(B,E),(D,E)(E,C)}.

Graph Terminology



1. **Vertex:** Each node of the graph is called a vertex. In the above graph, A, B, C, and D are the vertices of the graph.

2. **Edge:** The link or path between two vertices is called an edge. It connects two or more vertices. The different edges in the above graph are AB, BC, AD, and DC.

3. **Adjacent node:** In a graph, if two nodes are connected by an edge then they are called adjacent nodes or neighbors. In the above graph, vertices A and B are connected by edge AB. Thus A and B are adjacent nodes.

4. **Degree of the node:** The number of edges that are connected to a particular node is called the degree of the node. In the above graph, node A has a degree 2.

5. **Path:** The sequence of nodes that we need to follow when we have to travel from one vertex to another in a graph is called the path. In our example graph, if we need to go from node A to C, then the path would be A->B->C.

6. **Closed path:** If the initial node is the same as a terminal node, then that path is termed as the closed path.

7. **Simple path:** A closed path in which all the other nodes are distinct is called a simple path.

8. **Cycle:** A path in which there are no repeated edges or vertices and the first and last vertices are the same is called a cycle. In the above graph, A->B->C->D->A is a cycle.

9. **Connected Graph:** A connected graph is the one in which there is a path between each of the vertices. This means that there is not a single vertex which is isolated or without a connecting edge. The graph shown above is a connected graph.

10. **Complete Graph:** A graph in which each node is connected to another is called the Complete graph. If N is the total number of nodes in a graph then the complete graph contains N(N-1)/2 number of edges.

11. **Weighted graph:** A positive value assigned to each edge indicating its length (distance between the vertices connected by an edge) is called weight. The graph containing weighted edges is called a weighted graph. The weight of an edge e is denoted by w(e) and it indicates the cost of traversing an edge.

12. **Diagraph:** A digraph is a graph in which every edge is associated with a specific direction and the traversal can be done in specified direction only.

Graph Representation/Storing

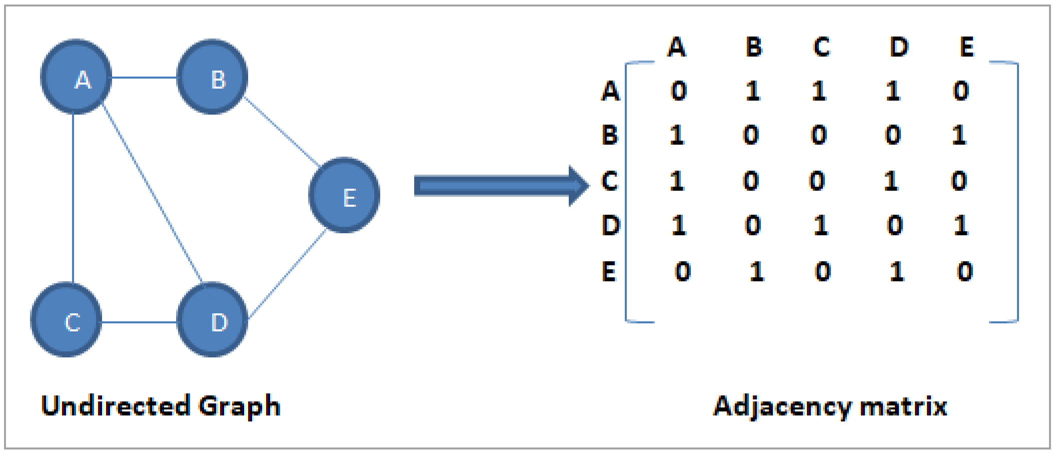
The way in which graph data structure is stored in memory is called “representation”. The graph can be stored as a sequential representation or as a linked representation.

Sequential Representation

In the sequential representation of graphs, we use the adjacency matrix. An adjacency matrix is a matrix of size n x n where n is the number of vertices in the graph.

The rows and columns of the adjacency matrix represent the vertices in a graph. The matrix element is set to 1 when there is an edge present between the vertices. If the edge is not present then the element is set to 0.

