Unit III: Graphs: Operations on Graphs: Vertex insertion, vertex deletion, find vertex, edge addition, edge deletion, Graph Traversals- Depth First Search and Breadth First Search (Non recursive) .Graph storage Representation- Adjacency matrix, adjacency lists.

<u>Graph</u>: - A graph is data structure that consists of following two components.

- A finite set of vertices also called as nodes.
- A finite set of ordered pair of the form (u, v) called as edge.

(or)

A graph G=(V, E) is a collection of two sets V and E, where

V→ Finite number of vertices

E→ Finite number of Edges,

Edge is a pair (v, w), where v, w ∈ V.

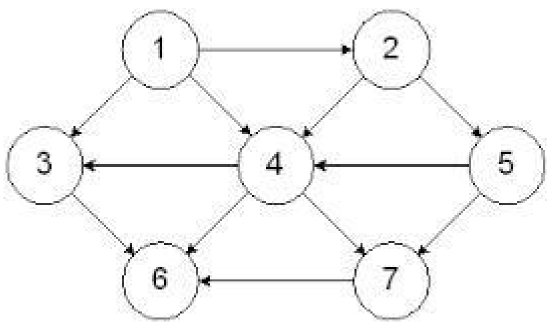
<u>Application of graphs:</u>

- Coloring of MAPS
- Representing network
  - Paths in a city
  - Telephone network
  - Electrical circuits etc.
- It is also using in social network including
  - LinkedIn
  - Facebook

<u>Types of Graphs:</u>

- Directed graph
- Undirected Graph

Directed Graph:



In representing of graph there is a directions are shown on the edges then that graph is called Directed graph.

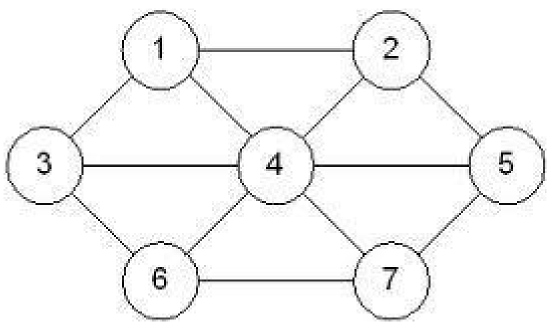
That is,

A graph G=(V, E) is a directed graph ,Edge is a pair (v, w), where v, w ∈ V, and the pair is ordered.

Means vertex 'w' is adjacent to v.

Directed graph is also called digraph.

Undirected Graph:



In graph vertices are not ordered is called undirected graph. Means in which (graph) there is no direction (arrow head) on any line (edge).

A graph G=(V, E) is a directed graph ,Edge is a pair (v, w), where v, w ∈ V, and the pair is not ordered.

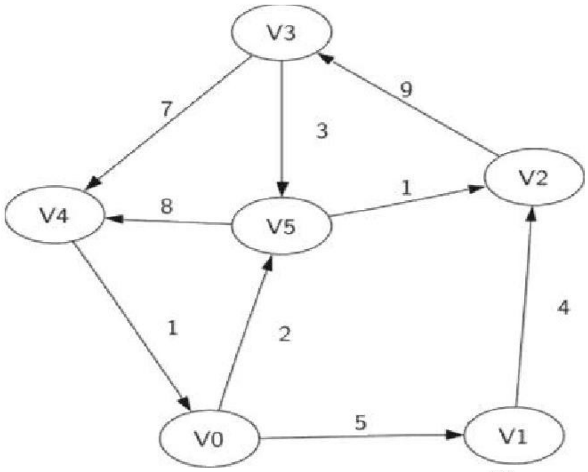
Means vertex 'w' is adjacent to 'v', and vertex 'v' is adjacent to 'w'

Note: in graph there is another component called weight/ cost.

Weight graph:

Edge may be weight to show that there is a cost to go from one vertex to another.

Example: In graph of roads (edges) that connect one city to another (vertices), the weight on the edge might represent the distance between the two cities (vertices).

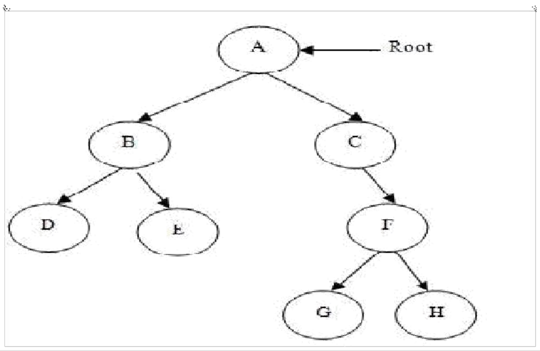


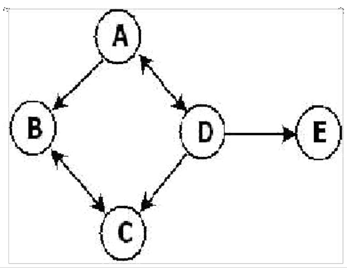
$$V = \{V0, V1, V2, V3, V4, V5\}$$

$$E = \left\{ \begin{array}{l} (v0,v1,5),(v1,v2,4),(v2,v3,9),(v3,v4,7),(v4,v0,1), \\ (v0,v5,2),(v5,v4,8),(v3,v5,3),(v5,v2,1) \end{array} \right\}$$

Difference between Trees and Graphs		
	Trees	**Graphs**
Path	Tree is special form of graph i.e. **minimally connected graph** and having only one path between any two vertices.	In graph there can be more than one path i.e. graph can have uni-directional or bi-directional paths (edges) between nodes
Loops	Tree is a special case of graph having no **loops**, no **circuits** and no self-loops.	Graph can have loops, circuits as well as can have **self-loops**.
Root Node	In tree there is exactly one root node and every **child** have only one **parent**.	In graph there is no such concept of **root** node.
Parent Child relationship	In trees, there is parent child relationship so flow can be there with direction top to bottom or vice versa.	In Graph there is no such parent child relationship.
Complexity	Trees are less complex then graphs as having no cycles, no self-loops and still connected.	Graphs are more complex in compare to trees as it can have cycles, loops etc
Types of Traversal	Tree traversal is a kind of special case of traversal of graph. Tree is traversed in **Pre-Order**, **In-Order** and **Post-Order**(all three in DFS or in BFS algorithm)	Graph is traversed by **DFS:** Depth First Search **BFS :** Breadth First Search **algorithm**
Connection Rules	In trees, there are many rules / restrictions for making connections between nodes through edges.	In graphs no such rules/ restrictions are there for connecting the nodes through edges.
DAG	Trees come in the category of **DAG : Directed Acyclic Graphs** is a kind of directed graph that have no cycles.	Graph can be **Cyclic or Acyclic**.
Different Types	Different types of trees are : **Binary Tree , Binary Search Tree, AVL tree, Heaps**.	There are mainly two types of Graphs :**Directed and Undirected graphs**.
Applications	Tree applications: sorting and searching like Tree Traversal & Binary Search.	Graph applications : Coloring of maps, in OR (**PERT & CPM**), algorithms, Graph coloring, job scheduling, etc.
No. of edges	Tree always has **n-1** edges.	In Graph, no. of edges depends on the graph.
Model	Tree is a **hierarchical model**.	Graph is a **network model**.

