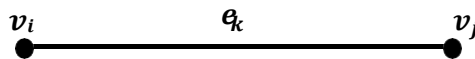


### ❖ INTRODUCTION

- ✓ Graph theory plays an important role in several areas of computer science such as switching theory and logical design, artificial intelligence, formal languages, computer graphics, operating systems, compiler writing, information organization and retrieval.

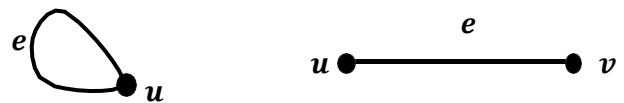
### ❖ GRAPH

- ✓ A graph  $G = \langle V, E, \emptyset \rangle$  consists of a nonempty set  $V$  called the set of nodes (points, vertices) of the graph,  $E$  is said to be the set of edges of the graph and  $\emptyset$  is a mapping from the set of edges  $E$  to a set of ordered or unordered pairs of elements of  $V$ .
- ✓  $V = V(G) = \{v_1, v_2, v_3, \dots\}$  = The set of nodes(vertices/points/dots/junctions).
- ✓  $E = E(G) = \{e_1, e_2, e_3, \dots\}$  = The set of edges(branch/line/arc).
- ✓ The elements of  $V$  are called **nodes/vertices** of a graph  $G$  and the elements of  $E$  are called **edges** of a graph  $G$ .



### NOTE:

- ✓ Throughout, we shall assume that the sets  $V$  and  $E$  of a graph  $G$  are finite.
- ✓ Any edge  $e$  can be made by one OR two nodes  $\{u, u\}$  OR  $\{u, v\}$  respectively.

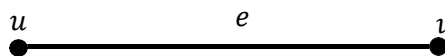


### ❖ ADJACENT NODES

- ✓ If two nodes  $u$  and  $v$  are joined by an edge  $e$  then  $u$  and  $v$  are said to be adjacent nodes.

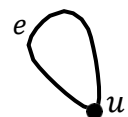
### ❖ INCIDENT EDGE

- ✓ An edge  $e \in E$  (directed/undirected) which joins the nodes  $u$  and  $v$  is said to be incident to the nodes  $u$  and  $v$ .



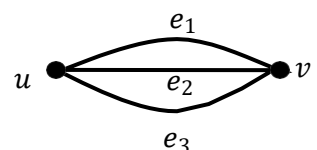
### ❖ LOOP(SLING)

- ✓ An edge  $e$  of a graph  $G$  that joins a node  $u$  to itself is called a loop. A loop is an edge  $e = \{u, u\}$ .



### ❖ PARALLEL EDGES

- ✓ If two nodes of a graph are joined by more than one edge then these edges are called parallel edges/multiple edges.

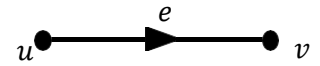


### ❖ DIRECTED EDGES

- ✓ In a graph  $G$  an edge 'e' which is associated with an ordered pair of nodes 'u' to 'v' is called directed edge of graph  $G$ .

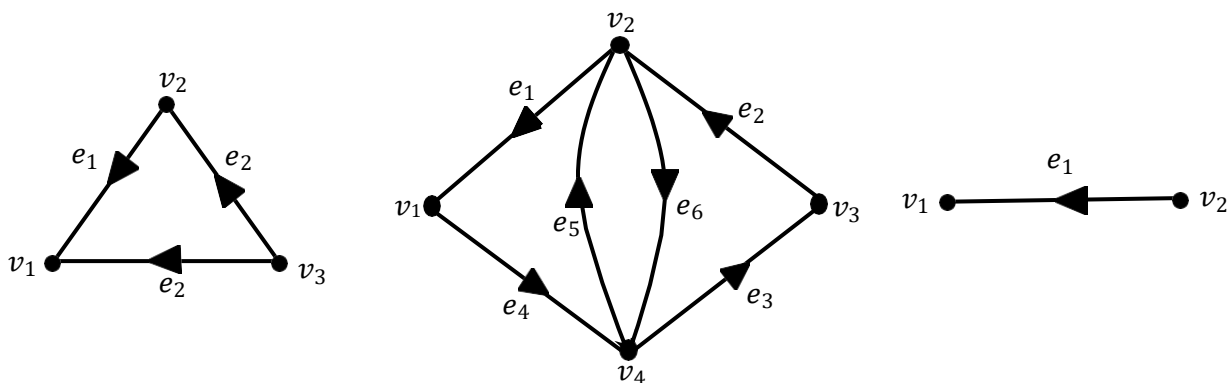
### ❖ INITIATING NODE AND TERMINATING NODE

- ✓ Let  $G = \langle V, E \rangle$  be a graph and let  $e \in E$  be a directed edge associated with the order pair  $\langle u, v \rangle$  of the nodes 'u' and 'v' then the edge 'e' is said to be initiating $\langle$ originating $\rangle$  in the node 'u' and terminating $\langle$ ending $\rangle$  in the node 'v'. The nodes 'u' and 'v' are called initial node and terminal node of the edge 'e' respectively.



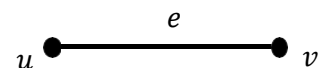
### ❖ DIRECTED GRAPH/DIGRAPH

- ✓ A graph in which every edge is directed is called a directed graph(digraph).



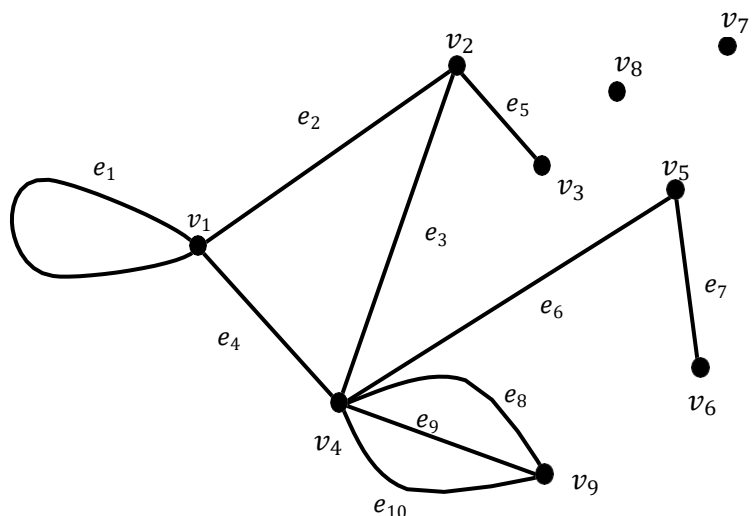
### ❖ UNDIRECTED EDGE

- ✓ In a graph  $G$  an edge 'e' which is associated with an unordered pair  $\{u, v\}$  of nodes 'u' and 'v' is called undirected edge of graph  $G$ .



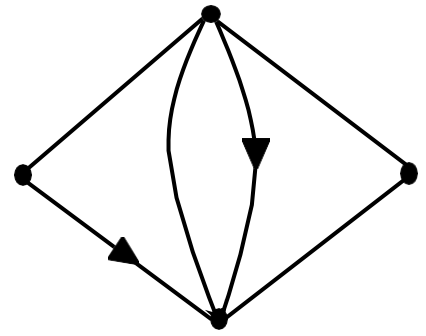
### ❖ UNDIRECTED GRAPH

- ✓ A graph in which every edge is undirected is called an undirected graph.



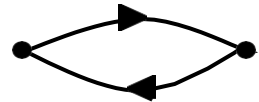
### ❖ MIXED GRAPH

- ✓ If some edges of a graph  $G$  are directed and some are undirected then  $G$  is said to be a mixed graph.



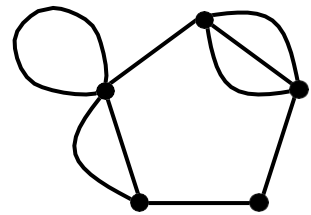
### ❖ DISTINCT EDGES

- ✓ The two possible edges between a pair of nodes which are opposite in direction which are known as distinct edges.



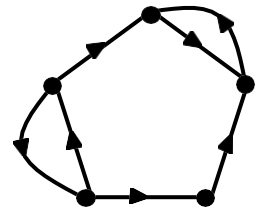
### ❖ MULTI GRAPH

- ✓ Any graph which contains some parallel edges is called a multigraph.



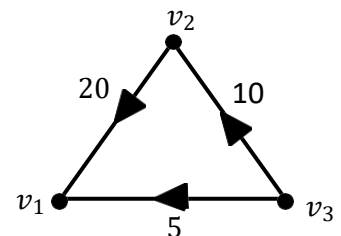
### ❖ SIMPLE GRAPH

- ✓ A graph which has neither loop nor parallel edges is called a simple graph.



### ❖ WEIGHTED GRAPH

- ✓ A graph in which weights are assigned to every edge is called a weighted graph.

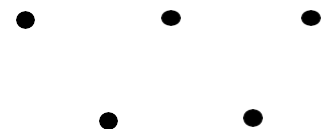


### ❖ ISOLATED NODE

- ✓ In a graph a node which is not adjacent to any other node is called an isolated node.

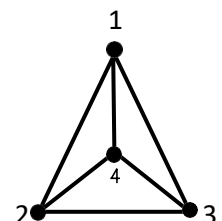
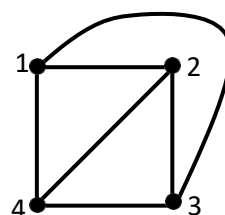
### ❖ NULL GRAPH

- ✓ A graph containing only isolated nodes is called a null graph.
- ✓ The set of edges in the null graph is empty.



### NOTE:

- ✓ It can happen that two diagrams which look entirely different but both may represent the same graph.



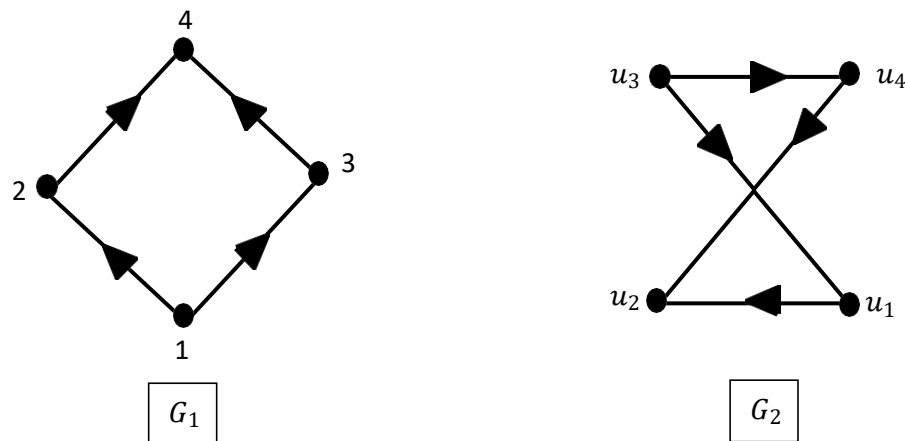
### ❖ METHOD-1: BASIC DEFINITIONS AND RELATED EXAMPLES

H	1	Define with example: graph, nodes and edges.	
T	2	Define with example: adjacent nodes, initiating node, terminating node and isolated node.	
T	3	Define with example: incident edges, loop, parallel edges, directed edges, undirected edges and distinct edges.	
H	4	Define with example: directed graph, undirected graph, mixed graph, multi graph, simple graph, weighted graph and null graph.	
C	5	Draw the undirected graph $G = \{V, E\}$ where, $V = \{a, b, c, d, e\}$ and $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ and its incidence relation given as: $e_1 = \{a, b\}$ , $e_2 = \{a, b\}$ , $e_3 = \{b, c\}$ , $e_4 = \{c, d\}$ , $e_5 = \{b, b\}$ , $e_6 = \{a, d\}$ & $e_7 = \{e, d\}$ . Discuss the terms defines in example 1 to 4 for G.	
C	6	Draw the directed graph $G = \langle V, E \rangle$ where, $V = \{a, b, c, d, e, f, g\}$ and $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}$ and its incidence relation given as: $e_1 = \langle b, a \rangle$ , $e_2 = \langle d, a \rangle$ , $e_3 = \langle b, c \rangle$ , $e_4 = \langle d, c \rangle$ , $e_5 = \langle c, f \rangle$ , $e_6 = \langle f, f \rangle$ , $e_7 = \langle e, c \rangle$ & $e_8 = \langle c, e \rangle$ . Discuss the terms defines in example 1 to 4 for G.	

