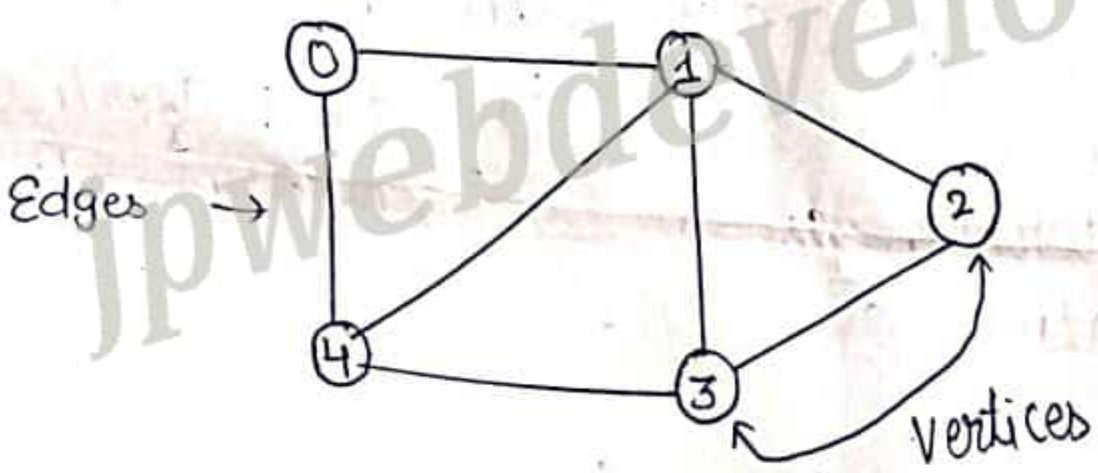
Graph (Data Structure Notes)

Introduction :-

- A graph can be defined as a group of vertices and edges that are used to connect these vertices.

Definition :- A graph G can be defined as an ordered set G (V, E) where V (G) represents the set of vertices and E (G) represents the set of edges which are used to connect these vertices.

> A Graph consists of a finite set of vertices (or nodes) and set of Edges which connect a pair of nodes.



→ In the above Graph, the set of vertices V = {0, 1, 2, 3, 4} and set of edges E = {01, 12, 23, 34, 04, 13, 14}

A Graph is a very important non-linear data Structure in which each node may have multiple successors as well as multiple Predecessors.

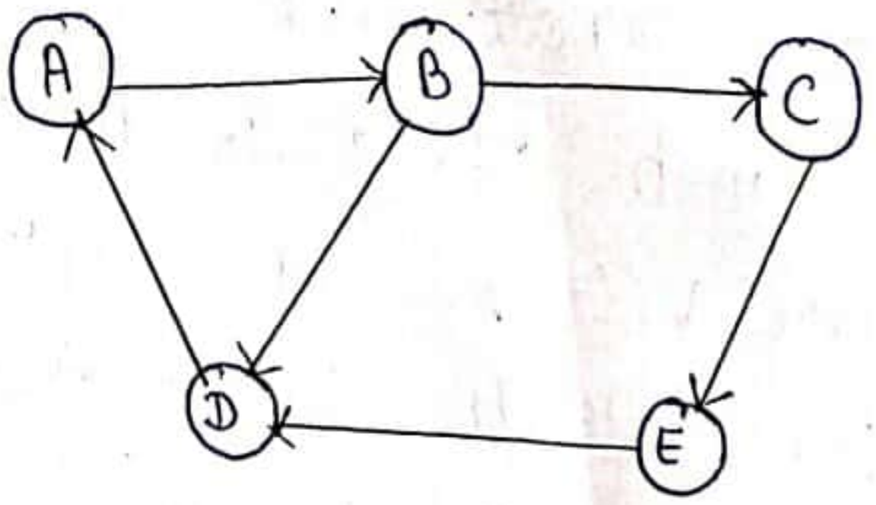
A Graph may be directed or Undirected

Directed Graph :- A directed graph or digraph is a graph, whose each edge is an ordered pair of vertices.

→ In this, Edges form an ordered pair.

→ Edges represent a specific path from some vertex A to some another vertex B.

(Node A is called intial Node while node B is called terminal Node).

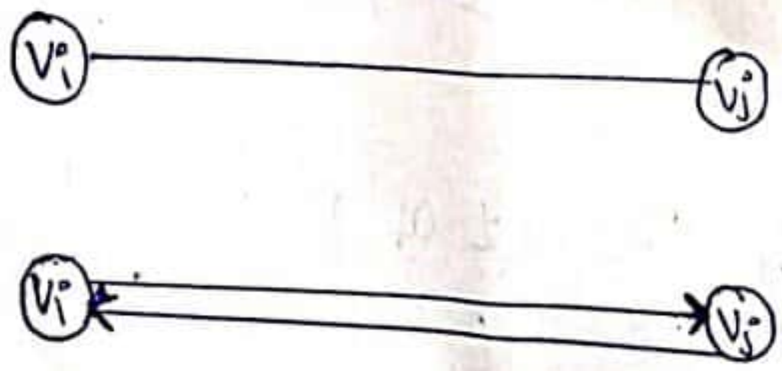


Undirected Graph:-

In undirected graph, edges are not associated with the directions with them.