Figure





Other types of graphs:

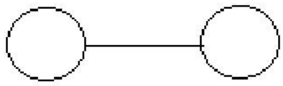
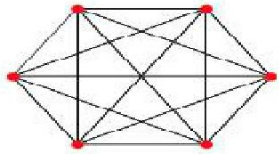
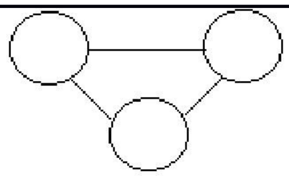
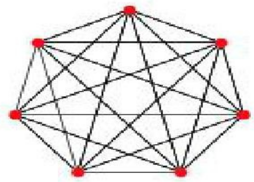
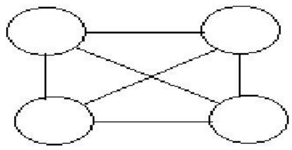
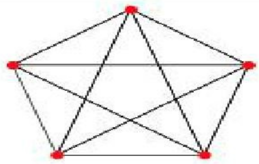
<u>Complete Graph:</u>

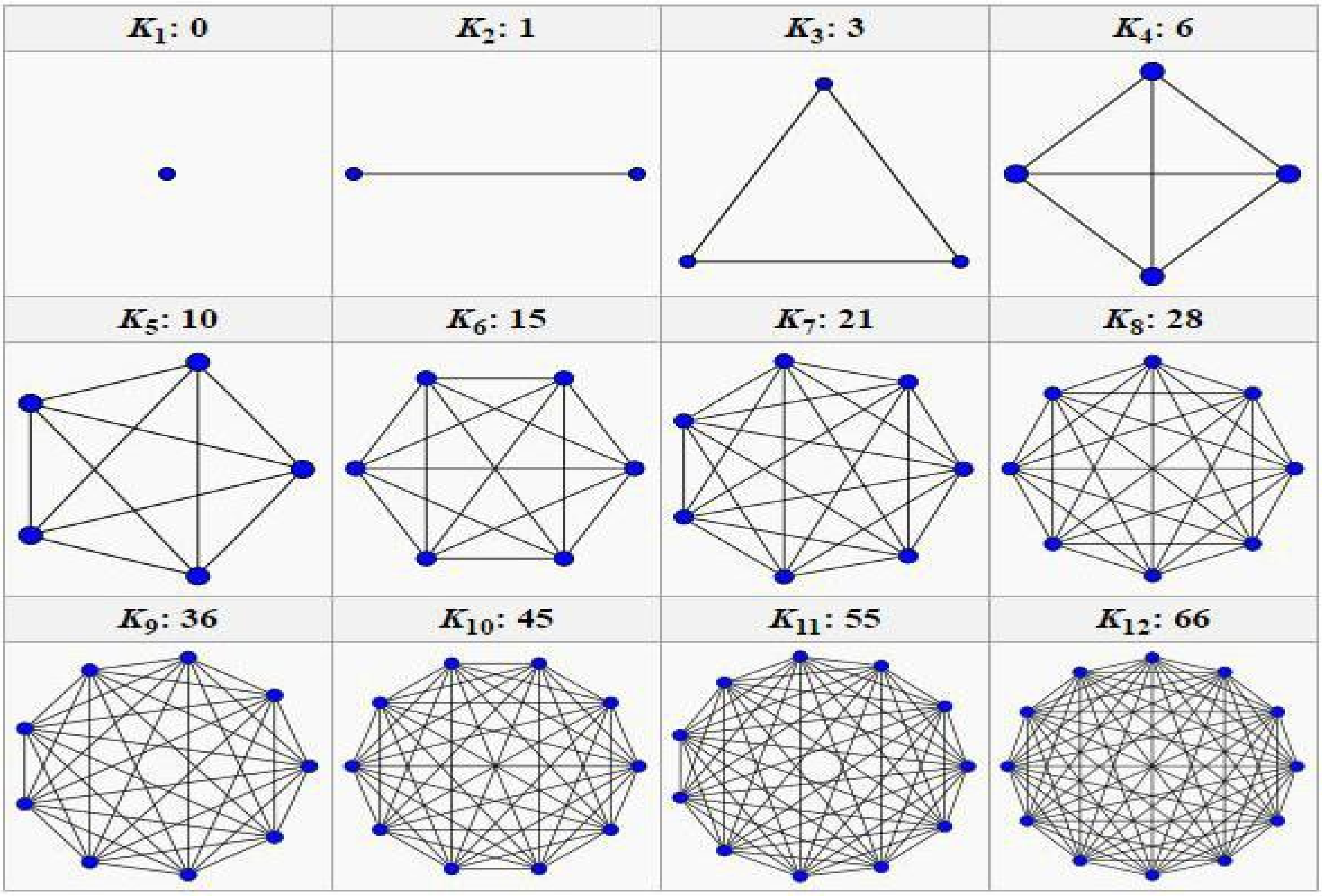
A **complete graph** is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge.

OR

If an undirected graph of n vertices consists of **n(n-1)/2** number of edges then the graph is called complete graph.

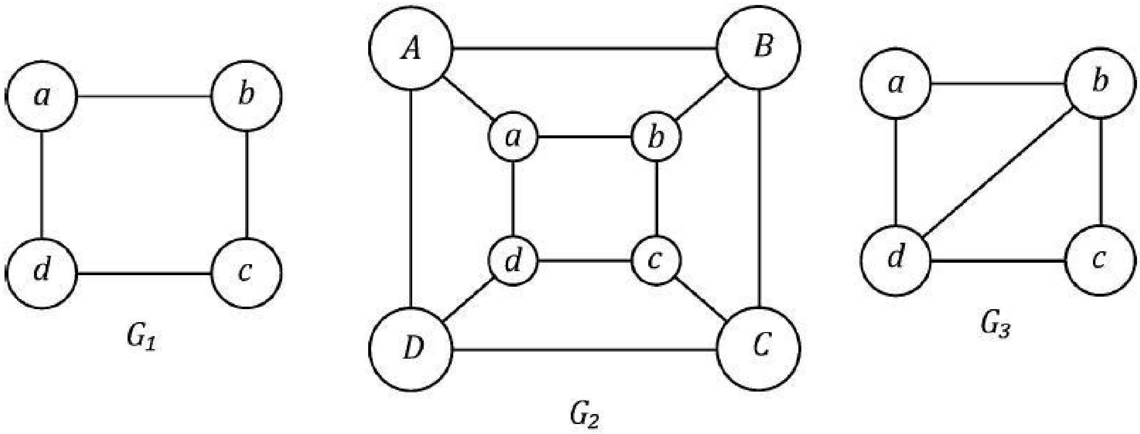
Example:

vertices	Edges	Complete graph	vertices	Edges	Complete graph
n=2	1		n=6	15	
n=3	3		n=7	21	
n=4	6		n=5	10	



<u>Sub graph:</u>

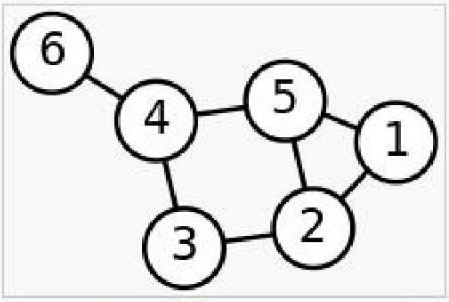
A sub-graph G' of graph G is a graph, such that the set of vertices and set of edges of G' are proper subset of the set of vertices and set of edges of graph G respectively.