### ❖ ISOMORPHIC GRAPH

- ✓ A graph  $G_1 = \{V_1, E_1\}$  is said to be isomorphic to the graph  $G_2 = \{V_2, E_2\}$  if there exists a bijection between the set of nodes  $V_1$  and  $V_2$  and a bijection between the set of edges  $E_1$  and  $E_2$  such that if  $e$  is an edge with end nodes  $u$  and  $v$  in  $G_1$  then the corresponding edge  $e'$  has its end nodes  $u'$  and  $v'$  in  $G_2$  which correspond to  $u$  and  $v$  respectively. If such pair of bijections exist then it is called a graph isomorphism and it is denoted by  $G_1 \cong G_2$ .
- ✓ According to the definition of isomorphism we note that any two nodes in one graph which are joined by an edge must have the corresponding nodes in the other graph also joined by an edge and hence a one to one correspondence exists between the edges as well.

### EXAMPLE 1:

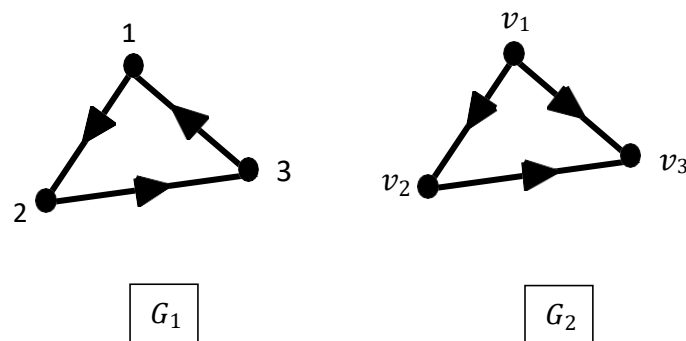


- ✓ Here  $G_1$  and  $G_2$  are isomorphic because of the existence of a mapping  $1 \rightarrow u_3, 2 \rightarrow u_2, 3 \rightarrow u_1$  &  $4 \rightarrow u_4$ .
- ✓ Under this mapping the edges  $\langle 1, 3 \rangle, \langle 1, 2 \rangle, \langle 2, 4 \rangle$  &  $\langle 3, 4 \rangle$  are mapped into  $\langle u_3, u_1 \rangle, \langle u_3, u_2 \rangle, \langle u_2, u_4 \rangle$  &  $\langle u_1, u_4 \rangle$  which are the only edges of the graph in  $G_2$ .

### NOTE:

- ✓ The two graphs which are isomorphic have the same number of nodes and edges but converse need not be true. i.e., If the two graphs which has same number of nodes and edges implies both graphs need not be isomorphic.

### EXAMPLE 2:

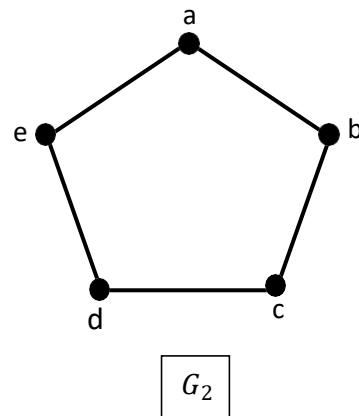
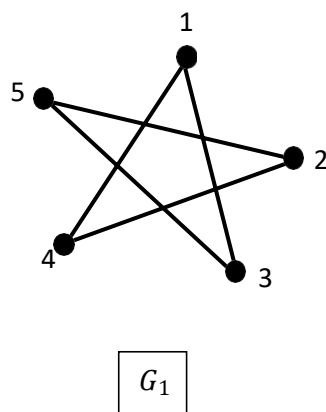


- ✓ Here  $g_1$  and  $g_2$  both has same number of nodes and edges but  $g_1$  is not isomorphic to  $g_2$ . I.e.,  $G_1 \not\cong G_2$  because the edge  $\langle 3, 1 \rangle \nrightarrow \langle v_1, v_3 \rangle$ .

### NOTE:

- ✓ The concept of isomorphism also defines in undirected graphs with the same definition given for directed graphs.

### EXAMPLE 3:



- ✓ Here  $G_1$  and  $G_2$  are isomorphic because of the existence of a mapping  $1 \rightarrow a, 2 \rightarrow d, 3 \rightarrow b, 4 \rightarrow e$  &  $5 \rightarrow c$ .
- ✓ Under this mapping, the edges  $\{1, 3\}, \{1, 5\}, \{1, 2\}, \{2, 4\}, \{2, 3\}, \{3, 5\}, \{4, 1\}$  are mapped into  $\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}, \{c, e\}$  which are the only edges of the graph in  $G_2$ .

#### ❖ ORDER OF A GRAPH

- ✓ The number of nodes in a graph  $G$  is called order of the graph  $G$ .

#### ❖ SIZE OF A GRAPH

- ✓ The number of edges in a graph  $G$  is called size of the graph  $G$ .

#### ❖ DEGREE OF A NODE

- ✓ Let  $G$  be an undirected graph then the degree of a node  $v$  in  $G$  is defined as the number of edges incident on  $v$ . It is denoted by  $d(v)$  or  $d_G(v)$  or  $\deg(v)$ .

#### NOTE:

- ✓ Self-loop will be counted twice in the degree of corresponding node.

#### ❖ ODD NODE

- ✓ A node with odd degree is called an odd node.

#### ❖ EVEN NODE

- ✓ A node with even degree is called an even node.

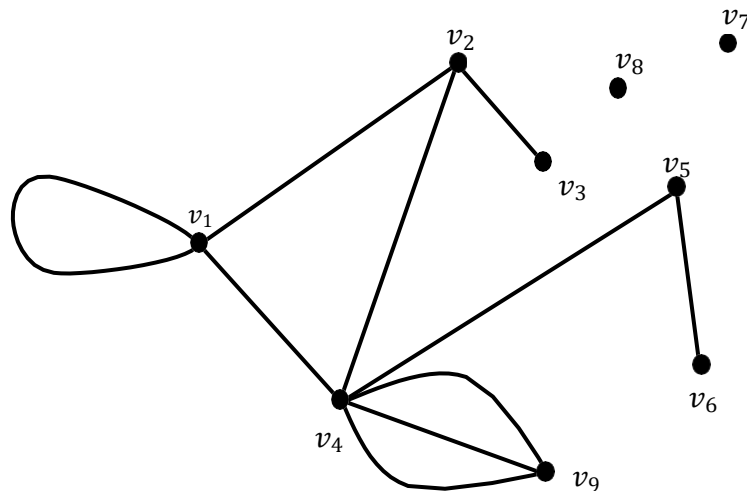
#### ❖ ISOLATED NODE

- ✓ A node with degree zero is called isolated node.

#### ❖ PENDANT NODE

- ✓ A node with degree one is called a pendent node.

### EXAMPLE:



- ✓ Here  $d(v_1) = 4, d(v_2) = 3, d(v_3) = 1, d(v_4) = 6, d(v_5) = 2, d(v_6) = 1, d(v_7) = 0, d(v_8) = 0$  &  $d(v_9) = 3$ .

- ✓ From degree of nodes we conclude that nodes  $v_1, v_4$  &  $v_5$  are even nodes, nodes  $v_2, v_3, v_6$  &  $v_9$  are odd nodes, nodes  $v_7$  &  $v_8$  are isolated nodes and nodes  $v_3$  &  $v_6$  are pendent nodes.

### ❖ HANDSHAKING THEOREM

- ✓ Any undirected graph  $G$  with  $n$  nodes  $v_1, v_2, \dots, v_n$  and  $e$  edges

$$\sum_{i=1}^n d(v_i) = 2e.$$

- ✓ In above graph

$$\begin{aligned} \sum_{i=1}^9 d(v_i) &= d(v_1) + d(v_2) + d(v_3) + d(v_4) + d(v_5) + d(v_6) + d(v_7) + d(v_8) + d(v_9) \\ &= 4 + 3 + 1 + 6 + 2 + 1 + 0 + 0 + 3 = 20 = 2 \times 10 = 2e. \end{aligned}$$

### ❖ RESULT

- ✓ In any undirected graph  $G$ , number of odd nodes must be even.
- ✓ In above graph there are 4(even) number of nodes.

### ❖ INDEGREE

- ✓ Let  $G$  be a directed graph then for any node  $v$  in  $G$ , the number of edges which have  $v$  as their terminal node is called the indegree of the node  $v$ .

OR

- ✓ In a directed graph  $G$ , the number of edges directed towards node  $v$  is called indegree of a node  $v$ . It is denoted by  $d^-(v)$ .

### ❖ OUTDEGREE

- ✓ Let  $G$  be a directed graph then for any node  $u$  in  $G$ , the number of edges which have  $u$  as their initial node is called the outdegree of the node  $u$ .

OR

- ✓ In a directed graph  $G$ , the number of edges directed outwards node  $u$  is called indegree of a node  $u$ . It is denoted by  $d^+(\tilde{u})$ .

### ❖ TOTAL DEGREE OF A NODE

- ✓ Sum of the indegree and the outdegree of a node  $v$  is called total degree of a node  $v$ . It is denoted by  $d(\tilde{v})$ . i.e.,  $d(\tilde{v}) = d^-(\tilde{v}) + d^+(\tilde{v})$ .

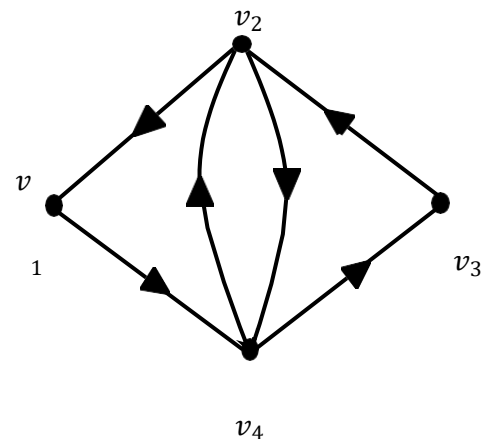
#### EXAMPLE:

$$d^+(\tilde{v}_1) = 1, d^-(\tilde{v}_1) = 1 \Rightarrow d(\tilde{v}_1) = 2$$

$$d^+(\tilde{v}_2) = 2, d^-(\tilde{v}_2) = 2 \Rightarrow d(\tilde{v}_2) =$$

$$4 \quad d^+(\tilde{v}_3) = 1, d^-(\tilde{v}_3) = 1 \Rightarrow d(\tilde{v}_3) = 2$$

$$d^+(\tilde{v}_4) = 2, d^-(\tilde{v}_4) = 2 \Rightarrow d(\tilde{v}_4) = 4$$



#### NOTE:

- ✓ The total degree of an isolated node is 0.
- ✓ The node with degree 1 is known as pendent node.