→ There is no direction associated with any of the edges.

→ An edge in an undirected graph connecting vertics vi and vj can be written as (Vi, Vj) or (Vj, Vi). as it can traverse in either direction.



* Graph Terminology:-

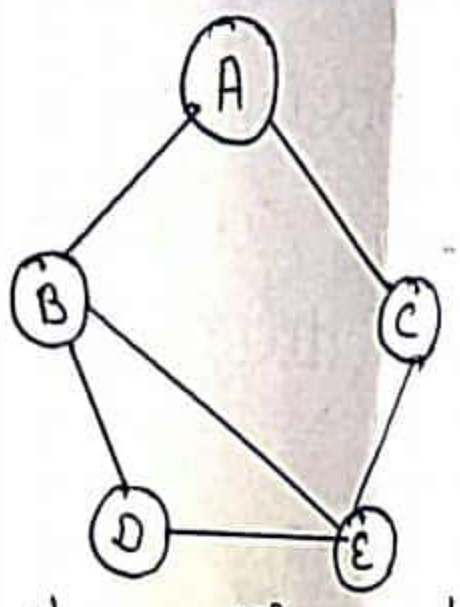
1. Path:- A path can be defined as the sequence of nodes that are followed in order to reach some terminal node V from the intial node U.

2. Closed Path:- A path will be called as closed path if the ~~int~~ initial node is same as terminal node. A Path will be closed path if $V_0 = V_N$.

3. Degree of VERTEX :- The number of edges connected with Vertex v_i is called the degree of Vertex v_i.

→ It is denoted by degree(v_i).

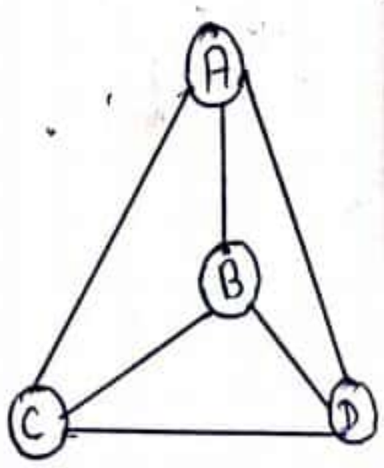
→ for example:-



degree (A) = 2, degree (B) = 3, degree (C) = 2, degree (D) = 2, degree (E) = 2.

(4) Cycle :- A Cycle can be defined as the path which has no repeated edges or vertices except the first and last vertices

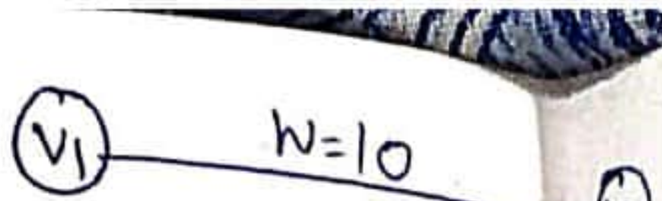
(5) Complete Graph :- A graph is said to be complete, if there are edges from any vertex to all other vertices.



(6) Connected Graph :- If every vertex is reachable from others by following some path.

→ In which some path exists between every two vertices (u, v) in V.

(7) Weighted Graphs :- In a weighted graph, each edge is assigned with some data such as weight.