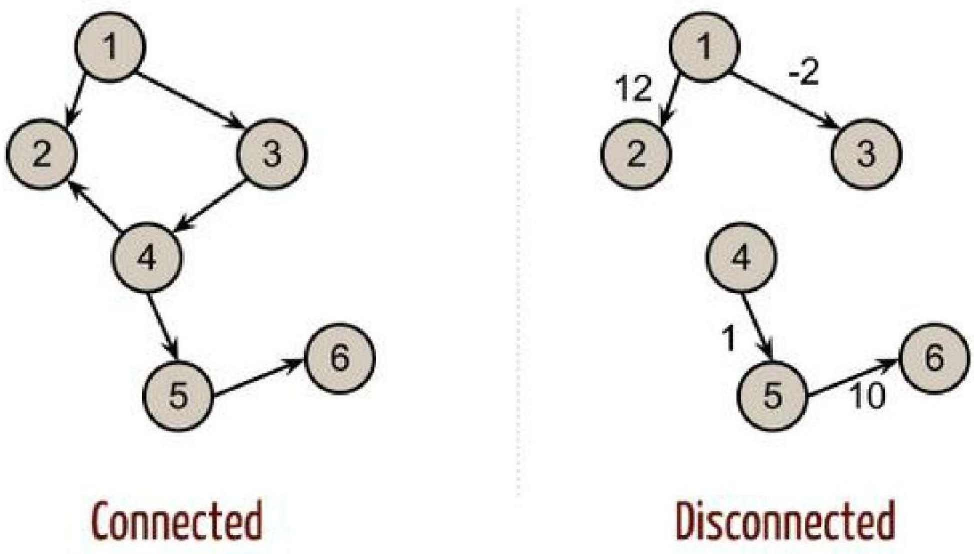
$G_1 \subseteq G_2,\ G_1 \subseteq G_3$ *but* $G_3 \nsubseteq G_2$.

Connected Graph:

A graph which is connected in the sense of a topological space (study of shapes), i.e., there is a path from any point to any other point in the graph. A graph that is not connected is said to be disconnected.

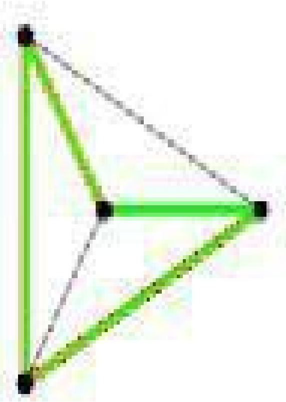


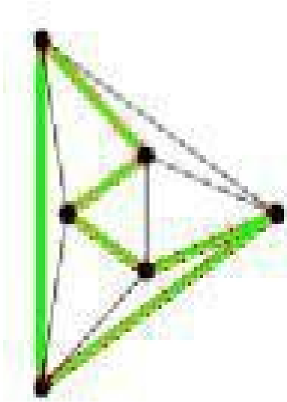
A connected graph

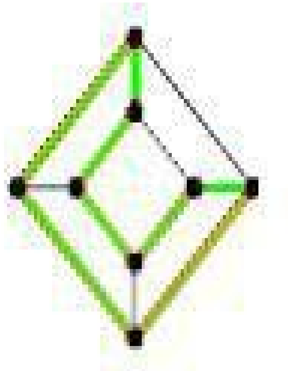


Path:

A path in a graph is a finite or infinite sequence of edges which connect a sequence of vertices. Means a path form one vertices to another vertices in a graph is represented by collection of all vertices (including source and destination) between those two vertices.

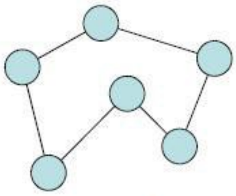




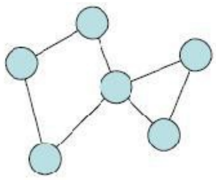


Cycle: A path that begins and ends at the same vertex.

Simple Cycle: a cycle that does not pass through other vertices more than once



Simple Cycle

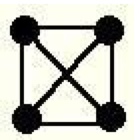


Cycle

Degree:

The degree of a graph vertex v of a graph G is the number of graph edges which touch v. The vertex degree is also called the local degree or valency. Or

The degree (or valence) of a vertex is the number of edge ends at that vertex.



For example, in this graph all of the vertices have degree three.

In a digraph (directed graph) the degree is usually divided into the **in-degree** and the **out-degree**

- In-degree: The in-degree of a vertex v is the number of edges with v as their terminal vertex.
- Out-degree: The out-degree of a vertex v is the number of edges with v as their initial vertex.

9/10/14

UNIT-4
GRAPHS

Graph :—

Graph is a non-linear data structure. That consists of following two components.

* A finite set of vertices also called nodes.
* A finite set of ordered pair of the form (u,v) called as edge.

Graph G = (V, E)

V → vertices ; E → Edges.

Edge is a pair (v,w) where (v,w) ∈ V.

Applications of graphs:—

* Colouring of maps
* Representing networks.
  * Path in a city.
  * Path in a telephone network.
  * Path in a circuit network.
* It will also used in social network's including facebook.

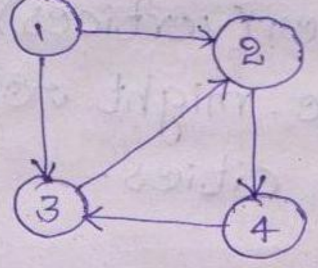
Types of graphs:—

1. Directed Graph.
2. Undirected Graph.

Directed graph :—

- In representation of graph there is a direction b/w the vertices. This can be shown by arrow mark on edge.

Graph $G = (V, W)$
where $(V, W) \in V \rightarrow$ vertex W is adjacent to vertex V.



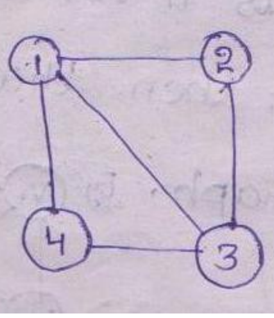
- In directed graph the vertices are ordered.