### ❖ RESULT

- ✓ In directed graph  $G$  with  $n$  nodes  $v_1, v_2, \dots, v_n$  and  $e$  edges

$$\sum_{i=1}^n d^+(\tilde{v}_i) = \sum_{i=1}^n d^-(\tilde{v}_i) = e \text{ and } \sum_{i=1}^n d(\tilde{v}_i) = 2e.$$

- ✓ In above example,

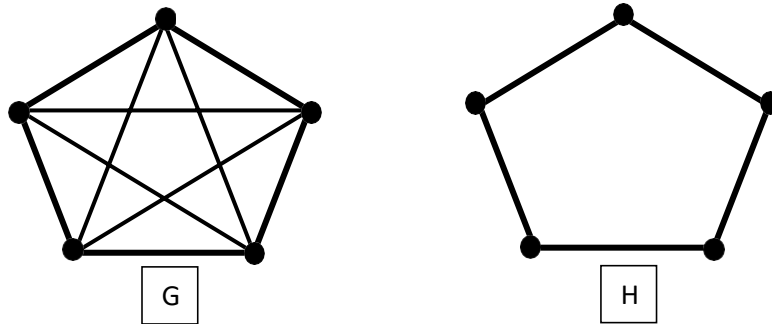
$$\sum_{i=1}^4 d^+(\tilde{v}_i) = \sum_{i=1}^4 d^-(\tilde{v}_i) = 6 \text{ and } \sum_{i=1}^4 d(\tilde{v}_i) = 12 = 2 \text{ times no. of edges.}$$

### ❖ SUBGRAPHS

- ✓ Let  $G$  and  $H$  be two graphs. Then  $H$  is said to be a subgraph of  $G$  if  $V(\tilde{H}) \subseteq V(\tilde{G})$  &  $E(\tilde{H}) \subseteq E(\tilde{G})$ . Here  $G$  is called super graph of  $H$ .



EXAMPLE:



- ✓ Here H is subgraph of G.

NOTE:

- ✓ The graph G as well as the null graph obtained from G by deleting all the edges of G are subgraphs of G. Other subgraphs of G can be obtained by deleting certain nodes and edges of G.

### ❖ NODE DELETED SUBGRAPH

- ✓ The graph obtained by deletion of a node  $v$  from a given graph  $G$  is called node deleted subgraph of  $G$ . It is denoted by  $G - \{v\}$ .

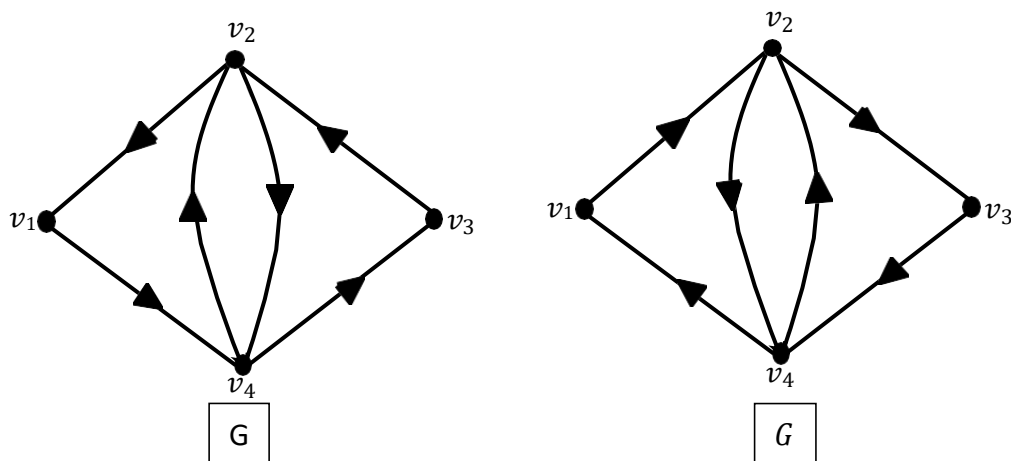
### ❖ EDGE DELETED SUBGRAPH

- ✓ The graph obtained by deletion of an edge  $e$  from a given graph  $G$  is called edge deleted subgraph of  $G$ . It is denoted by  $G - \{e\}$ .

### ❖ CONVERSE (REVERSAL/DIRECTIONAL DUAL) OF A DIGRAPH

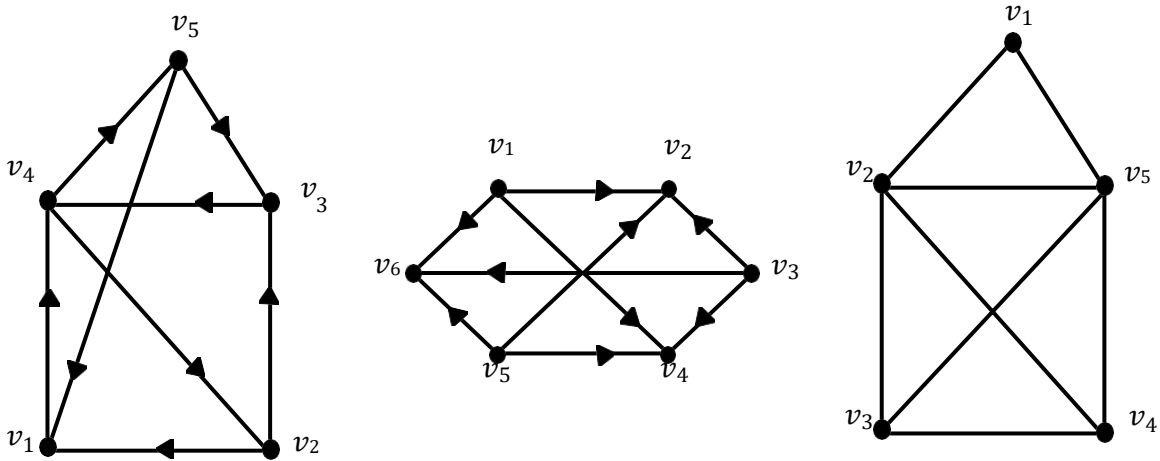
- ✓ The converse of a digraph  $G = \langle V, E \rangle$  to be a digraph  $G^* = \langle V, E^* \rangle$  in which the relation  $E^*$  is the converse of the relation  $E$ . The diagram  $G^*$  is obtained from  $G$  by simply reversing the directions of the edges in  $G$ . The converse  $G^*$  is also called the reversal or directional dual of a digraph  $G$ .

EXAMPLE:



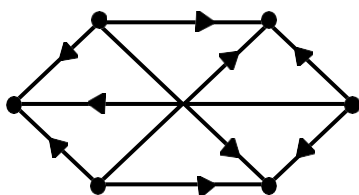
- ✓ In the above diagram  $G$  &  $G^*$  are converse of each other.

## ❖ METHOD-2: PROPERTIES OF GRAPHS

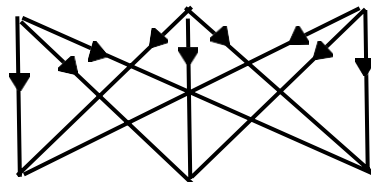
H	1	Define with example: isomorphism of graphs.	
C	2	Draw all possible different simple digraphs having three nodes up to isomorphism. Show that there is only one digraph with no edges, one with one edge, four with two edges, four with three edges, four with four edges, one with five edges and one with six edges. Assume that there are no loops.	
T	3	Define with example: degree of a node, odd node, even node, pendant node and isolated node for undirected graph.	
T	4	Define with example: indegree, outdegree and total degree for directed graph.	
C	5	Show that the sum of indegrees of all the nodes of a simple digraph is equal to the sum of outdegrees of all its nodes and that this sum is equal to the number of edges of the graph.	
H	6	Define with example: subgraph of a graph and converse of a digraph.	
T	7	Consider the following graphs: Determine the degree of each node and verify Handshaking theorem. 	

C 8 Check whether the following pair of graphs G & H are isomorphic or not with description.

(A).

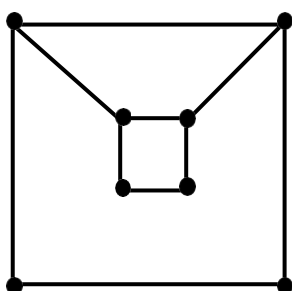


G

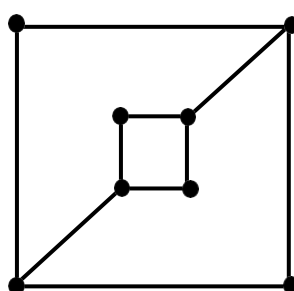


H

(B).

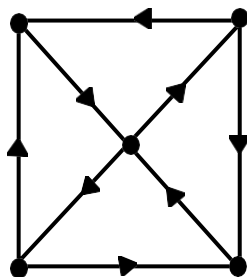


G

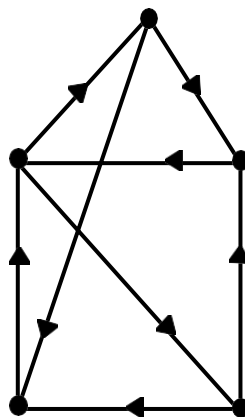


H

(C).



G



H

C	9	<p>Define node deleted subgraph and edge deleted subgraph. Also find subgraphs from the given graph <math>G</math> by deleting (I) node <math>v_1</math>, <math>G - v_1</math> &amp; (II) edge <math>e_4</math>, <math>G - e_4</math>.</p>	
H	10	<p>Define converse of a digraph and find it for given graph <math>G</math>.</p>	

### NOTE:

- ✓ In this method we introduce some additional terminology associated with a simple digraph.
- ❖ **PATH OF A GIVEN GRAPH**
  - ✓ Let  $G = (V, E)$  be a simple digraph. Consider a sequence of edges of  $G$  such that the terminal vertex of any edge in the sequence is the initial vertex of next edge. Such a sequence is called a path of the graph  $G$ .
  - ✓ A path is said to traverse through the nodes appearing in the sequence originating in the initial node of the first edge and ending in the terminal node of the last edge in the sequence.
- ❖ **LENGTH OF PATH**
  - ✓ The number of edges appearing in the sequence of a path is called the length of the path.
- ❖ **SIMPLE PATH (EDGE SIMPLE)**
  - ✓ A path in a digraph in which all the edges are distinct is called a simple path (edge simple).
- ❖ **ELEMENTARY PATH (NODE SIMPLE)**
  - ✓ A path in a digraph in which all the nodes through which it traverses are distinct is called an elementary path (node simple).

### NOTE:

- ✓ Naturally every elementary path of a digraph is also simple.