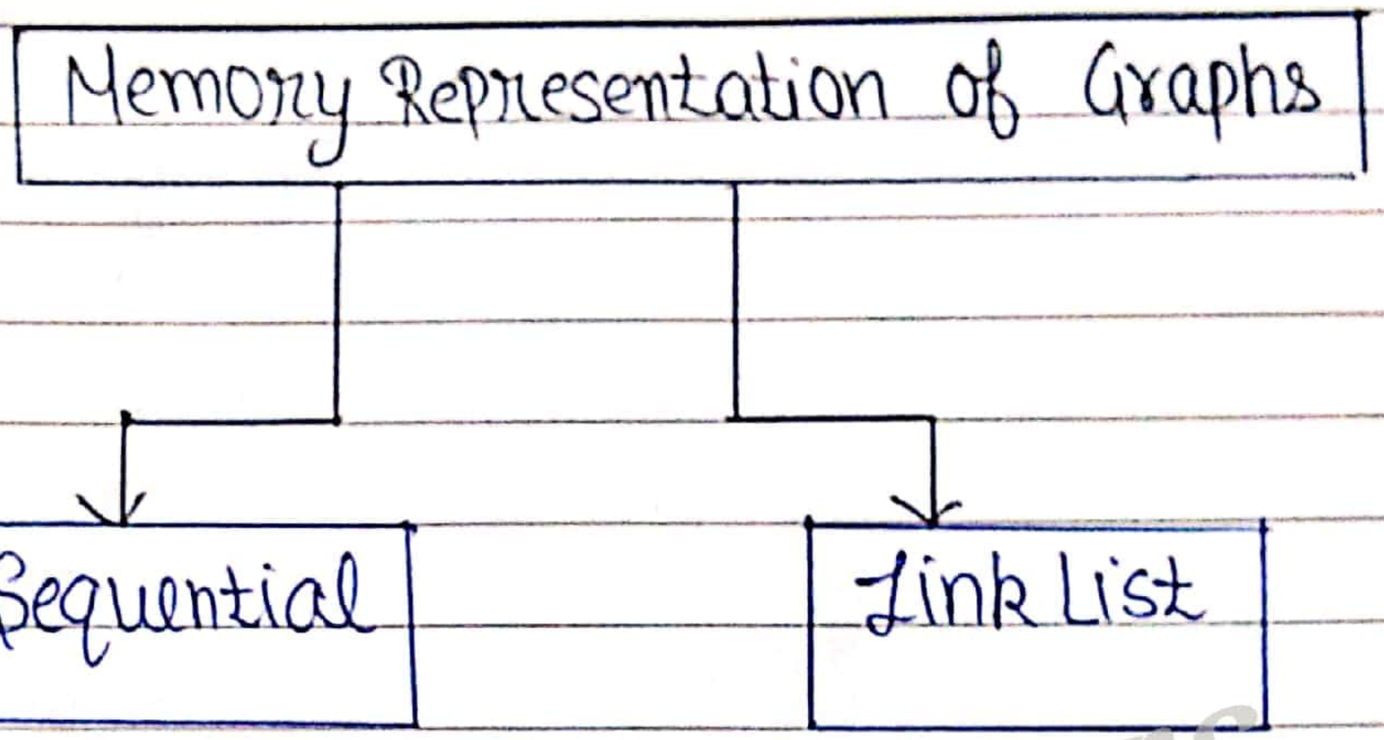
(8) Adjacent Nodes:- If two nodes u and v are connected via an edge e, then the nodes u and v are called as adjacent nodes.

(9) Loop:- An edge that is associated with the similiar end points can be called as loop.

10). Multigraph:- It may be possible that there are two or more edges connecting the same vertices of graph. such a graph structure is called multigraph.

* Representation to Graphs :-

There are two standard ways :-



1 Sequential Representation :-

- Sequential Representation methods use Matrix or 2-D array to represent the information of Graph in computer memory.
- Two ways to represent graph in sequential manner.

__/__/____

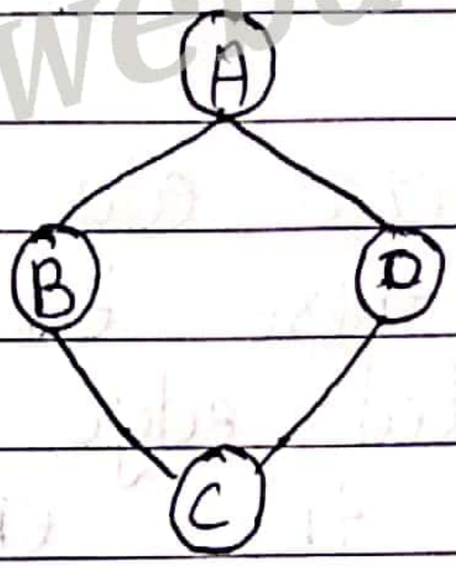
(a) Adjacency Matrix

- A Graph is represented by using a Square Matrix.

- In Adjacency Matrix, the rows and columns are represented by graph vertices.

- A graph having n vertices, will have a dimension nxn.

Undirected Graph



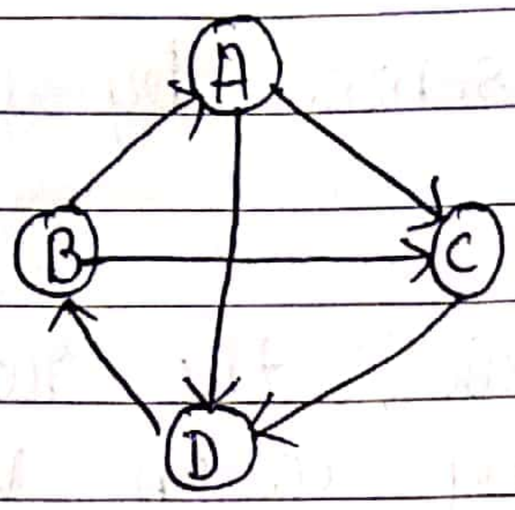
The adjacency matrix

	A	B	C	D
A	0	1	0	1
B	1	0	1	0
C	0	1	0	1
D	1	0	1	0

- There are four vertices hence the size of matrix is 4X4 (four rows, four columns)
- In this, we can see mapping between the vertices (A, B, C, D) is represented by adjacency matrix

__/__/____

Directed Graph :-



adjacency matrix

	A	B	C	D
A	0	0	1	1
B	1	0	1	0
C	0	0	0	1
D	0	1	0	0

- As there is a directed edge between A to C and A to D, so these entries are 1 and there is no directed edge between A to A and A to B so these entries are 0 and so on.