<u>Given below is an example graph that shows its adjacency matrix.</u>

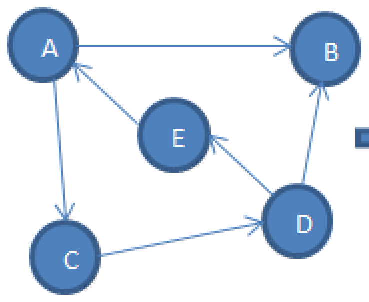


$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 1 & 1 & 0 \\ B & 1 & 0 & 0 & 0 & 1 \\ C & 1 & 0 & 0 & 1 & 0 \\ D & 1 & 0 & 1 & 0 & 1 \\ E & 0 & 1 & 0 & 1 & 0 \end{array}$$

Note that since this is an undirected graph, and the edge is present in both directions. **<u>For Example,</u>** as edge AB is present, we can conclude that edge BA is also present.

In the adjacency matrix, we can see the interactions of the vertices which are matrix elements that are set to 1 whenever the edge is present and to 0 when the edge is absent.

Now let us see the adjacency matrix of a directed graph.



Directed Graph

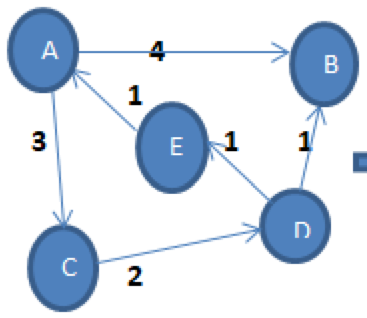
$$\begin{array}{c|ccccc} & A & B & C & D & E \\ A & 0 & 1 & 1 & 0 & 0 \\ B & 0 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 1 & 0 \\ D & 0 & 1 & 0 & 0 & 1 \\ E & 1 & 0 & 0 & 0 & 0 \end{array}$$

Adjacency matrix

As shown above, the intersection element in the adjacency matrix will be 1 if and only if there is an edge directed from one vertex to another.

In the above graph, we have two edges from vertex A. One edge terminates into vertex B while the second one terminates into vertex C. Thus in adjacency matrix the intersection of A & B is set to 1 as the intersection of A & C.

Given below is the weighted graph and its corresponding adjacency matrix.



Weighted Graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ A & 0 & 4 & 3 & 0 & 0 \\ B & 0 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 2 & 0 \\ D & 0 & 1 & 0 & 0 & 1 \\ E & 1 & 0 & 0 & 0 & 0 \end{array}$$

Adjacency matrix

We can see that the sequential representation of a weighted graph is different from the other types of graphs. Here, the non-zero values in the adjacency matrix are replaced by the actual weight of the edge.

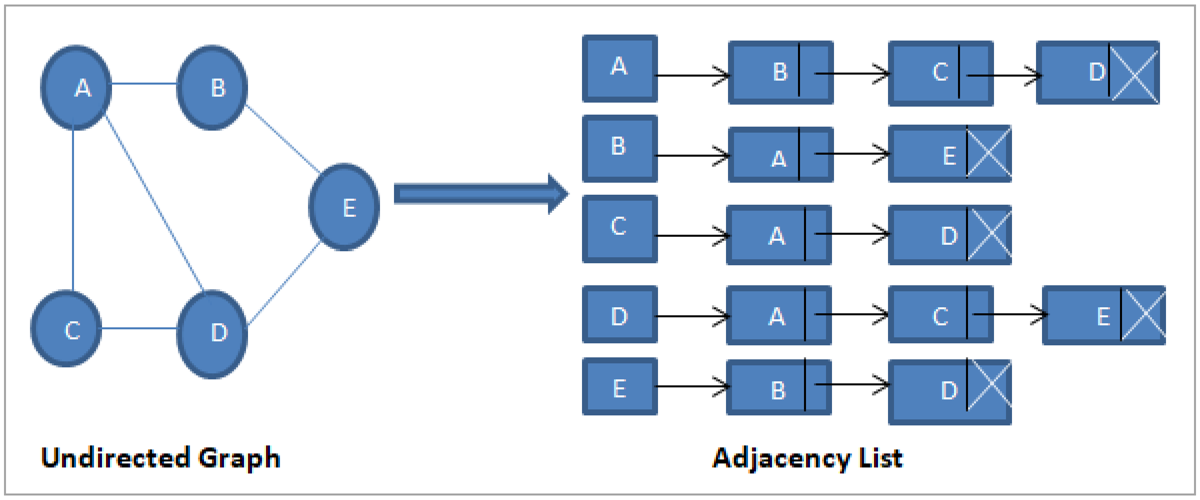
The edge AB has weight = 4, thus in the adjacency matrix, we set the intersection of A and B to 4. Similarly, all the other non-zero values are changed to their respective weights.