### EXAMPLE:

- ✓ Paths originating in the node 1 and ending in node 3 are

$$P_1 = \langle 1, 2 \rangle, \langle 2, 3 \rangle$$

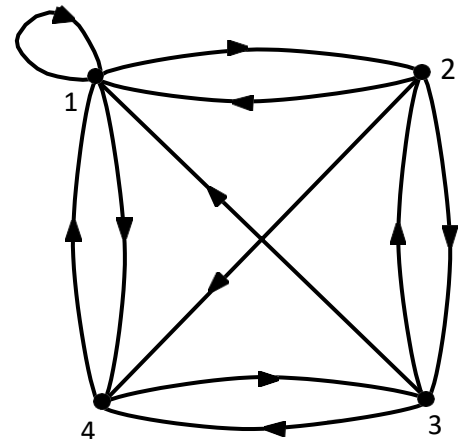
$$P_2 = \langle 1, 4 \rangle, \langle 4, 3 \rangle$$

$$P_3 = \langle 1, 2 \rangle, \langle 2, 4 \rangle, \langle 4, 3 \rangle$$

$$P_4 = \langle 1, 2 \rangle, \langle 2, 4 \rangle, \langle 4, 1 \rangle, \langle 1, 2 \rangle, \langle 2, 3 \rangle$$

$$P_5 = \langle 1, 2 \rangle, \langle 2, 4 \rangle, \langle 4, 1 \rangle, \langle 1, 4 \rangle, \langle 4, 3 \rangle$$

$$P_6 = \langle 1, 1 \rangle, \langle 1, 1 \rangle, \dots, \langle 1, 2 \rangle, \langle 2, 3 \rangle$$



- ✓ The paths  $P_1$ ,  $P_2$  &  $P_3$  of the digraph in above figure are elementary. The path  $P_5$  is simple but not elementary.
- ✓ If there exist a path from  $u$  to  $v$  then there must be an elementary path from  $u$  to  $v$ .

### ❖ CYCLE (CIRCUIT)

- ✓ A path which originates and ends in the same node is called a cycle (circuit).

### EXAMPLE:

- ✓ The following are some of the cycles in the given graph:

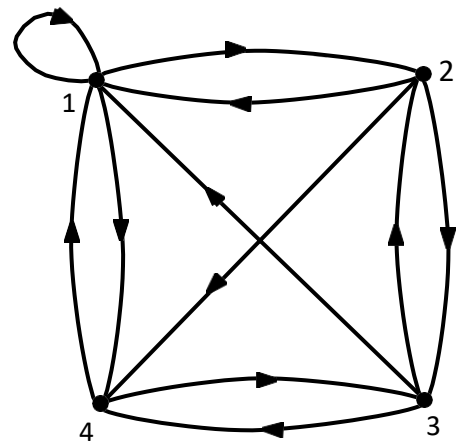
$$C_1 = \langle 1, 1 \rangle$$

$$C_2 = \langle 1, 2 \rangle, \langle 2, 1 \rangle$$

$$C_3 = \langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 3, 1 \rangle$$

$$C_4 = \langle 1, 4 \rangle, \langle 4, 3 \rangle, \langle 3, 1 \rangle$$

$$C_5 = \langle 1, 4 \rangle, \langle 4, 3 \rangle, \langle 3, 2 \rangle, \langle 2, 1 \rangle$$



### ❖ SIMPLE CYCLE

- ✓ A cycle is called simple if its path is simple path. i.e., no edge in the cycle appears more than once in the path.

### ❖ ELEMENTARY CYCLE

- ✓ A cycle is called elementary if it does not traverse through any node more than once.

### NOTE:

- ✓ In a cycle the initial node appears at least twice even if it is an elementary cycle.

- ✓ Observe that any path which is not elementary contains cycle traversing through those nodes which appear more than once in the path.
- ✓ In above graph all the cycles  $C_1, C_2, C_3, C_4$  &  $C_5$  are simple as well as elementary.

### ❖ ACYCLIC GRAPH

- ✓ A simple digraph which does not have any cycles is called acyclic. An acyclic graph does not have any loop.

### ❖ REACHABILITY

- ✓ A node  $v$  of a simple digraph is said to be reachable (accessible) from the node  $u$  of the same digraph if there exist a (at least one) path from  $u$  to  $v$ .
- ✓ It is clear from the definition that reachability is a binary relation on the set of nodes of a simple digraph. Reachability is reflexive and transitive relation. Reachability is not necessarily symmetric nor it is antisymmetric.

### ❖ GEODESIC

- ✓ If a node  $v$  is a reachable from the node  $u$  then a path of minimum length from  $u$  to  $v$  is called a geodesic.

### ❖ DISTANCE

- ✓ The length of a geodesic from the node  $u$  to the node  $v$  is called the distance and it is denoted by  $d < u, v >$ .

### ❖ DIAMETER

- ✓ The diameter of a simple digraph  $G = \langle V, E \rangle$  is given by  $\delta$ , where  $\delta = \max_{u,v \in V} d \checkmark u, v \checkmark$ .

### NOTE:

- ✓ It is assumed that  $d < u, u > = 0$  for any node  $u$ .

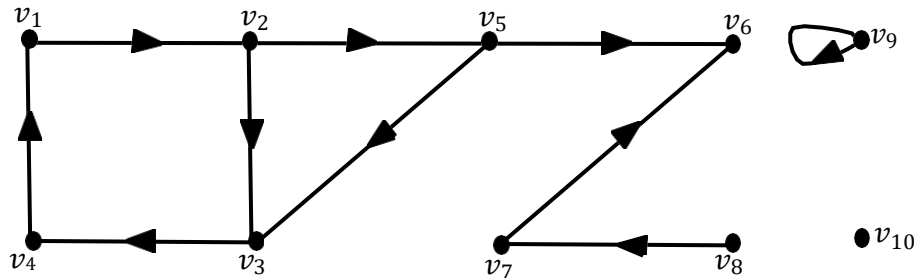
### ❖ PROPERTIES OF REACHABILITY

- ✓ If  $v$  is reachable from  $u$  then  $d < u, v >$  satisfies the following properties:
  1.  $d < u, v > \geq 0$ .
  2.  $d < u, u \geq 0$ .
  3.  $d < u, v > + d < v, w > \geq d < u, w >$ . (Triangle inequality)
- ✓ If  $v$  is not reachable from  $u$  then it is customary to write  $d < u, v > = \infty$ .
- ✓ If  $v$  is reachable from  $u$  and  $u$  is reachable from  $v$  then  $d < u, v >$  is not necessarily equal to  $d < v, u >$ .

### ❖ REACHABLE SET OF A GIVEN NODE

- ✓ The set of nodes which are reachable from a given node  $v$  is said to be the reachable set of  $v$ . The reachable set of  $v$  is written as  $R\{v\}$ . For any subset  $S \subseteq V$ , the reachable set of  $S$  is the set of nodes which are reachable from any node of  $S$ . This set is denoted by  $R(S)$ .

**EXAMPLE:**



- ✓ In the above graph all the reachable sets are given as below

$$R\{v_1\} = R\{v_2\} = R\{v_3\} = R\{v_4\} = R\{v_5\} = \{v_1, v_2, v_3, v_4, v_5, v_6\}$$

$$R\{v_6\} = \{v_6\}, R\{v_7\} = \{v_6, v_7\}, R\{v_8\} = \{v_6, v_7, v_8\}, R\{v_9\} = \{v_9\}$$

$$R\{v_{10}\} = \{v_{10}\},$$

$$R\{v_5, v_8, v_9, v_{10}\} = V = R\{v_1, v_8, v_9\}$$

✓

### ❖ NODE BASE

- ✓ In a digraph  $G = (V, E)$ , a subset  $X \subseteq V$  is called a node base if its reachable set is  $V$  and no proper subset of  $X$  has this property.
- ✓ In the above graph the set  $\{v_1, v_8, v_9, v_{10}\}$  is a node base and similarly the set  $\{v_5, v_8, v_9, v_{10}\}$  is a node base.

### ❖ CONNECTEDNESS

- ✓ An undirected graph is said to be connected if for any pair of nodes of the graph the two nodes are reachable from one another.

### ❖ WEAKLY CONNECTED

- ✓ A digraph is said to be weakly connected (connected) if it is connected as an undirected graph in which the direction of the edge is neglected. i.e., if the graph when treated as an undirected graph is connected.

### ❖ STRONGLY CONNECTED

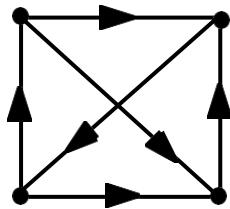
- ✓ If for any pair of nodes of the graph both the nodes of the pair are reachable from one another then the graph is called strongly connected.

### ❖ UNILATERALLY CONNECTED

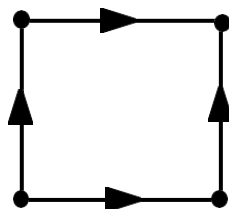
- ✓ A simple digraph is said to be unilaterally connected if for any pair of nodes of the graph at least one of the node of the pair is reachable from the other node.

### NOTE:

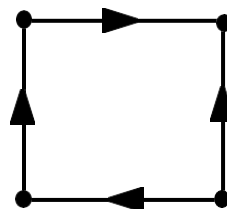
- ✓ Observe that a unilaterally connected digraph is weakly connected but a weakly connected digraph is not necessarily unilaterally connected.
- ✓ A strongly connected digraph is both unilaterally and weakly connected.



A



B

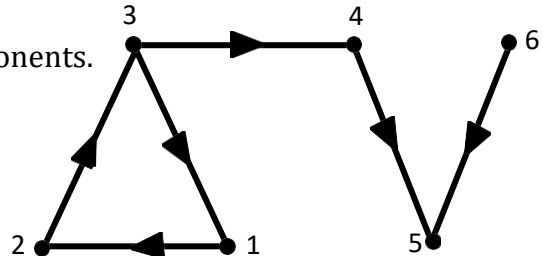


C

- ✓ The digraph in figure(A) is strongly connected, (B) is weakly connected but not unilaterally connected while (C) is unilaterally connected but not strongly connected.

### ❖ STRONG, WEAK AND UNILATERAL COMPONENTS OF A GRAPH

- ✓ For a simple digraph, a maximal strongly connected subgraph is called a strong component. Similarly, a maximal unilaterally connected subgraph is called a unilateral component and maximal weakly connected subgraph is called a weak component.
- ✓ For the digraph given in above figure  
 $\{1,2,3\}$ ,  $\{4\}$ ,  $\{5\}$ ,  $\{6\}$  are the strong components.  
 $\{1,2,3,4,5\}$ ,  $\{6\}$  are the unilateral components.  
 $\{1,2,3,4,5,6\}$  is the weak component because the graph is weakly connected.



### ❖ METHOD-3: PATH, REACHABILITY AND CONNECTEDNESS

H	1	<p>Give three elementary paths from <math>v_1</math> to <math>v_3</math> for the digraph given in following figure. Is there any cycle in the graph? What is the shortest distance between <math>v_1</math> and <math>v_3</math>.</p>	
---	---	---	--

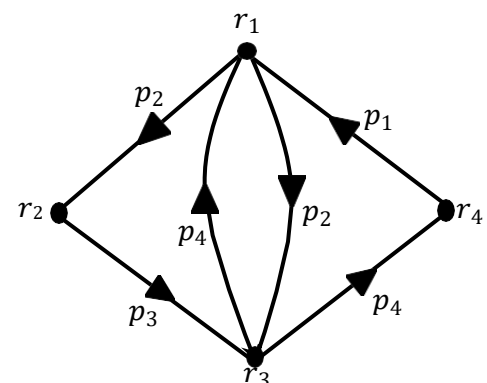


C	2	Give all elementary cycles for the following graph. Obtain an acyclic digraph by deleting one edge of the given digraph.	
C	3	Prove that in a simple digraph, the length of any elementary path is less than or equal to $n-1$ , where $n$ is the number of nodes in the graph. Similarly, the length of any elementary cycle does not exceed $n$ .	
T	4	Find the diameter of the digraphs given in the examples 1 and 2.	
H	5	Find the reachable set of $\{v_1, v_4\}$ , $\{v_4, v_5\}$ and $\{v_3\}$ for the digraph given in example-2.	
C	6	Find the reachable set for all the nodes in the following digraph.	
C	7	Find a node base for each of the digraphs given in the examples 1 and 2.	
C	8	Determine whether the digraphs in example 1 and 2 are strongly, weakly or unilaterally connected.	
H	9	Find the strong, weak and unilateral component for the digraph given in the example 2.	
T	10	Find the strong, weak and unilateral component of the digraph given in the following figure.	