(2) Incidence Matrix

	e1	e2	e3	e4	e5	e6
V1	1	1	0	0	0	0
V2	1	0	1	1	0	0
V3	0	1	1	0	1	0
V4	0	0	0	1	1	1
V5	0	0	0	0	0	1

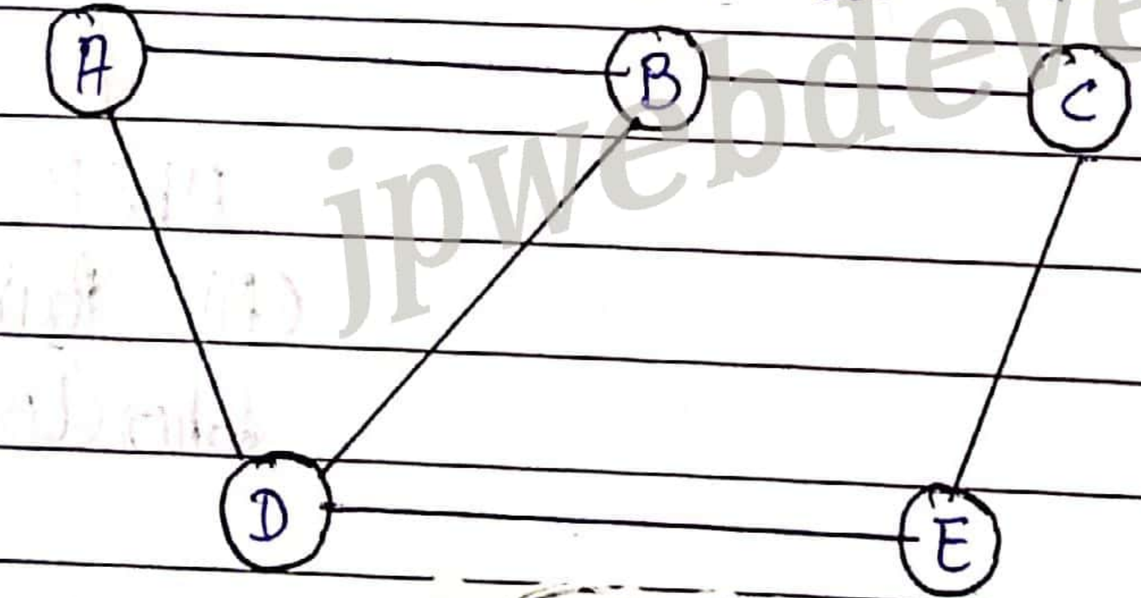
__/__/____

(2) Linked Representation :-

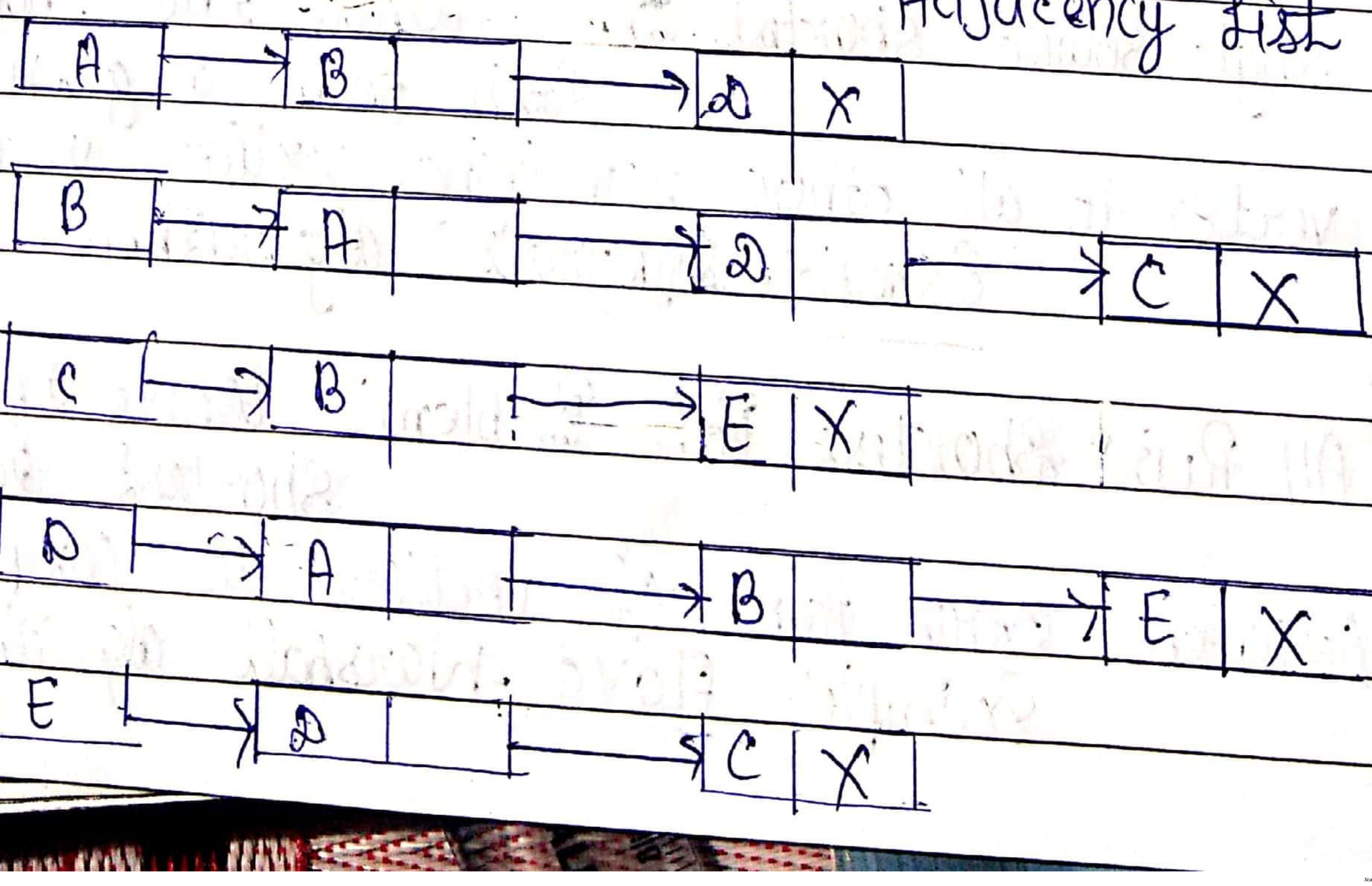
- In the Linked Representation, an adjacency list is used to store the Graph into the memory.
- In linked representation, we store a graph as a linked structure.
- We store all the vertices in a list and then for each vertex, we have a linked list of its adjacent vertices.