Undirected graph :—

- In graph vertices are not ordered, then it is called undirected graph, means there is no directions on any line (edge) in graph G.

Graph $G = (V, W)$
where $(V, W) \in V$

- In undirected graph with edge (V, W), W is adjacent to V & V is adjacent to W.



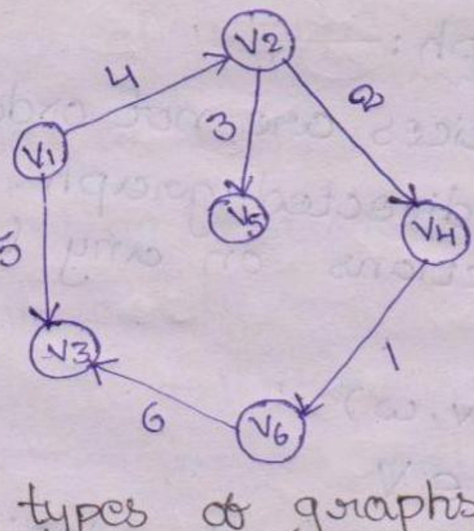
Note :—

* In graph there is a another component called "Weight".

Weight :—

Edge may be weighted to show that there is a cost to go from one vertex to another.

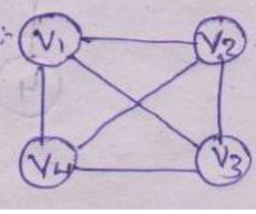
Eg:- In graph of roads (edges) that connect one city to another (vertices). The weight on the edge might represent the distance b/w two cities.



Another types of graphs:—

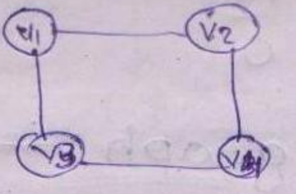
· Complete graph:—

Ifn a graph there is n vertex, if $\frac{n(n-1)}{2}$ edges are there, then that graph is called complete graph. Eg:-



Connected graph:

A graph which is connected in the sense of a topological space. i.e., there is a path from any pt (vertex) to another pt in the graph.



v_1 to v_4

$v_1 \rightarrow v_3$ | $v_1 \rightarrow v_2$
$v_3 \rightarrow v_4$ | $v_2 \rightarrow v_4$.

Graph storage representation:

There are two methods for graph storage representation (i) Adjacency matrix.
(ii) Adjacency list.

Adjacency matrix:

In which a two dimensional matrix is used for representing graph.

In this matrix.,

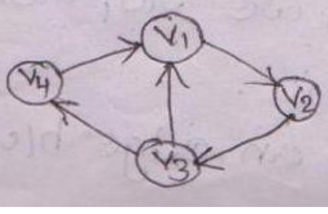
rows $\longrightarrow$ source vertex.
columns $\longrightarrow$ Destination vertex.

If graph G contains 'n' vertices then the matrix M of size $n \times n$

Directed graph:

In directed graph there is an edge b/w V_i & V_j then $M[i][j] = 1$. If there is no edge then $M[i][j] = 0$.

Eg:



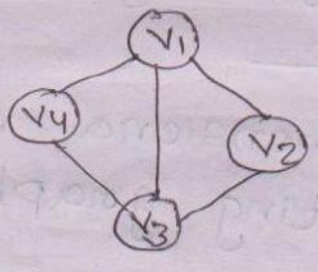
adjacency matrix:

	v_1	v_2	v_3	v_4
v_1	0	1	0	0
v_2	0	0	1	0
v_3	1	0	0	1
v_4	1	0	0	0

For undirected graph :—

For undirected graph G, if there is an edge b/w v_i & v_j then $M[i][j] = 1$ & $M[j][i] = 1$.

Eg.:-



	v_1	v_2	v_3	v_4
v_1	0	1	1	1
v_2	1	0	1	0
v_3	1	1	0	1
v_4	1	0	1	0

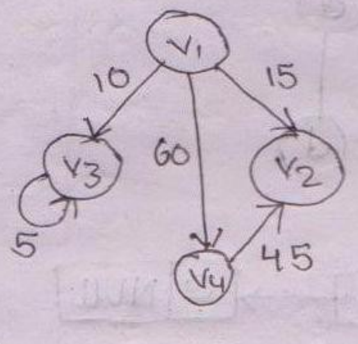
Weighted graph :—

In weighted graph, weight (or) cost (or) distance are mentioned on edge. In representation, we represent these values in matrix.

- If there is an edge b/w v_i & v_j then

$M[i][j] = cost[edge].$

Eg:-



	V_1	V_2	V_3	V_4
V_1	0	15	10	60
V_2	0	0	0	0
V_3	0	0	5	0
V_4	0	45	0	0

Adjacency List :-

An adjacency list of a graph is a collection of unordered list - one for each vertex in the graph.

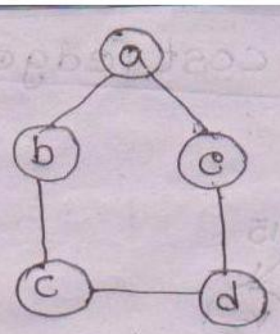
- It is done by using linked list.
- Each List describes the set of neighbours of its vertex.

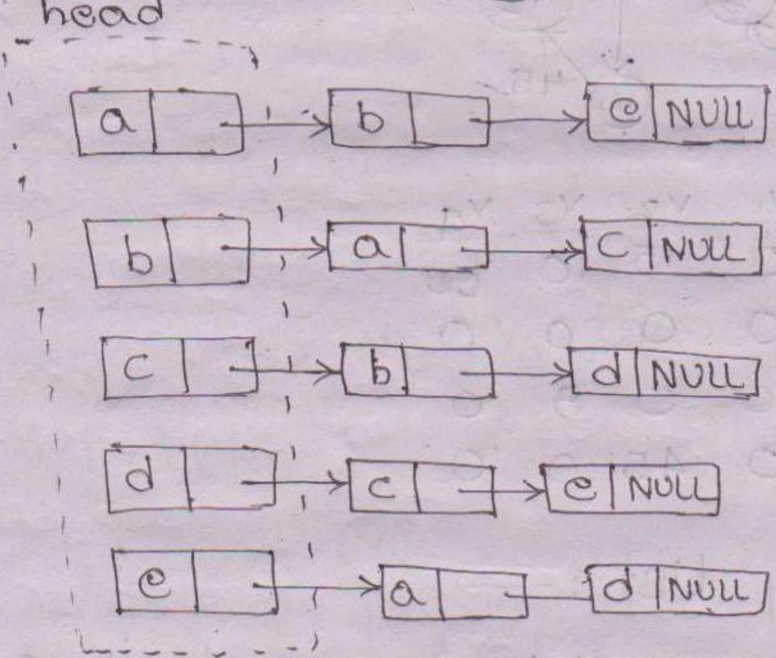
For undirected graph :-

The adjacency list representation will be done in three ways.