### ❖ APPLICATIONS TO REPRESENT RESOURCE ALLOCATION STATUS OF AN OPERATING SYSTEM AND DETECTION AND CORRECTION OF DEADLOCKS:

- ✓ We shall now show how a simple digraph can be used to represent the resource allocation status of an operating system.
- ✓ In a multi programmed computer system it appears that several programs are executed at one time. In reality, the programs are sharing the resources of the computer system such as tape units, disk devices, the central processor, main memory and compilers. A special set of programs called an operating system controls the allocation of these resources to the programs. When a program requires the use of a certain resource and the operating system must ensure that the request is satisfied.
- ✓ It may happen that requests for resources are in conflict. For example, program A may have control of resource  $r_1$  and require resource  $r_2$  but program B has control of resource  $r_2$  and requires resource  $r_1$ . In such a case the computer system is said to be in a state known as deadlock and the conflicting requests must be resolved. A directed graph can be used to model resource requests and assist in the detection and correction of deadlocks.
- ✓ It is assumed that all resource requests of a program must be satisfied before that program must be satisfied before that program can complete execution. If any requested resources are unavailable at the time of the request the program will assume control of the resources which are available but must wait for the unavailable resources.
- ✓ Let  $P_t = \{p_1, p_2, \dots, p_m\}$  represent the set of programs in the computer system at time  $t$ . Let  $A_t \subseteq P_t$  be the set of active programs or programs that have been allocated at least a portion of their resource requests at time  $t$ . Let  $R_t = \{r_1, r_2, \dots, r_n\}$  represent the set of resources in the system at time  $t$ . An allocation graph  $G_t$  is a directed graph representing the resource allocation status of the system at time  $t$  and consisting of a set of nodes  $V = R_t$  and a set of edges  $E$ . Each resource is represented by a node of the graph. There is a directed edge from node  $r_i$  to  $r_j$  if and only if there is a program  $p_k$  in  $A_t$  that has been allocated resource  $r_i$  but is waiting for  $r_j$ .

- ✓ For example, let  $R_t = \{r_1, r_2, r_3, r_4\}$  and  $A_t = \{p_1, p_2, p_3, p_4\}$  and the resource allocation status be
  - $p_1$  has resource  $r_4$  and requires  $r_1$
  - $p_2$  has resource  $r_1$  and requires  $r_2$  and  $r_3$
  - $p_3$  has resource  $r_2$  and requires  $r_3$
  - $p_4$  has resource  $r_3$  and requires  $r_1$  and  $r_4$ .



Then the allocation graph at time  $t$  is given in figure.

- ✓ It can be shown that the state of deadlock exists in a computer system at time  $t$  if and only if the allocation graph  $G_t$  contains strongly connected components. In the case of our example the graph  $G_t$  is strongly connected.

### ❖ MATRIX REPRESENTATION OF A GRAPH

- ✓ A diagrammatic representation of a graph has limited usefulness. Furthermore, such a representation is only possible when the number of nodes and edges is reasonably small. An alternating method of representing graphs using matrices has several advantages. It is easy to store and manipulate matrices and the graphs represented by them in a computer. Well known operations of matrix algebra can be used to calculate paths, cycles and other characteristics of a graph.
- ✓ Given a simple digraph  $G = \langle V, E \rangle$ , it is necessary to assume some kind of ordering of the nodes of the graph in the sense that a particular node is called a first node, another a second node and so on. Our matrix representation of  $G$  depends upon the ordering of the nodes.

### ❖ ADJACENCY MATRIX

- ✓ Let  $G = \langle V, E \rangle$  be a simple digraph in which  $V = \{v_1, v_2, \dots, v_n\}$  and the nodes are assumed to be ordered from  $v_1$  to  $v_n$ . An  $n \times n$  matrix  $A$  whose elements  $a_{ij}$  are given by

$$a_{ij} = \begin{cases} 1 & \text{if } \langle v_i, v_j \rangle \in E \text{ If there is an edge from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

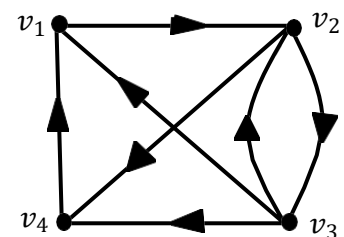
is called the adjacency matrix of the graph  $G$ .

- ✓ Recall that the adjacency matrix is the same as the relation matrix or the incidence matrix of the relation  $E$  in  $V$ . Any element of the adjacency matrix is either 0 or 1.

### EXAMPLE:

- ✓ The adjacency matrix of the above graph is  $A =$

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$



### NOTE:

- ✓ The sum of all 1's in a row indicates the outdegree of the corresponding node.
- ✓ The sum of all 1's in a column indicates the indegree of the corresponding node.

### EXAMPLE:

- ✓ From the adjacency matrix( $A$ ) of the above digraph we can calculate outdegree, indegree and total degree of each nodes which as follow.

Node	Outdegree	Indegree	Total degree
$v_1$	1	2	3
$v_2$	2	2	4
$v_3$	3	1	4
$v_4$	1	2	3

- ✓ An adjacency matrix completely defines a simple digraph.

#### ❖ BOOLEAN (BIT) MATRIX

- ✓ Any matrix whose elements are either 0 or 1 is called a Boolean matrix or bit matrix.

#### NOTE:

- ✓ For a given digraph  $G = \langle V, E \rangle$ , an adjacency matrix depends upon the ordering of the elements of  $V$ . For different ordering of the elements of  $V$  we get different adjacency matrices of the same graph  $G$ .
- ✓ However, any one of the adjacency matrices of  $G$  can be obtained from another adjacency matrix of the same graph by interchanging some of the rows and the corresponding column of the matrix. But the digraphs of both the matrix are isomorphic.
- ✓ If a digraph is reflexive then the diagonal elements of the adjacency matrix are 1s.
- ✓ The adjacency matrix for a symmetric digraph is also symmetric. i.e.,  $a_{ij} = a_{ji}$  for all  $i$  and  $j$ .
- ✓ If a digraph is antisymmetric, then  $a_{ij} = 1$  implies  $a_{ji} = 0$  and  $a_{ij} = 0$  implies that  $a_{ji} = 1$  for all  $i$  and  $j$ .
- ✓ For a null graph which consists of only  $n$  nodes but no edges, the adjacency matrix is a null matrix.
- ✓ If there are loops at each node but no other edges in the graph then the adjacency matrix is the identity matrix.
- ✓ If  $G = \langle V, E \rangle$  is a simple digraph whose adjacency matrix is  $A$  then the adjacency matrix of  $G$ , the converse of  $G$ , is the transpose of  $A$ , that is  $A^T$ .

#### ❖ PATH (REACHABILITY) MATRIX OF A GRAPH

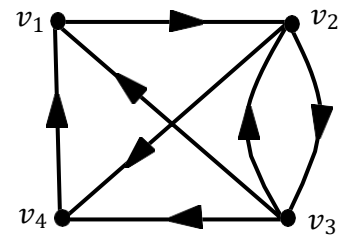
- ✓ Let  $G = \langle V, E \rangle$  be a simple digraph in which  $|V| = n$  and the nodes of  $G$  are assumed to be ordered. An  $n \times n$  matrix  $P$  whose elements are given by

$$p_{ij} = \begin{cases} 1 & \text{if there exists a path from } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

is called the path matrix (reachability matrix) of the graph  $G$ .

## EXAMPLE:

- ✓ The path matrix of given graph is  $P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ .



## NOTE:

- ✓ The path matrix can be calculated from the matrix  $B_n$  by choosing  $p_{ij} = 1$  if the element  $b_{n_{ij}}$  of  $B_n$  is nonzero and  $p_{ij} = 0$  if the element  $b_{n_{ij}}$  of  $B_n$  is zero. Where,  $B_n = A + A^2 + A^3 + \dots + A^n$ .  $A$  is adjacency matrix of given graph and  $n$  is number of nodes in given graph.

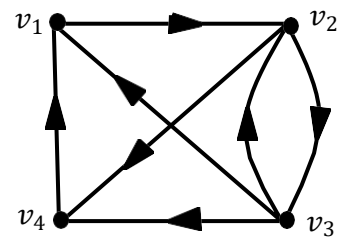
## ❖ DETERMINE NUMBER OF PATHS OF LENGTH N THROUGH ADJACENCY

### MATRIX:RESULT:

- ✓ Let  $A$  be the adjacency matrix of a digraph  $G$ . The element in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of  $A^n$  ( $n$  is a nonnegative integer) is equal to the number of paths of length  $n$  from the  $i^{\text{th}}$  node to the  $j^{\text{th}}$  node.

## EXAMPLE:

- ✓ Find the path matrix of given graph using adjacency matrix. Also find number of paths of length 4 between  $v_2$  &  $v_4$  from the adjacency matrix and mention it from the given graph.



- ✓ The adjacency matrix of the given graph is  $A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$ .

- ✓ Here no. of nodes in the given graph is  $|V| = 4$ .

$$\text{Hence, } B_n = B_4 = A + A^2 + A^3 + A^4$$

$$= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 1 \\ 2 & 2 & 1 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 1 & 1 \\ 2 & 2 & 2 & 3 \\ 3 & 3 & 2 & 3 \\ 2 & 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 3 & 4 & 2 & 3 \\ 5 & 5 & 4 & 6 \\ 7 & 7 & 4 & 7 \\ 3 & 2 & 1 & 2 \end{bmatrix}$$

- ✓ From  $B_4$  we conclude that all entries are nonzero implies all entries of path matrix  $P$  are 1(one).

- ✓ Hence, path matrix  $P$  for the given graph is  $P = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$ .

- ✓ The number of paths of length 4 between  $v_2$  &  $v_4$  is the element of 2<sup>nd</sup> row and 4<sup>th</sup> column of  $A^4$ , which is 3.

- ✓ The paths of length 4 from  $v_2$  to  $v_4$  are

$$P_1 = \tilde{I} < v_2, v_3 >, < v_3, v_1 >, < v_1, v_2 >, < v_2, v_4 > \tilde{I}$$

$$P_2 = \tilde{I} < v_2, v_4 >, < v_4, v_1 >, < v_1, v_2 >, < v_2, v_4 > \tilde{I}$$

$$P_3 = \tilde{I} < v_2, v_3 >, < v_3, v_2 >, < v_2, v_3 >, < v_3, v_4 > \tilde{I}.$$

### NOTE:

- ✓ The path matrix only shows the presence or absence of at least one path between a pair of points and also the presence or absence of a cycle at any node. It does not show all the paths that may exist. In this sense a path matrix does not complete information about a graph as does the adjacency matrix. The path matrix is important in its own right.
- ✓ It may be remarked that if we are interested in knowing the reachability of one node from another, it is sufficient to calculate  $B_{n-1} = A + A^2 + \dots + A^{n-1}$ , because a path of length  $n$  cannot be elementary for graph with  $n$  nodes.
- ✓ For the purpose of reachability, every node is assumed to be reachable from itself.