Example :-

(Undirected Graph)



Adjacency list



__/__/____

* Shortest Path Algorithms :-

Shortest Path Problem is a problem of finding the shortest path(s) between vertices of a given graph.

Shortest Path Algorithm :- Shortest Path algorithms are a family of algorithms used for solving the shortest path problem.

Types of Path Problems

- SSSP (Single Source Shortest Path)
- APSP (All Pair Shortest Path)

1. Single Source shortest Path :- Where the Shortest Path from a given source vertex to all other remaining vertices is computed.
   Example :- Dijkstra's algorithm.

2. All Pair Shortest Path Problem :- Where the Shortest path between every pair of vertices is compited.
   Example :- Floyd-Warshall algorithm.

__/__/____

Operations

* Ways to find the Shortest Path in a Graph

(a) Breadth first search > Traversing
(b) Depth first search

(a) Breadth first search (BFS):-

- It is a vertex based technique for finding a shortest path in a graph.
- It uses a Queue data structure. (FIFO).
- In BFS, one vertex is selected at a time which is visited and marked then its adjacent are visited and stored in the Queue.

Algorithm:-

Step 1: SET STATUS = 1 (ready state) for each node in G.

Step 2:- Enqueue the starting node A and set its ~~sta~~ status to waiting status (STATUS = 2)

__/__/____

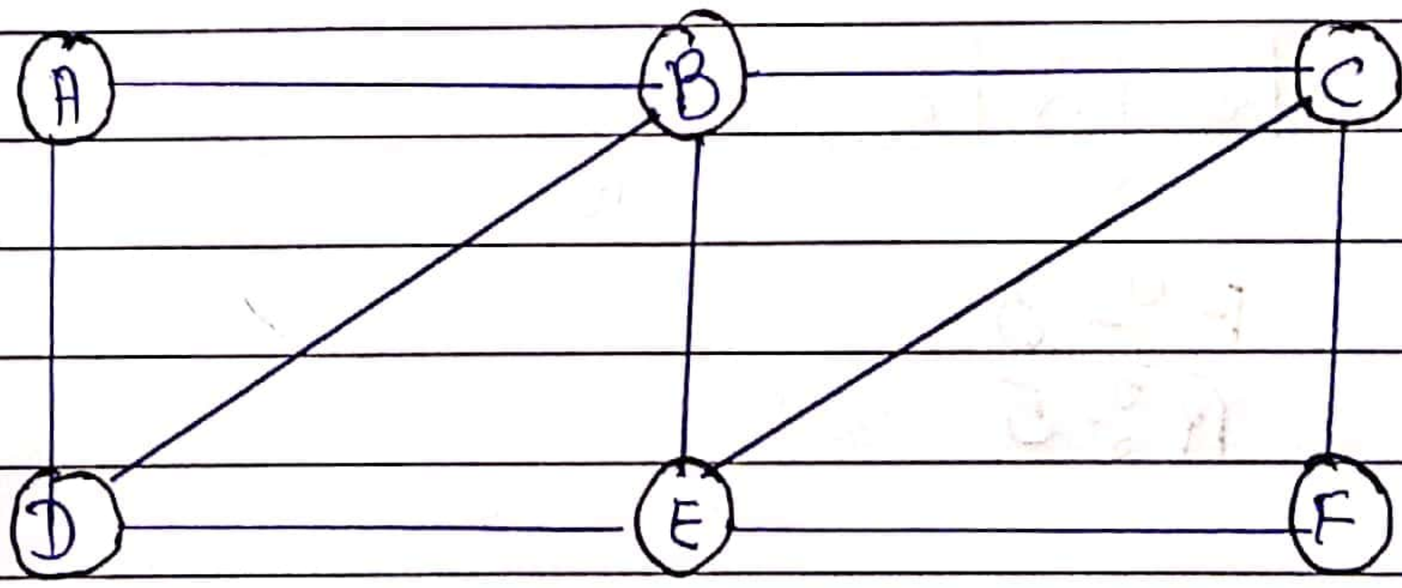
Step3 Repeat Step 4 and 5 until Queue is empty.