Whether the graph is sparse (fewer edges) or dense, it always takes more amount of space.

Linked Representation

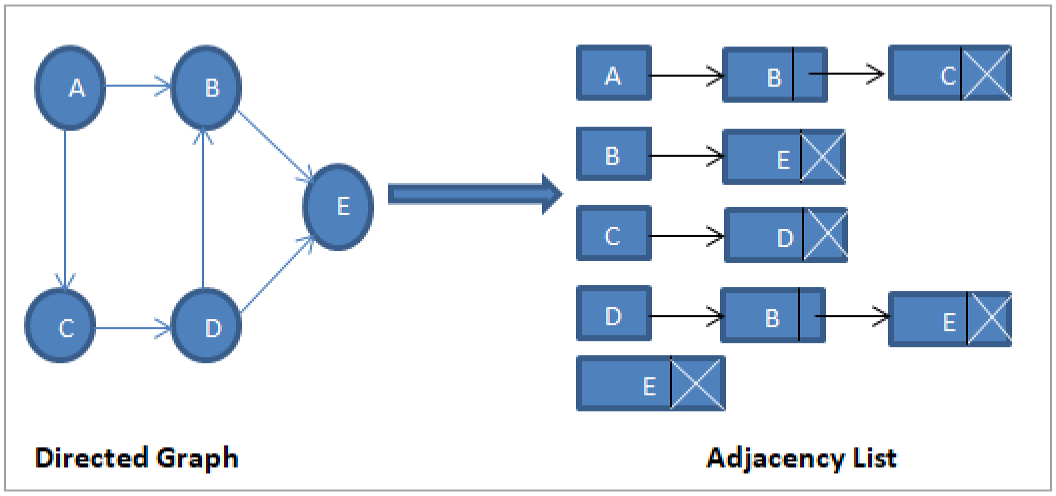
We use the adjacency list for the linked representation of the graph. The adjacency list representation maintains each node of the graph and a link to the nodes that are adjacent to this node. When we traverse all the adjacent nodes, we set the next pointer to null at the end of the list.

Let us first consider an undirected graph and its adjacency list.



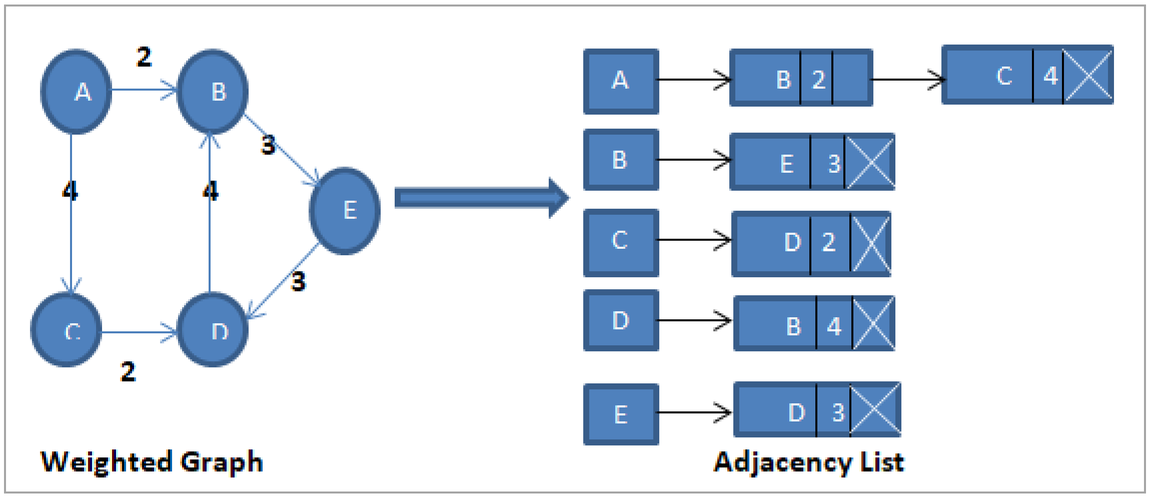
As shown above, we have a linked list (adjacency list) for each node. From vertex A, we have edges to vertices B, C and D. Thus these nodes are linked to node A in the corresponding adjacency list.

Next, we construct an adjacency list for the directed graph.



In the above-directed graph, we see that there are no edges originating from vertex E. Hence the adjacency list for vertex E is empty.