### ❖ WARSHALL'S ALGORITHM TO PRODUCE PATH MATRIX:

- ✓ Let  $G$  be a directed graph with  $m$  vertices  $v_1, v_2, \dots, v_m$ . Suppose we want to find the path matrix  $P$  of the graph  $G$ . Warshall gave an algorithm which is much more efficient than calculating the powers of the adjacency matrix  $A$ .
- ✓ First we define  $m$ -square Boolean matrices  $P_0, P_1, \dots, P_m$  as follows. Let  $P_k[i, j]$  denote the  $ij^{\text{th}}$  entry of the matrix  $P_k$ . Then we define:

$$P_k[i, j] = \begin{cases} 1 & \text{if there is a simple path from } v_i \text{ to } v_j \text{ which does not use} \\ & \text{any others vertices except possibly } v_1, \dots, v_k. \\ 0 & \text{otherwise} \end{cases}$$

- ✓ That is  $P_0[i, j] = 1$  if there is an edge from  $v_i$  to  $v_j$ .  
 $P_1[i, j] = 1$  if there is a simple path from  $v_i$  to  $v_j$  which does not use any other vertex except possibly  $v_1$ .  
 $P_2[i, j] = 1$  if there is a simple path from  $v_i$  to  $v_j$  which does not use any other vertices except possibly  $v_1$  and  $v_2$ . And so on.
- ✓ Observe that the first matrix  $P_0 = A$  is the adjacency matrix of  $G$ . Furthermore, since  $G$  has only  $m$  vertices, the last matrix  $P_m = P$  is the path matrix of  $G$ .
- ✓ Warshall observed that  $P_k[i, j] = 1$  can occur only if one of the following two cases occurs:
  1. There is a simple path from  $v_i$  to  $v_j$  which does not use any other vertices except possibly  $v_1, v_2, \dots, v_{k-1}$ ; hence  $P_{k-1}[i, j] = 1$ .
  2. There is a simple path from  $v_i$  to  $v_k$  and a simple path from  $v_k$  to  $v_j$  where each simple path does not use any other vertices except possibly  $v_1, v_2, \dots, v_{k-1}$ . Hence  $P_{k-1}[i, k] = 1$  &  $P_{k-1}[k, j] = 1$ .

These two cases are pictured as follows:

(1)  $v_i \rightarrow \dots \rightarrow v_j$  and (2)  $v_i \rightarrow \dots \rightarrow v_k \rightarrow \dots \rightarrow v_j$

- ✓ Here  $\rightarrow \dots \rightarrow$  denotes part of a simple path which does not use any other vertices except possibly  $v_1, v_2, \dots, v_{k-1}$ . Accordingly the elements of  $P_k$  can be obtained by

$$P_k[i, j] = P_{k-1}[i, j] \vee \neg P_{k-1}[i, k] \wedge P_{k-1}[k, j] \neg$$

- ✓ where we use the logical operations of  $\wedge$  AND and  $\vee$  OR. In other words we can obtain each entry in the matrix  $P_k$  by looking at only three entries in the matrix  $P_{k-1}$ .

### ❖ ALGORITHM:

- ✓ A directed graph  $G$  with  $M$  vertices is maintained in memory by its adjacency matrix  $A$ . This algorithm finds the (Boolean) path matrix  $P$  of the graph  $G$ .

- ✓ Step 1.

Repeat for  $I, J = 1, 2, \dots, M$ : [initializes  $P$ .]

If  $A[I, J] = 0$ , then: Set  $P[I, J] := 0$ ;

Else: Set  $P[I, J] := 1$ .

[End the loop.]

- ✓ Step 2.

Repeat steps 3 and 4 for  $K = 1, 2, \dots, M$ : [Updates  $P$ .]

- ✓ Step 3.

Repeat Step 4 for  $I = 1, 2, \dots, M$ :

- ✓ Step 4.

Repeat for  $J = 1, 2, \dots, M$ :

Set  $P[I, J] := P[I, J] \vee \neg P[I, K] \wedge P[K, J] \neg$ .

[End of loop.]

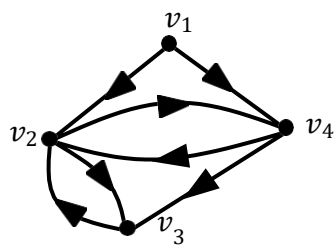
[End of step 3 loop.]

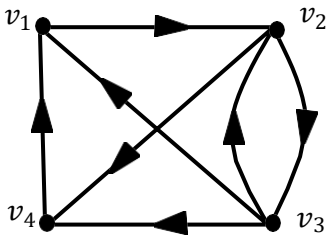
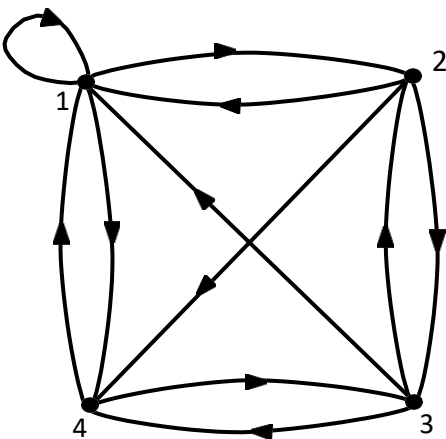
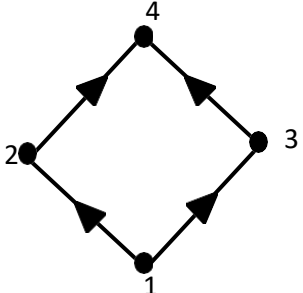
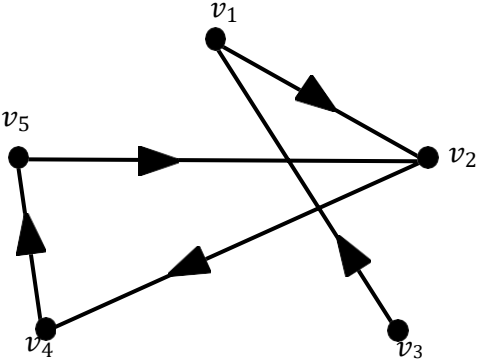
[End of step 2 loop.]

- ✓ Step 5.

Exit.

### ❖ METHOD-4: MATRIX REPRESENTATION OF A GRAPH

T	1	Obtain the adjacency matrix $A$ of the digraph given in the figure. Find the elementary paths of lengths 1 and 2 from $v_1$ to $v_4$ . Show that there is also a simple path of length 4 from $v_1$ to $v_4$ . Verify the results by calculating $A^2, A^3$ & $A^4$ .	
---	---	---	---

C	2	Find A and $A^2$ without matrix multiplication for the given digraph. Where, A is an adjacent matrix.	
T	3	From the adjacency matrix of given graph calculate outdegree, indegree and total degree of each nodes and also verify it from the graph.	
H	4	Find the adjacency matrix A and path matrix from $B_4 = A + A^2 + A^3 + A^4$ for the given graph. Also verify path matrix from given graph.	
C	5	Apply Warshall's algorithm to produce a path matrix for given graph.	



### ❖ INTRODUCTION OF TREE

- ✓ Trees are useful in describing any structure which involves hierarchy. Familiar examples of such structures are family trees, the decimal classifications of books in a library, the hierarchy of positions in an organization, an algebraic expression involving operations for which certain rules precedence are prescribed, etc.

### ❖ ACYCLIC GRAPH

- ✓ A digraph which does not have any cycle is called acyclic graph.

### ❖ TREE

- ✓ A tree is a connected acyclic graph.

### ❖ DIRECTED TREE

- ✓ A directed tree is an acyclic graph which has one node called root with indegree 0, while all other nodes have indegree 1.

### NOTE:

- ✓ Every directed tree must have at least one node. An isolated node is also a directed tree.

### ❖ FOREST

- ✓ A set of disjoint trees is called a forest.

### ❖ ROOT

- ✓ A directed tree which has a node with indegree 0 is called roots of tree.

### ❖ LEAF (TERMINAL) NODE

- ✓ In a directed tree, any node which has outdegree 0 is called a terminal node or a leaf.

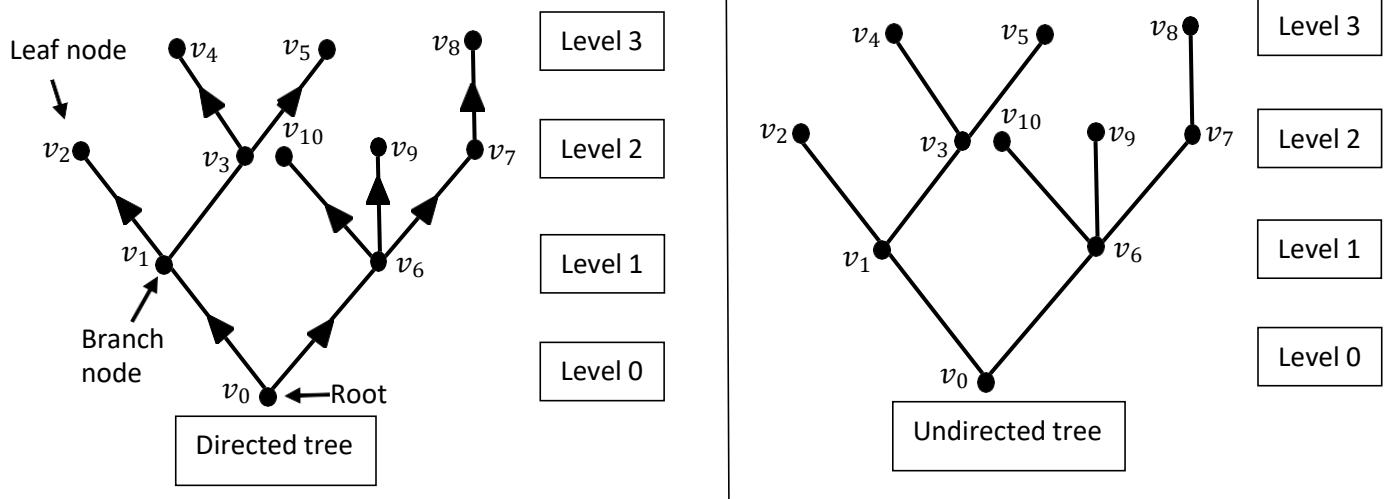
### ❖ BRANCH NODE

- ✓ The nodes which are not terminal nodes are known as branch nodes.

### ❖ LEVEL OF A NODE

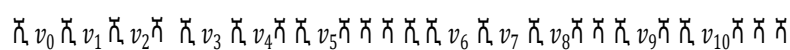
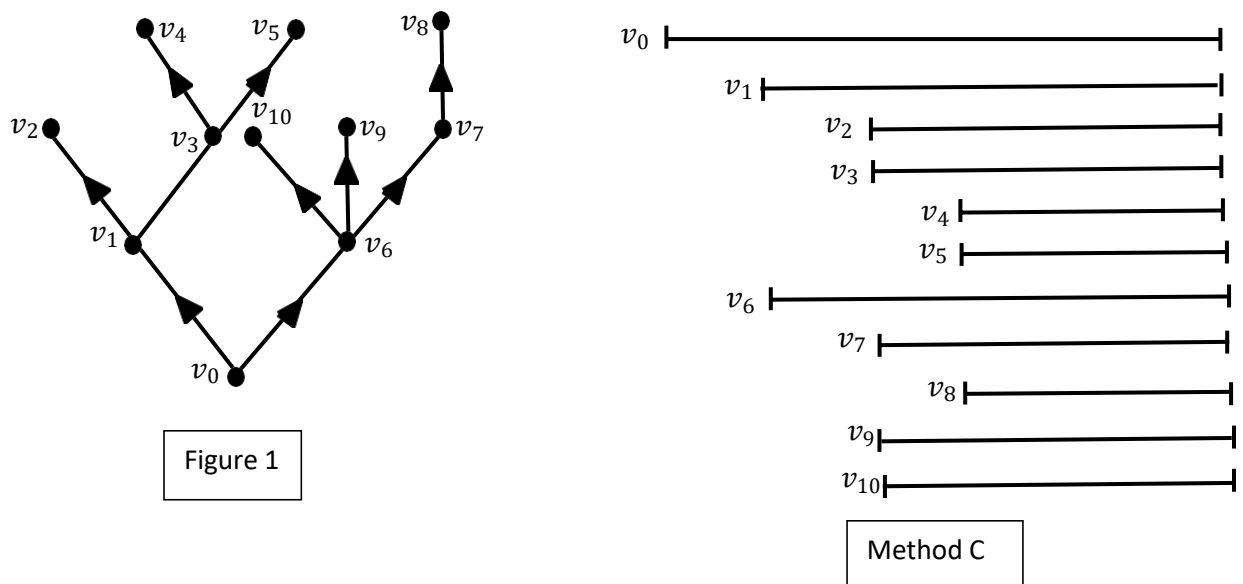
- ✓ The level of any node is the length of its path from the root.
- ✓ The level of the root of a directed tree is 0, while the level of any node is equal to its distance from the root.
- ✓ Observe that all the paths in a directed tree are elementary and the length of a path from any node to another node if such a path exists is the distance between the nodes because a directed tree is acyclic.