Step4 Dequeue a node N. Process n and set its STATUS = 3 (processed state)

Step5 Enqueue all the neighbours of N that are in ready state (STATUS = 1) and set their STATUS = 2 (Waiting state)

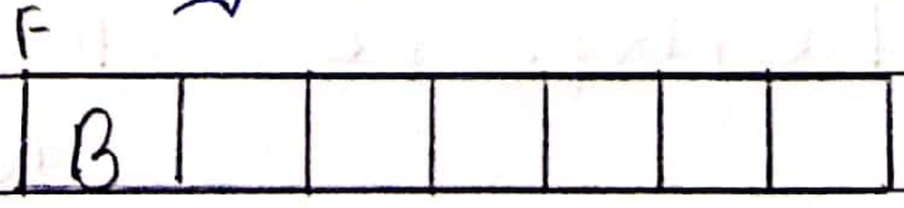
Step 6 Exit.

Example of BFS Algorithm



Explanation:- Assume B as the starting vertex.

Step I:- Initially put B to the queue.

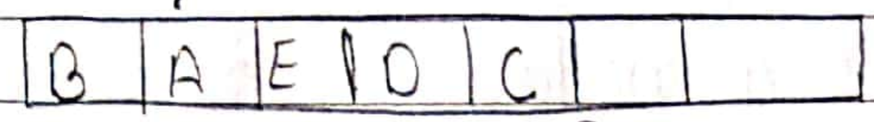


F = 0

R = 0

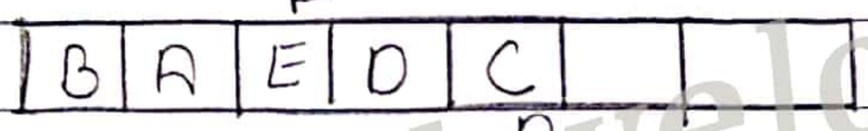
__/__/____

Step :-2 Remove the front element B from queue By setting front = front + 1 and add neighbours of B to the queue. becomes :-



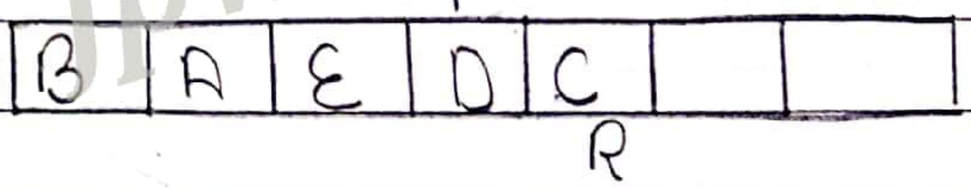
F = 1
R = 4

Step :-3 Remove the front element A from the queue by setting front :- front + 1 and add neighbours of A to the queue becomes :-



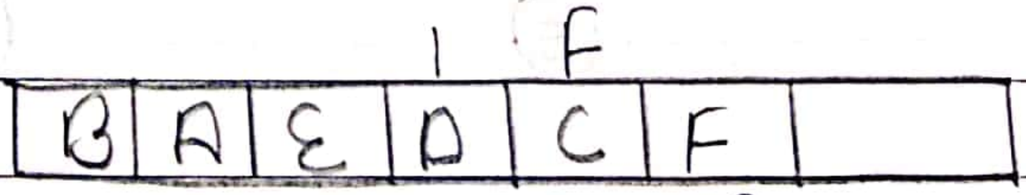
F = 2
R = 4

Step :-4



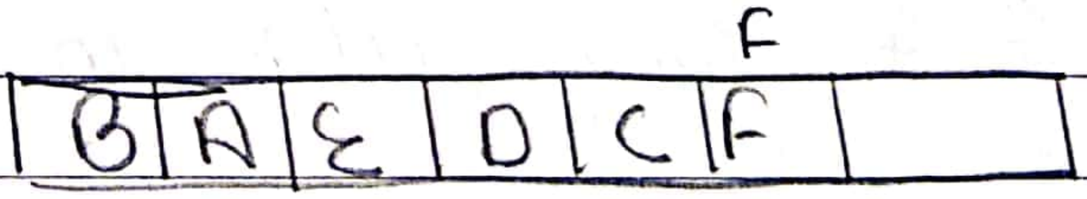
F :- 3
R :- 5

Step :-5



F :- 4
R :- 5

Step :-6



F :- 5
R :- 5

Step :-7 Remove the front element F so now queue becomes empty and each and every vertex is traversed. Order is BAEDCF.

__/__/____

(2) Depth first Search (DFS):-

- The DFS traversal algorithm visits each node in a graph.