Now let us construct the adjacency list for the weighted graph.



For a weighted graph, we add an extra field in the adjacency list node to denote the weight of the edge as shown above.

Adding vertex in the adjacency list is easier. It also saves space due to the linked list implementation. When we need to find out if there is an edge between one vertex to another, the operation is not efficient.

Basic Operations For Graphs

Following are the basic operations that we can perform on the graph data structure:

- **Add a vertex:** Adds vertex to the graph.
- **Add an edge:** Adds an edge between the two vertices of a graph.
- **Display the graph vertices:** Display the vertices of a graph.

Graph Traversal

Graph traversal is a method used to search nodes in a graph. The graph traversal is used to decide the order used for node arrangement. It also searches for edges without making a loop, which means all the nodes and edges can be searched without creating a loop.

There are two graph traversal structures.

1. DFS (Depth First Search): In-depth search method

The DFS search begins starting from the first node and goes deeper and deeper, exploring down until the targeted node is found. If the targeted key is not found, the search path is changed to the path that was stopped exploring during the initial search, and the same procedure is repeated for that branch.

The spanning tree is produced from the result of this search. This tree method is without the loops. The total number of nodes in the stack data structure is used to implement DFS traversal.

Steps followed to implement DFS search:

Step 1 – Stack size needs to be defined depending on the total number of nodes.

Step 2 – Select the initial node for transversal; it needs to be pushed to the stack by visiting that node.

Step 3 – Now, visit the adjacent node that is not visited before and push that to the stack.

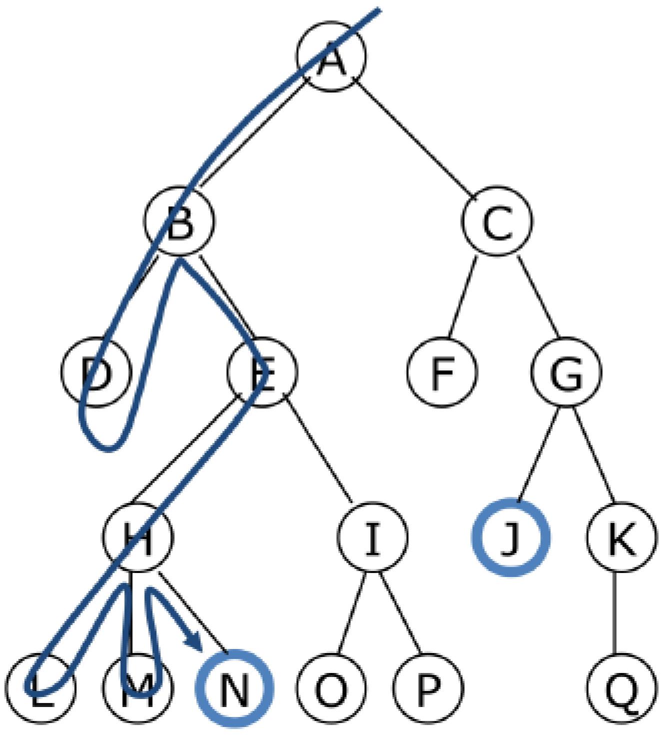
Step 4 – Repeat Step 3 until there is no adjacent node that is not visited.

Step 5 – Use backtracking and one node when there are no other nodes to be visited.

Step 6 – Empty the stack by repeating steps 3,4, and 5.

Step 7 – When the stack is empty, a final spanning tree is formed by eliminating unused edges.

Depth-first searching



- A depth-first search (DFS) explores a path all the way to a leaf before backtracking and exploring another path
- For example, after searching A, then B, then D, the search backtracks and tries another path from B
- Node are explored in the order A B D E H L M N I O P C F G J K Q
- N will be found before J

Applications of DFS are:

Solving puzzles with only one solution.

To test if a graph is bipartite.

Topological Sorting for scheduling the job and many others.

2. BFS (Breadth-First Search): Search is implemented using a queuing method

Breadth-First Search navigates a graph in a breadth motion and utilises based on the Queue to jump from one node to another, after encountering an end in the path.

Steps followed to implement BFS search,

Step 1 – Based on the number of nodes, the Queue is defined.