**EXAMPLE:**

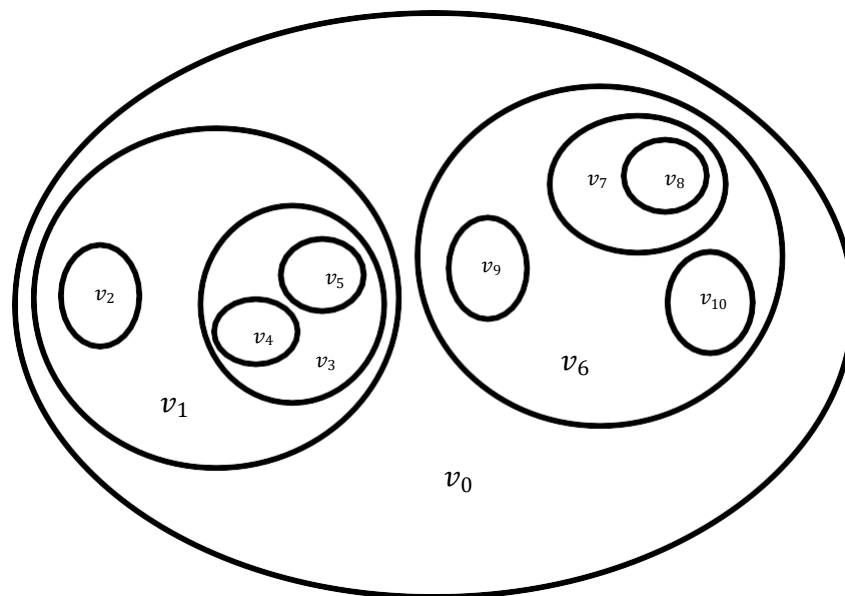


## ❖ DIFFERENT REPRESENTATIONS OF A TREE

- ✓ There are several other ways in which a directed tree can be represented graphically.
- ✓ These methods of representation for the directed tree of figure 1 are given in figures A, B and C. The method(A) uses the familiar technique of Venn diagrams to show subtrees, the method(B) uses the convention of nesting parentheses and the method(C) method is the one used in the list of contents of books.



### Method B



Method A

### ❖ BINARY TREE

- ✓ In a directed tree the outdegree of every node is less than or equal to 2 then the tree is called binary tree.

**EXAMPLE:** Figure A shows binary tree.

### ❖ FULL (COMPLETE) BINARY TREE

- ✓ If the outdegree of every node is exactly equal to 2 or 0 then the tree is called a full or complete binary tree.

**EXAMPLE:** Figure B shows

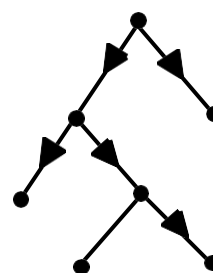
binarytree.

### ❖ M-ARY TREE

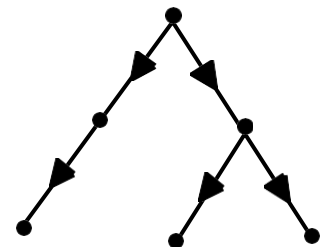
- ✓ In a directed tree the outdegree of every node is less than or equal to m then the tree is called m-ary tree.

**EXAMPLE:** Figure C shows 3-

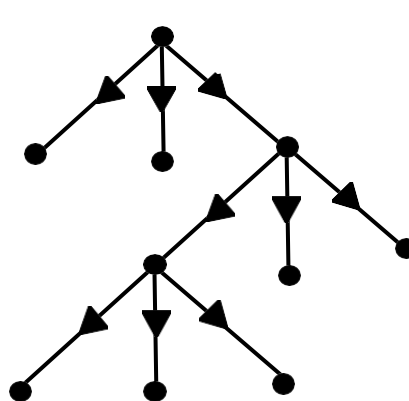
arytree.



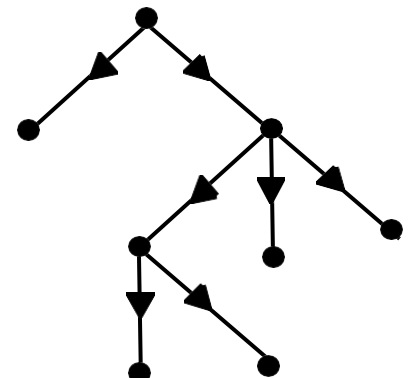
A



B



C



D

### ❖ FULL (COMPLETE) M-ARY TREE

- ✓ If the outdegree of every node is exactly equal to  $m$  or  $0$  then the tree is called a full or complete  $m$ -ary tree.

**EXAMPLE:** Figure D shows full 3-ary tree.

### ❖ DESCENDENT OF NODE U

- ✓ The node which is reachable from  $u$  is called descendent of  $u$ .

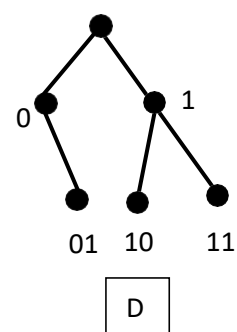
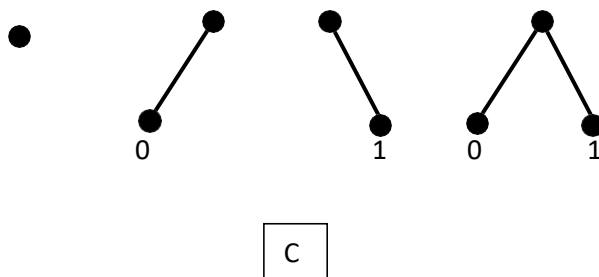
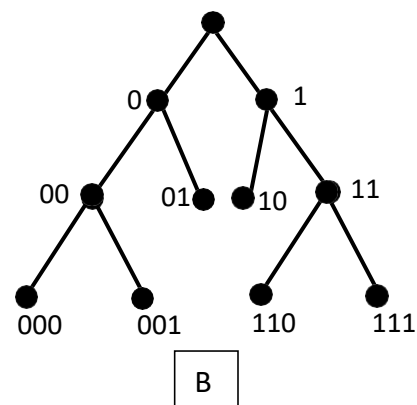
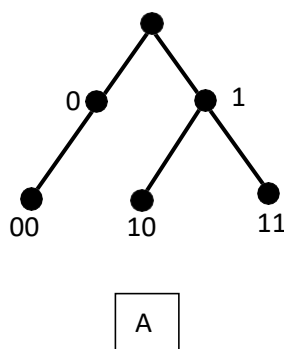
### ❖ SON OF NODE U

- ✓ The node which is reachable from  $u$  through a single edge is called son of  $u$ .

### ❖ POSITIONAL M-ARY TREE

- ✓ If we consider  $m$ -ary trees in which the  $m$  sons of any node are assumed to have  $m$  distinct positions. If such positions are taken into account then the tree is called a positional  $m$ -ary tree.

**EXAMPLE:**



- ✓ Figure A shows a binary tree, B shows a full binary tree and C shows all four possible arrangements of sons of a node in a binary tree. The binary trees shown in figure A and D are distinct positional trees although they are not distinct ordered trees. In a positional binary tree, every node is uniquely represented by a string over the alphabet  $\{0, 1\}$ , the root being represented by an empty string. Any son of a node  $u$  has a string which is prefixed by the string of  $u$ . The string of any terminal node is not prefixed to the string of any other node. The

set of strings which correspond to terminal node from a prefix code. Thus, the prefix code of the binary tree in B is  $\{000, 001, 01, 10, 110, 111\}$ . A similar representation of nodes of a positional m-ary tree by means of string over an alphabet  $\{0, 1, \dots, m-1\}$  is possible.

### ❖ CONVERTING ANY M-ARY TREE TO A BINARY TREE

Delete all the branches originating in every node except the left most branch.

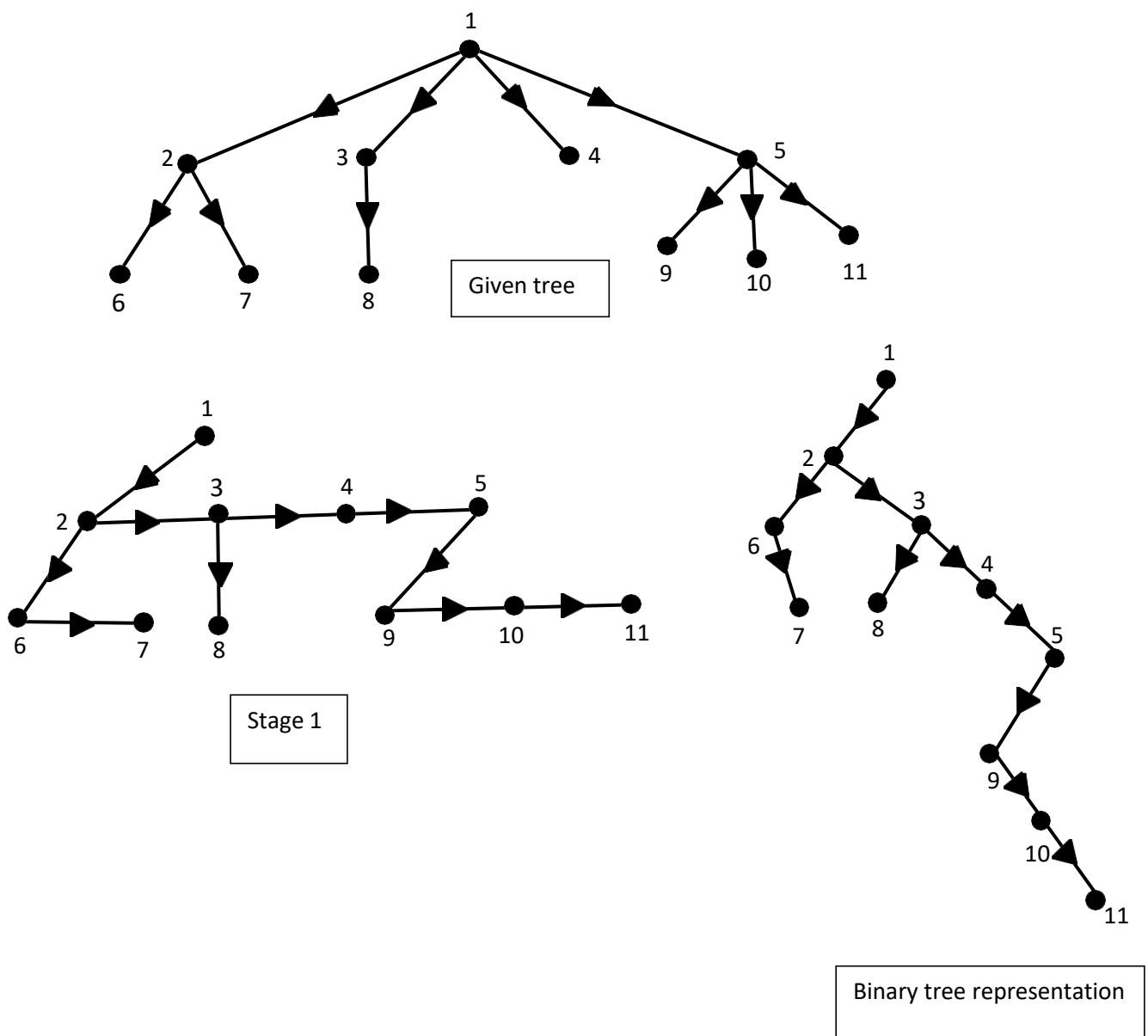
Draw edges from a node to the node on the right, if any, which is situated at the same level.