- Graph in a collection of two sets, vertices and edge. For these two sets we maintain two structures

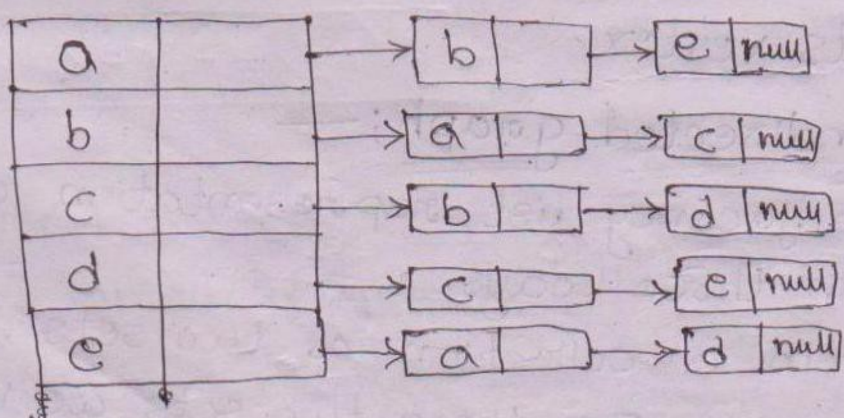
Ex:-





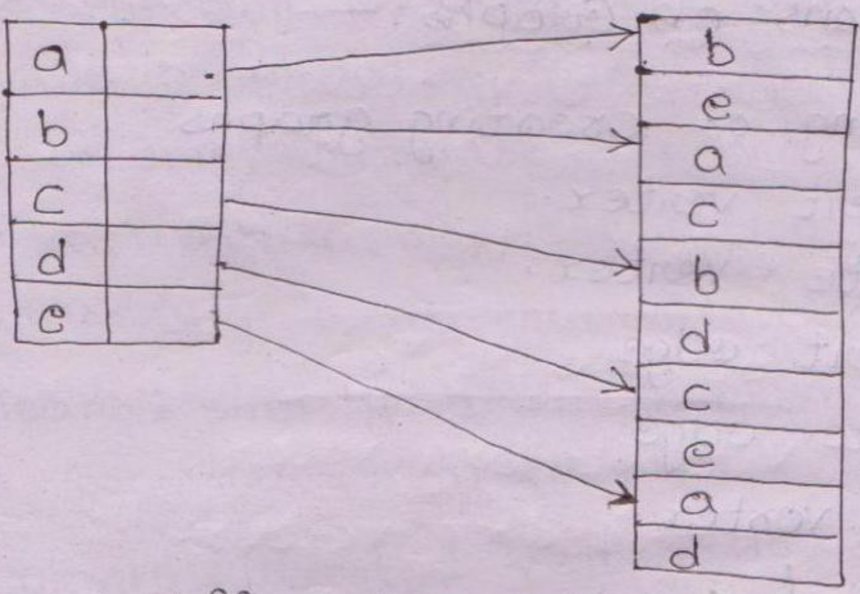
Method-2 :—

In this we use mixed data structure i.e., we use array ~~for~~ head nodes instead of head list as a linked list.



Method-3 :—

In this method adjacency list using cursor is used.



Compare different graph storage representation:-

	Adjacency list	Adjacency matrix
Storage	$O(\|V\|+\|E\|)$	$O(\|V\|^2)$
Add vertex	$O(1)$	$O(\|V\|^2)$
Delete vertex	$O(\|E\|)$	$O(\|V\|^2)$
Add edge	$O(1)$	$O(1)$
Delete edge	$O(\|E\|)$	$O(1)$
Finding vertices (u,v) are adjacent	$O(\|V\|)$	$O(1)$

NOTE:-

* In adjacency list when removing edge we must find all vertices are vertices we must find all vertices and edges.
* In adjacency matrix it is slow for adding or removing vertices because matrix must be resized.

Operations on Graphs :—

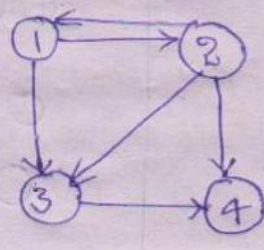
* Storing or creating graphs.
* Insert vertex.
* Delete vertex.
* Insert edge
* Delete edge
* Find vertex.

i) Storing & creating graphs :—

There are two methods for storing.

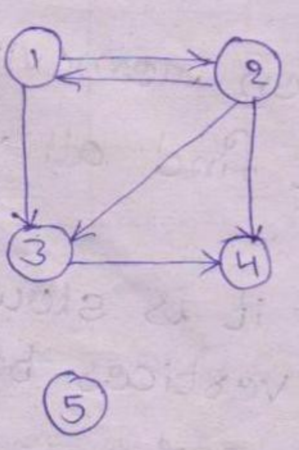
1. Adjacency list
2. Adjacency matrix.

ii) Insert vertex :—



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

$\boxed{\text{addVertex}(V_n);}$

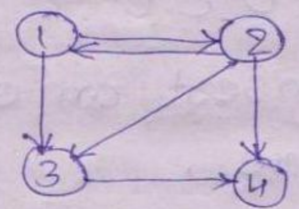


	1	2	3	4	5
1	0	1	1	0	0
2	1	0	1	1	0
3	0	0	0	1	0
4	0	0	0	0	0
5	0	0	0	0	0

This function inserted one more node into the graph, after inserting the graph size becomes increase by one. So, the size of matrix (representation of graph) increase by 1 at column level & row level.

- Means., simply after inserting vertex $n \times n$ becomes $(n+1) \times (n*1)$.
- The newly inserted vertex(s) do not have indegree & out degree.

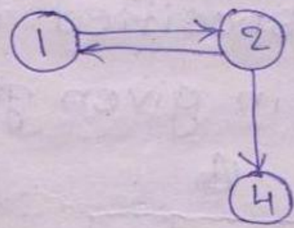
(iii) Delete Vertex :—



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

$\boxed{\text{deleteVertex}(V_G)\ ;}$

Delete vertex 3 :—



	1	2	4
1	0	1	0
2	1	0	1
~~3~~			
4	0	0	0

This function used to delete specified node/vertex which are present in the stored graph G.

If node is present then matrix (representation of graph) that vertex number column & row.

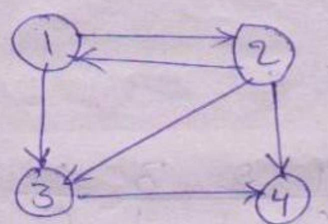
- If the given node is not present in given graph, then return node not available.

(iv) Insert edge:-

$\boxed{\text{addEdge}(V_s, V_e);}$

where., $V_s \rightarrow$ Starting vertex

$V_e \rightarrow$ Ending vertex.