- The DES algorithm starts with the intial node of the graph G, and then goes to deeper and deeper until we find the goal node or node which has no children.

- The data structure which is being used in DFS is **Stack**

1. Pop a node off the stack.
2. Traverse or Print the current node.
3. Push all nodes adjacent to the current node onto the stack.

Algorithm

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)

__/__/____

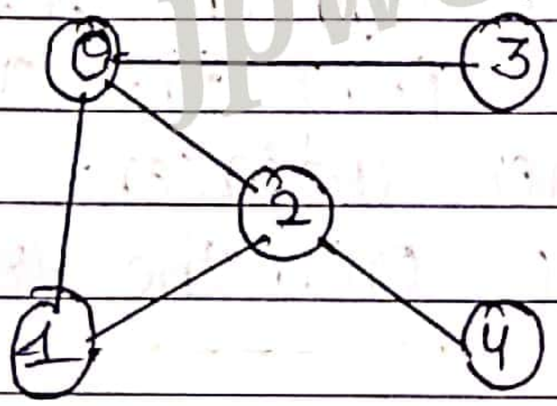
Step 3 Repeat Steps 4 and 5 until [STACK] is empty

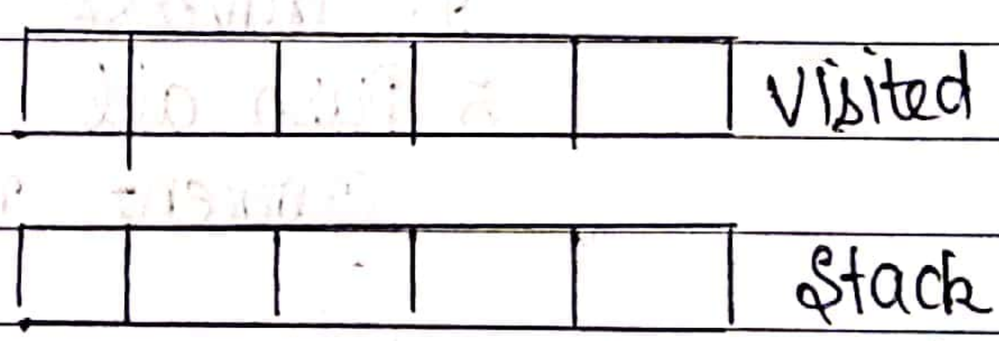
Step 4 Pop the top node N. Process and set its STATUS = 3 (processed state)

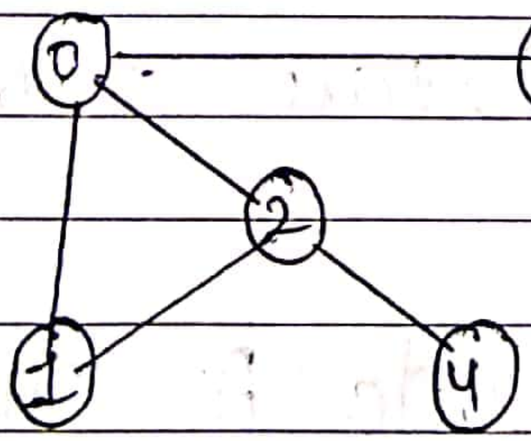
Step 5 Push on the stack all the neighbours of N that are in the ready state (1) and set their STATUS = 2 (waiting status)

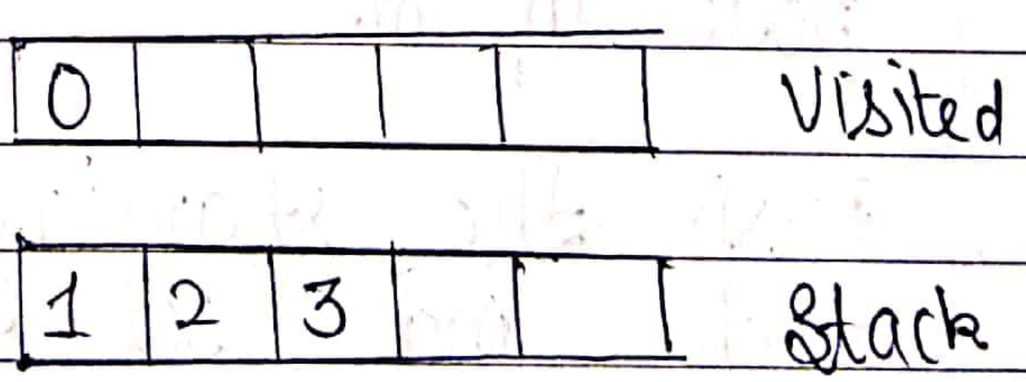
Step 6 Exit.