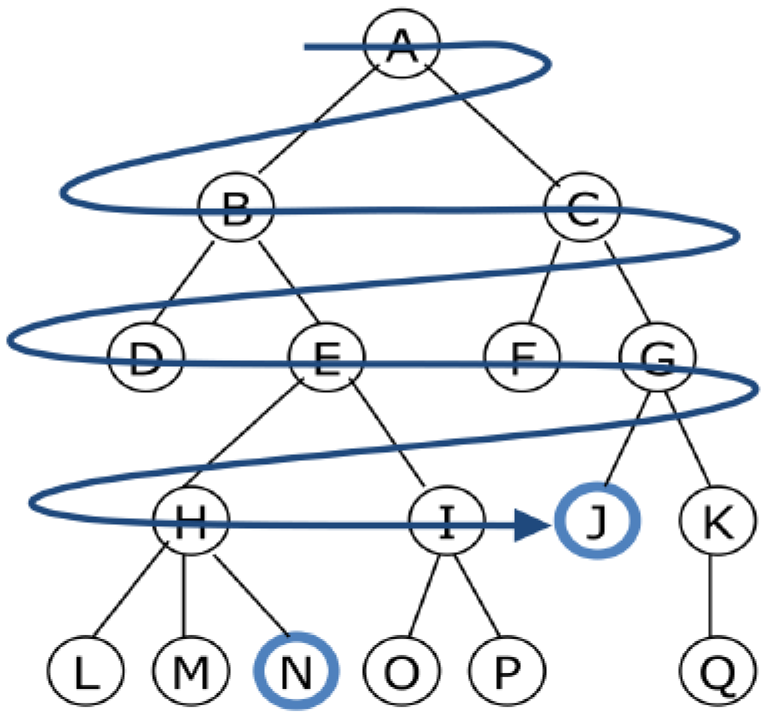
Step 2 – Start from any node of the traversal. Visit that node and add it to the Queue.

Step 3 – Now check the non-visited adjacent node, which is in front of the Queue, and add that into the Queue, not to the start.

Step 4 – Now start deleting the node that doesn't have any edges that need to be visited and is not in the Queue.

Step 5 – Empty the Queue by repeating steps 4 and 5.

Breadth-first searching:



- A breadth-first search (BFS) explores nodes nearest the root before exploring nodes further away
- For example, after searching A, then B, then C, the search proceeds with D, E, F, G
- Node are explored in the order A B C D E F G H I J K L M N O P Q
- J will be found before N

How to do breadth-first searching:

```
Put the root node on a queue;
while (queue is not empty) {
   remove a node from the queue;
   if (node is a goal node) return success;
   put all children of node onto the queue;
}
return failure;
```

Applications of BFS are:

- Peer to Peer Networks- Like in Bittorrent, it is used to find all adjacent nodes.
- Crawlers in Search Engine.
- Social Networking Websites and many more.

Real-world Applications of Graph in the Data Structure

- Graphs are used in many day-to-day applications like network representation (roads, optical fibre mapping, designing circuit board, etc.). Ex: In the Facebook data network, nodes represent the user, his/her photo or comment, and edges represent photos, comments on the photo.

The Graph in data structure has extensive applications. Some of the notable ones are:

- **Social Graph APIs**– It is the primary way the data is communicated in and out of the Facebook social media platform. It is an HTTP-based API, which is used to programmatically query data, upload photos and videos, make new stories, and many other tasks. It is composed of nodes, edges, and fields; to query, the specific object nodes are used. Edges for a group of objects subjected to a single object and fields are used to fetch data about each object among the group.
- **Yelp's GraphQL API**– It's a recommendation engine used to fetch the specific data from the Yelp platform. Here, orders are used to find the edges, after which the specific node is queried to fetch the exact result. This speeds up the retrieval process.

  On the Yelp platform, the nodes represent the business, containing id, name, is_closed, and many other graph properties.
- **Path Optimization Algorithms-** They are employed to find the best connection which fits the criteria of speed, safety, fuel, etc. BFS is used in this algorithm. The best example is Google Maps Platform (Maps, Routes APIs).
- **Flight Networks-** In flight networks, this is used to find the optimised path that fits the graph data structure. This also aids in the model and optimises airport procedures efficiently.

LAB TASK:

Implement graph data structure using adjacency list

Implement graph data structure using adjacency Matrix

Implement all basic functions:

- **Add a vertex:** Adds vertex to the graph.
- **Add an edge:** Adds an edge between the two vertices of a graph.
- **Display the graph vertices:** Display the vertices of a graph.
- Traversal through dfs,bfs.