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

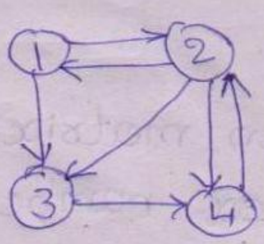
This function used to insert an edge b/w two vertices. Those are.,

$V_s \rightarrow$ starting vertex of edge.

$V_e \rightarrow$ Ending vertex of edge.

If two vertices that specified in addEdge are available in given graph G. then we put the value on

$\boxed{G[V_s][V_e] = \text{cost of edge}}$



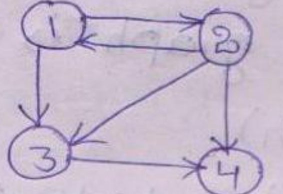
	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	0	0	0	1
4	0	1	0	0

$G[4][2] = 1$

ii) Delete edge:-

$\boxed{\text{deleteEdge (Vs, Ve);}}$

case:



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

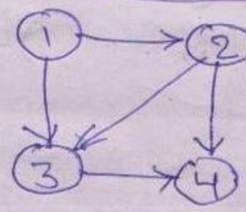
This function used to delete an edge b/w two vertices. Those are.,

$V_S \rightarrow$ starting vertex

$V_e \rightarrow$ Ending vertex

If two vertices that specified in deleteEdge are available in given graph G then we put the value.

$\boxed{G[V_S][V_e] = \text{cost of edge}}$



$G[2][1] = 0;$

	1	2	3	4
1	0	1	1	0
2	$\boxed{0}$	0	1	1
3	0	0	0	1
4	0	0	0	0

Graph Traversing:-

Graph traversing is the process of visiting every node (vertex) exactly once.

- Graph traversing is for

a) Finding a particular node available or not in a given graph.

b) Finding all reachable nodes.
c) Finding best reachable nodes.
d) Finding best path through a graph.
e) Determining whether a graph is DAG. (directed acyclic graph)
DAG:- A directed graph which doesn't contains a cycle's.

- The main goal of graph traverse is to find all nodes reachable from a given node

* In undirected graph we follow all edges.
* In directed graph we follow outedge

- Graph traversing has two strategies/algorithms

1) DFS (Depth First search algorithm).
2) BFS (Breadth " " ").

DFS :—

Introduction :—

Depth First search / Traversing

DFS / Traversing was investigated in 19th century. by a french mathematician, charles Tremaux for solving maze.

Use of DFS

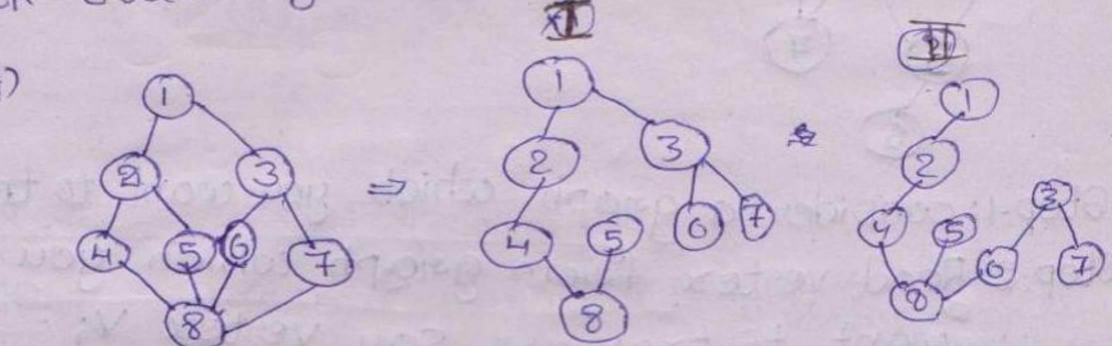
Definition:—

DFS is an algorithm for traversing / Searching / tree data structure.

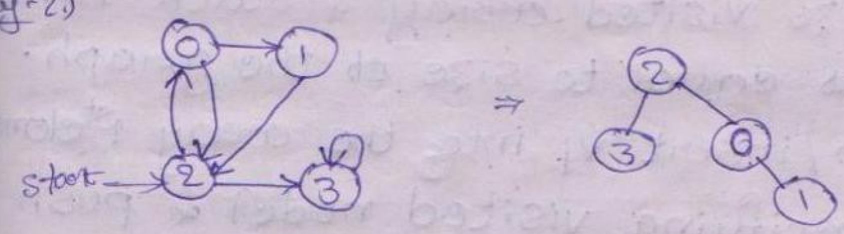
Working of DFS :—

In these start at the root (selecting some arbitary node as the root in the graph) & explore's (moves / visit). As far as possible along each branch before back-tracking.

Eg:1)



Eg-2)



· Spanning tree: —

Sub-graph N of a graph G is called spanning tree if it satisfy the following two properties.

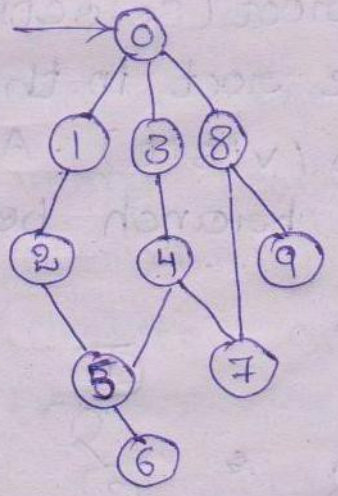
i) It includes all vertices/nodes of a graph G.

ii) There is no cycle's (It must be a tree).

We start from one vertex & traverse the path as deeply as we go. When there is no vertex for that. then we traverse back & search for unvisited vertex.

Implementation of DFS: —

DFS is implemented by using ~~gra~~ stack data structure.



Step-1: consider a graph which you want to traverse.

Step-2: Read vertex from graph which you want to traverse, say vertex V_i.

Step-3: Initialize visited array & stack. Array size is equal to size of the graph.

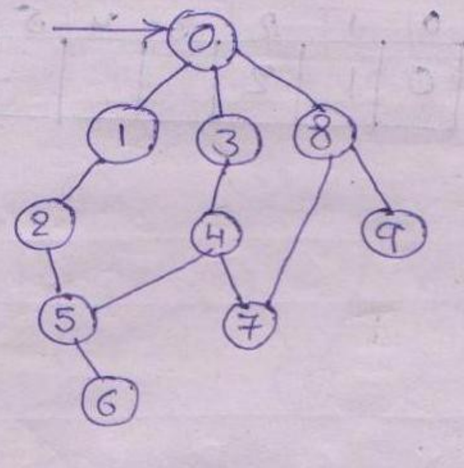
Step-4: Assign/insert V_1 into the array 1st element (for specifying visited node) & push all adjacent nodes/vertices of V_i into the stack.

Step-5: Pop the top value of the stack & insert it to the visited array. & push all adjacency nodes of poped node/element.