Example





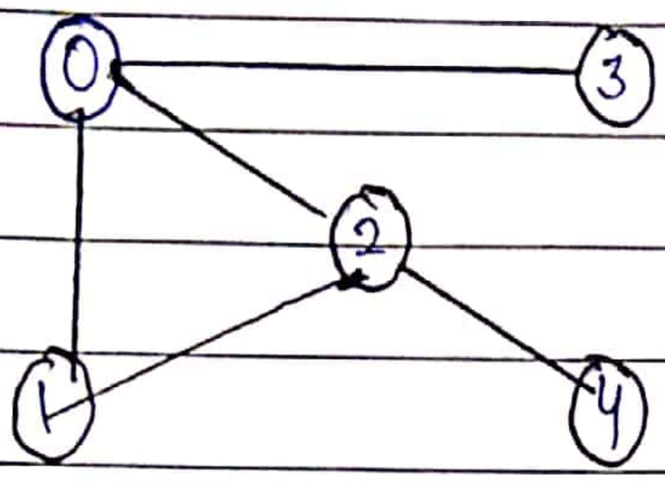


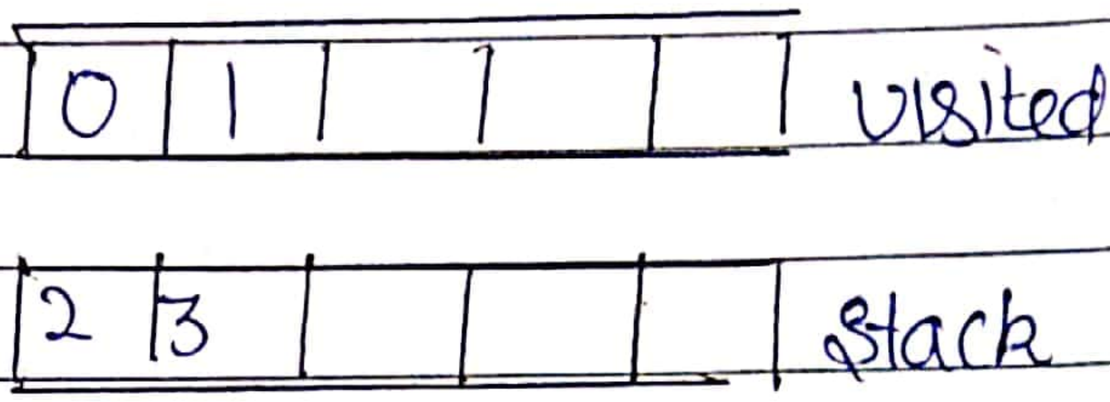


(Visit the element and put it in visited list

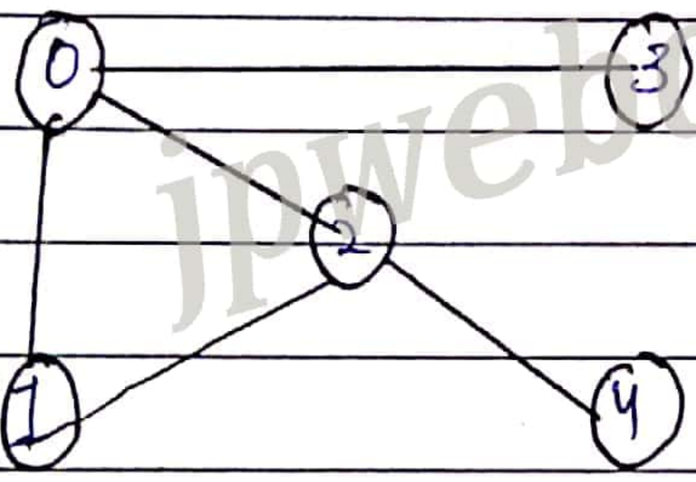
__/__/____

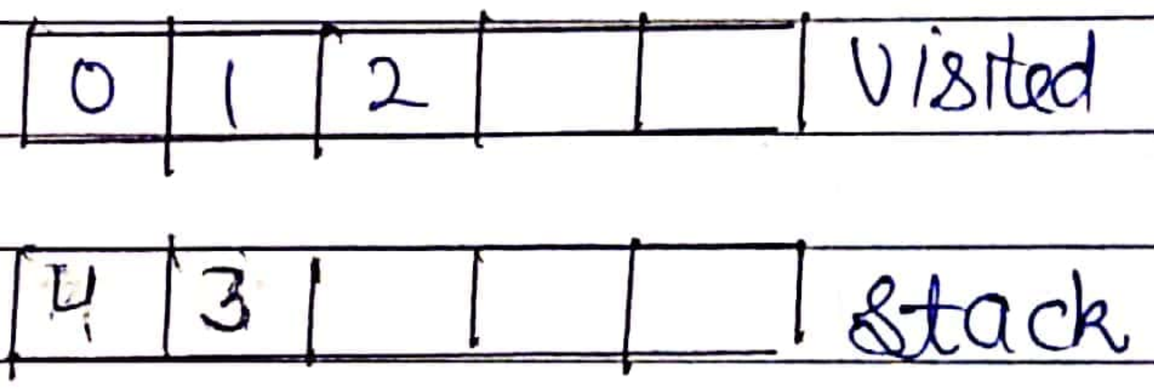
Now, we visit the element at the top of stack i.e. 1 and go to adjacent nodes.





Vertex 2 has an unvisited adjacent vertex in 4, so we add to the top of stack and visit it:





→ add to the stack and visit it.