Step-6: If pop value (top of stack) is already present in array then don't insert into visited array, just we discard (or) neglect the top value.

Step-7: Do step-5 until stack is empty & array is full.

Step-1:-



Step-2:- Here $v_i = 0$.

Step:3: visited array (Length of visited array is no. of nodes in the graph).

0	1	2	3	4	5	6	7	8	9

Take stack. Initially stack is empty

Step-4:-

1
3
8

0	1	2	3	4	5	6	7	8	9
0									

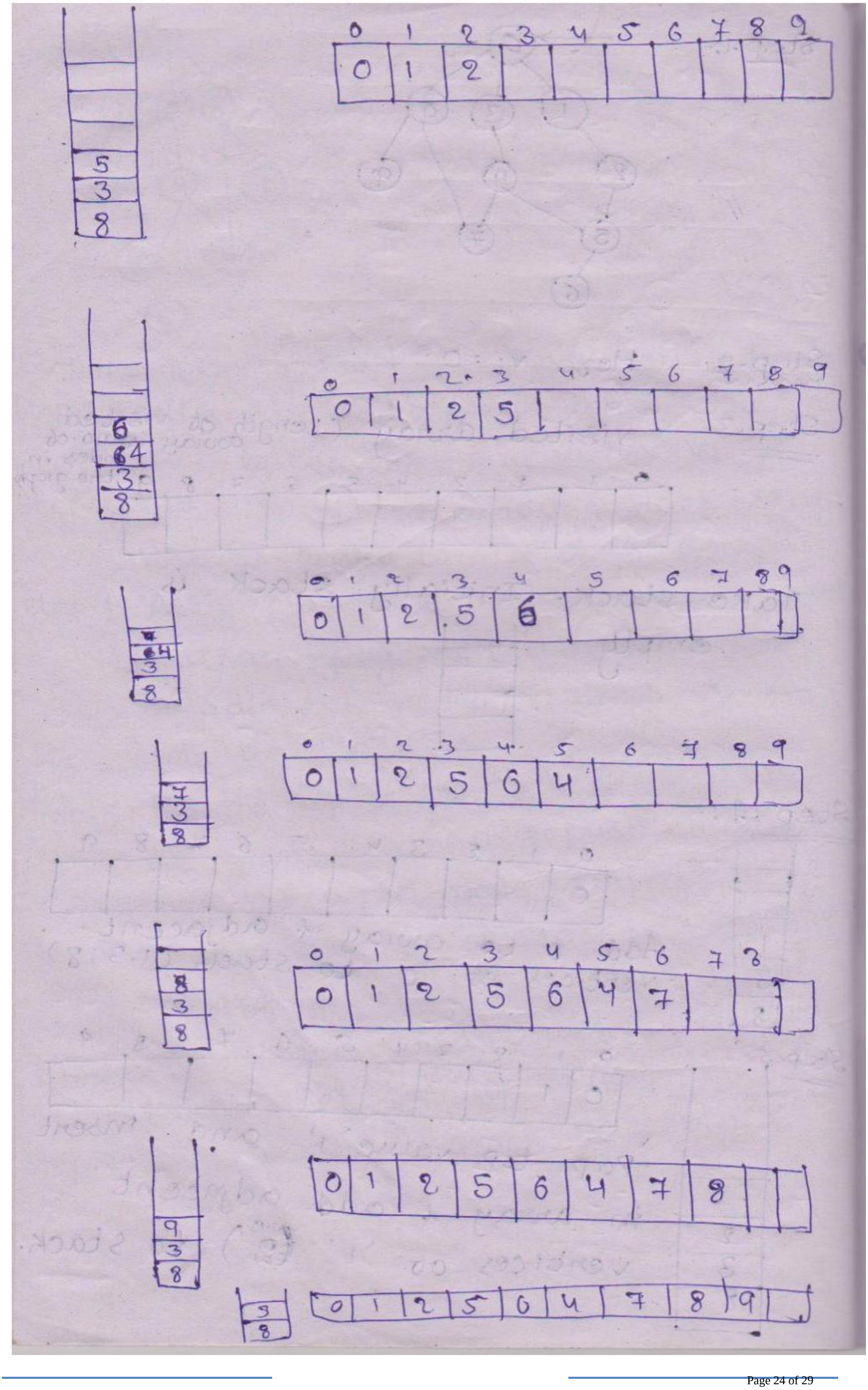
Add 0 to array & adjacent vertices of 0 to stack (1,3,8)

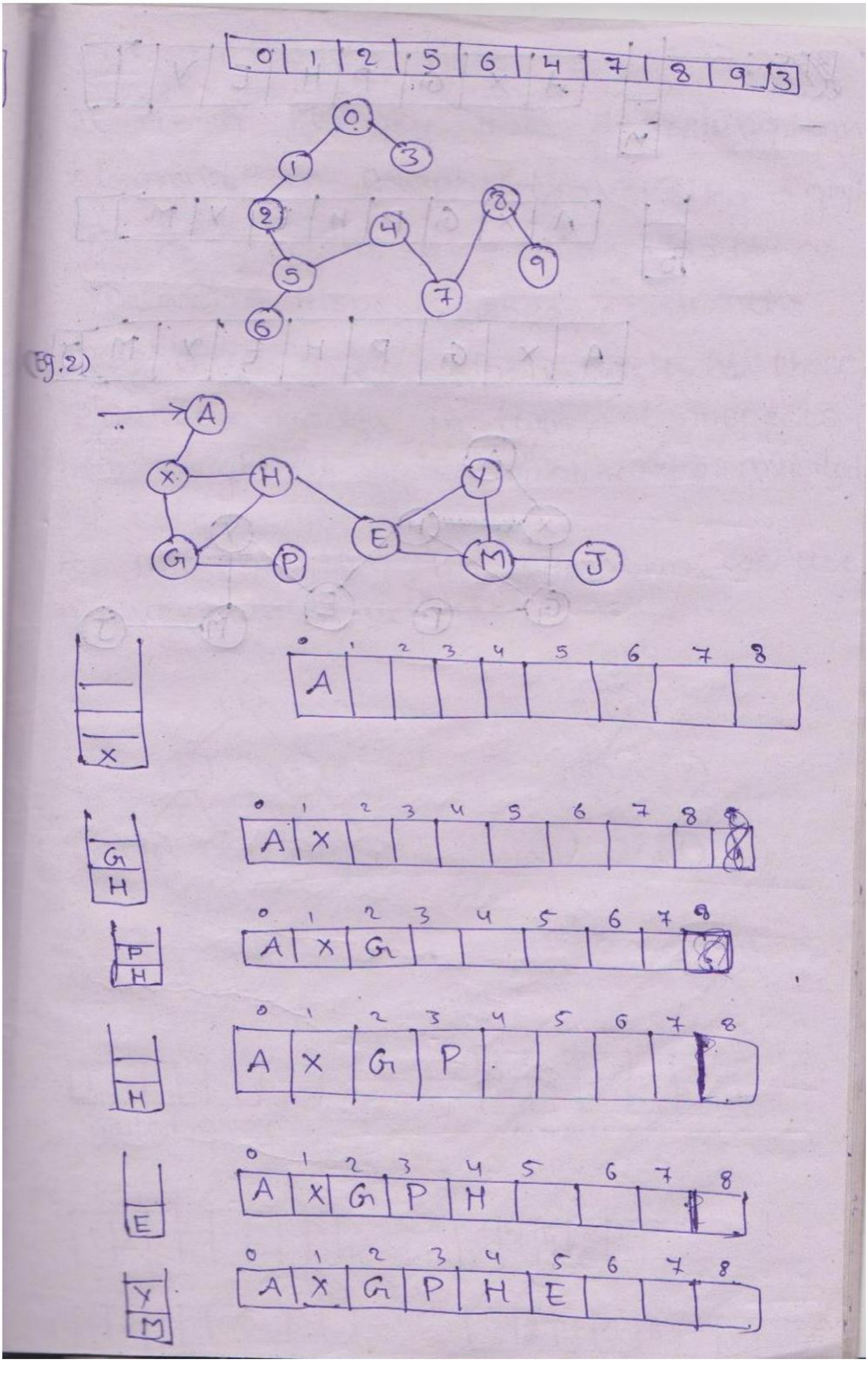
Step-5:

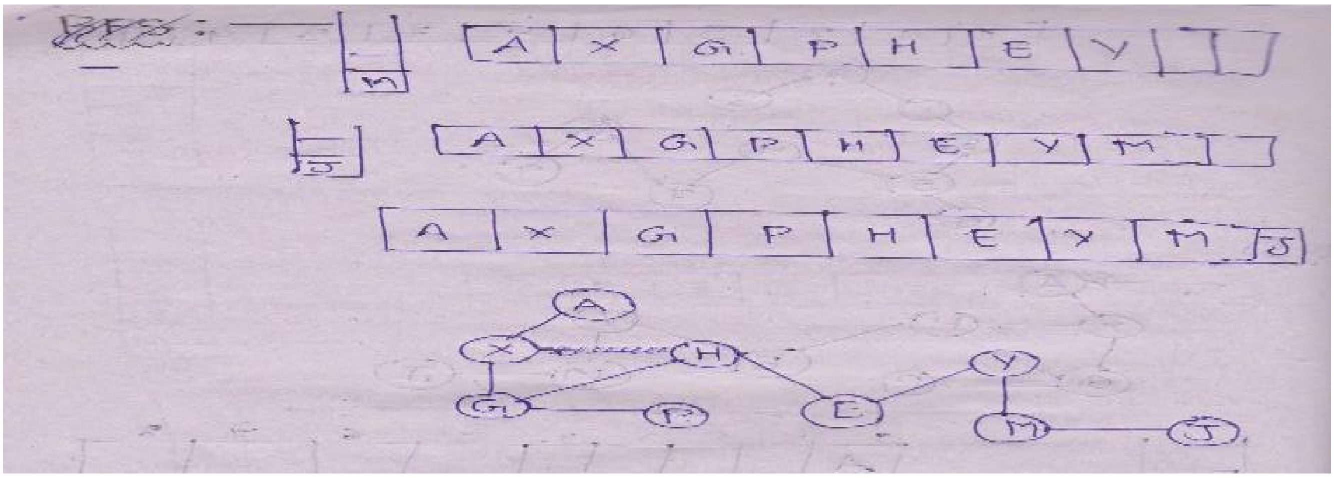
0	1	2	3	4	5	6	7	8	9
0	1								

Pop top value '1' and insert in array & add adjacent vertices of '1' (i.e. 2) to stack.

2
3
8


0 1 2 3 4 5 6 7 8 9
0 1 2
5
3
8
0 1 2 3 4 5 6 7 8 9
0 1 2 5
6
64
3
8
0 1 2 3 4 5 6 7 8 9
0 1 2 5 6
0 1 2 3 4 5 6 7 8 9
0 1 2 5 6 4
7
3
8
0 1 2 3 4 5 6 7 8
0 1 2 5 6 4 7
8
3
8
0 1 2 5 6 4 7 8
9
3
8
0 1 2 5 6 4 7 8 9
9
8


0 1 2 5 6 4 7 8 9 3
Eg.2)

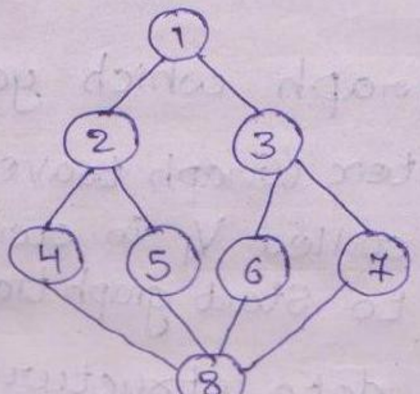

M
A X G P H E Y
J
A X G P H E Y M
A X G P H E Y M J
A
X
H
Y
G
P
E
M
J

BFS :— (Breadth first search)

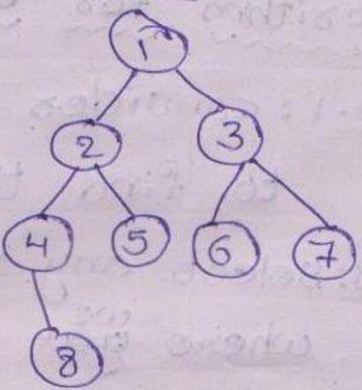
It is a stragey for searching an element in graph (traversing a graph)

- The BFS begins at certain node or vertex & inspects (observe) all the neighbour nodes. Then each of those neighbour nodes in turn, it inspects their neighbour nodes which were invisited and so on....
- For implementing BFS operation we use queue data structure.

Ex:-



BFS



1							

visited array

2	3			

Queue.

1	2						

	3	4	5	

1	2	3					

	3	4	5	6	7	

1	2	3	4				

				5	6	7	8

1	2	3	4	5			

				8	6	7	8

1	2	3	4	5	6		

						7	8

1	2	3	4	5	6	7	

							8

1	2	3	4	5	6	7	8

Algorithm for BFS:—

Step-1: consider the graph which you want to find the vertex (Graph traversing).

Step-2: select any vertex called V_i in our graph where we ~~you~~ want to start graph traversing

Step-3: consider any two data structures,
- visited array (size of the graph.
- queue d.s (FIFO)

Step-4: Assign starting vertex V_i into the visited array & the adjacent vertices or adjacent nodes of V_i are insert into the queue.

Step-5:- Now pop element of queue by using FIFO principle, that popped element insert into the array second element. The popped element adjacent vertices inserted into queue.

Step-6: Do step-5 until there is no vertex left in graph & there is no loop.