Choose its left and right sons as below.

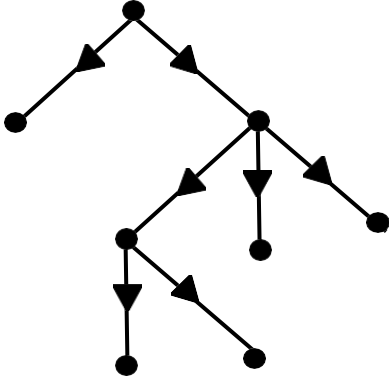
(a). The left son is the node which is immediately below the given node.

(b). The right son is the node to the immediate right of the given node on some horizontal line.

**EXAMPLE:**



### ❖ METHOD-5: PROPERTIES OF TREE AND ITS REPRESENTATIONS

H	1	Define with example: acyclic graph, tree, directed tree, forest, root, leaf node, branch node and level of a node.	
T	2	Explain different representation of a tree with example.	
H	3	Define with example: binary tree, complete binary tree, m-ary tree and complete m-ary tree.	
C	4	Write the steps of converting any m-ary tree to a binary tree and convert the following tree into a binary tree.	

### ❖ REPRESENTATIONS OF A BINARY TREE

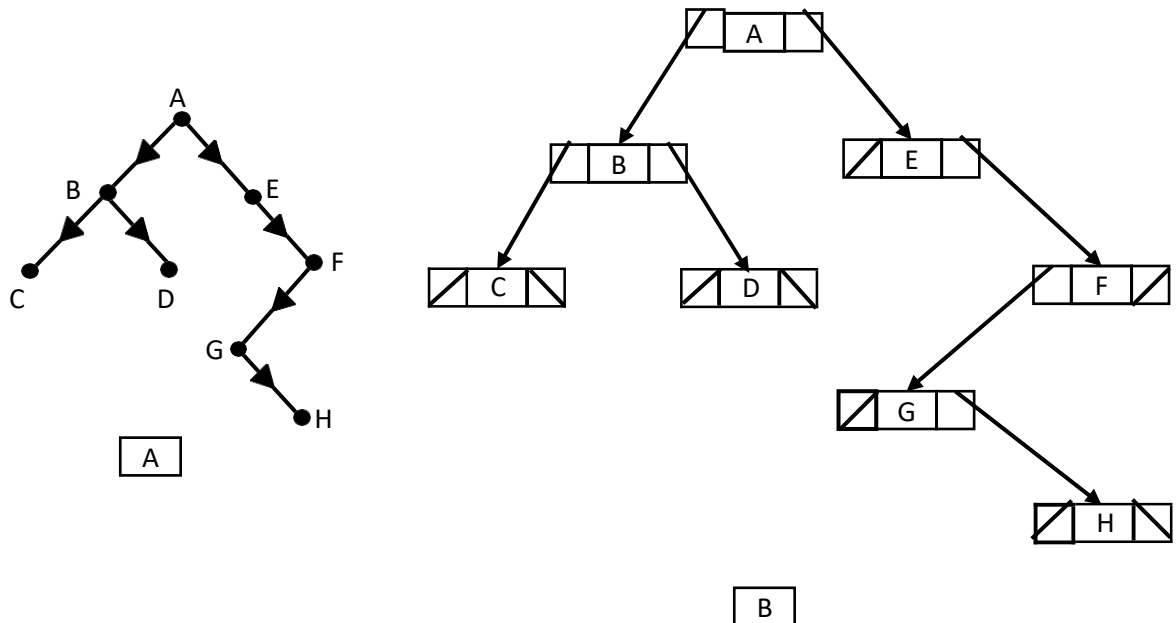
- ✓ The representation of a binary tree is simple compared to those for general tree.

### ❖ LINKED-LIST

- ✓ Computer representation of trees based on linked allocation seems to be more popular because of the ease with which nodes can be inserted in and deleted from a tree and because tree structures can grow to an arbitrary size, a size which is often unpredictable.
- ✓ Linked allocation techniques will be used to represent binary trees. A number of possible traversals which can be performed on binary trees are described. The subsections end with a symbol table algorithm based on a tree structure.
- ✓ A binary tree has one root node with no descendants or else a left, or a right, or a left and right subtree descendant(s). Each subtree descendant is also a binary tree and we do make the distinction between its left and right branches. A convenient way of representing binary trees is to use linked allocation techniques involving nodes with structure where LLINK or RLINK contain a pointer to the left subtree respectively of the node in question. Data contains the information which is to be associated with this particular node. Each pointer can have a value NULL.
 

LLINK	DATA	RLINK
-------	------	-------
- ✓ An example of a binary tree as a graph and its corresponding linked representation in memory are given in figure A and B respectively. Observe the very close similarity between

the figures as drawn. Such a similarity illustrates that the linked storage representation of a tree is closer to the logical structuring of the data involved. This property can be useful in designing algorithms which process tree structures.



### ❖ TREE TRAVERSAL

- ✓ Tree traversal is a procedure by which each node is processed exactly once in some systematic manner.

### ❖ PRE-ORDER TRAVERSAL

- ✓ Process the root node.  
Traverse the left subtree in pre-order.  
Traverse the right subtree in pre-order.

### ❖ IN-ORDER TRAVERSAL

- ✓ Traverse the left subtree in in-order.  
Process the root node.  
Traverse the right subtree in in-order.

### ❖ POST ORDER TRAVERSAL

- ✓ Traverse the left subtree in post-order.  
Traverse the right subtree in post-order.  
Process the root node.

### EXAMPLE:

- ✓ The pre-order, in-order and post-order traversals of the tree given in following table which process the nodes in the following order:

ABCDEFGH (pre-order)

CBDAEFGH (in-order)

CDBHGF EA (post-order)

- ✓ The following table shows the trace of algorithm preorder for the above graph

Stack contents	P	Visit P	Output String
	NA	A	A
NE	NB	B	AB
NE ND	NC	C	ABC
NE ND	NULL		
NE	ND	D	ABCD
NE	NULL		
	NE	E	ABCDE
NF	NULL		
	NF	F	ABCDEF
	NG	G	ABCDEFG
NH	NULL		
	NH	H	ABCDEFGH
	NULL		

❖ **ALGORITHM PREORDER:**

- ✓ Given a binary tree whose root node address is given by a variable T and whose node structure is the same as previously described, this algorithm traverses the tree in preorder. An auxiliary stack S is used and TOP is the index of the top element of S. P is a temporary variable which denotes when we are in the tree.
1. [Initialize] If  $T = \text{NULL}$ , then Exit (the tree has no root and therefore is not a proper binary tree); otherwise set  $P \leftarrow T$  and  $\text{TOP} \leftarrow 0$ .
  2. [Visit node, stack right branch address and go left] Process node P. If  $\text{RLINK}[P] \neq \text{NULL}$ , then set  $\text{TOP} \leftarrow \text{TOP} + 1$  and  $S[\text{TOP}] \leftarrow \text{RLINK}[P]$ . Set  $P \leftarrow \text{LLINK}[P]$ .
  3. [End of chain?] If  $P \neq \text{NULL}$ , then go to step 2.
  4. [Unstack a right branch address] If  $\text{TOP} = 0$ , then Exit; otherwise set  $P \leftarrow S[\text{TOP}]$ ,  $\text{TOP} \leftarrow \text{TOP} - 1$ , and go to step 2.
- ✓ In the second and third steps of the algorithm, we visit and process a node. The address of the right branch of such a node, if it exists, is stacked and a chain of left branches is followed until this chain ends. At this point we enter step 4 and delete from the stack the address of



the root node of the most recently encountered right subtree and process it according to steps 2 and 3. A trace of the algorithm for the binary tree given in the above graph appears in above table, where the rightmost element in the stack is considered to be its top element and the notation “NE,” for example, denotes the address of node E. The visit of a node in this case merely involves the output of the label for that node.

❖ **ALGORITHM POSTORDER:**

- ✓ The same node structure described previously is assumed and T is again a variable which contains address of the root of the tree. A stack S with its top element pointer is also required, but in this case each node will be stacked twice namely once when its left subtree is traversed and once when its right subtree is traversed. On completion of these two traversals, the particular node being considered is processed. Hence, we must be able to distinguish two types of stack entries. The first type of entry indicates that a left subtree is being traversed, while the second indicates the traversal of a right subtree. For convenience we will use negative pointer values for the second type of entry. This of course assumes that valid pointer data is always nonzero and positive.
- 1. [Initialize] If  $T = \text{NULL}$ , then Exit (the tree has no root and therefore is not a proper binary tree); otherwise set  $P \leftarrow T$  and  $\text{TOP} \leftarrow 0$ .
- 2. [Stack node address and go left] Set  $\text{TOP} \leftarrow \text{TOP} + 1$ ,  $S[\text{TOP}] \leftarrow P$  and  $P \leftarrow \text{LLINK}[P]$ .
- 3. [End of chain?] If  $P \neq \text{NULL}$ , then go to step 2.
- 4. [Unstack a node address] If  $\text{TOP} = 0$ , then Exit; otherwise set  $P \leftarrow S[\text{TOP}]$ ,  $\text{TOP} \leftarrow \text{TOP} - 1$ , and go to step 2.
- 5. [Restack address if right subtree is not traversed] If  $P < 0$ , then go to step 6; otherwise set  $\text{TOP} \leftarrow \text{TOP} + 1$ ,  $S[\text{TOP}] \leftarrow -P$ ,  $P \leftarrow \text{RLINK}[P]$ , and go to step 3.
- 6. [Visit node] Set  $P \leftarrow -P$ , process node P, and go to step 4.
- ✓ In the second and third steps, a chain of left branches is followed and the address of each node which is encountered is stacked. At the end of such a chain, the stack entry for the last node encountered is checked against zero. If it is positive, the negative address of that node is restacked and the right branch of this node is taken and processed according to steps 2 and 3. If the stack value is negative however we have finished traversing the right subtree of that node. The node is then processed and the next stack entry is subsequently checked.

❖ **APPLICATIONS OF LIST STRUCTURES AND GRAPHS**

- ✓ Representation of a structure which is more general than a tree such a structure is called a list structure and several programming languages have been developed to allow easy programming languages have been developed to allow easy processing of structures similar

to those that will be described. The need for list processing arose from the high cost of rapid computer storage and the unpredictable nature of the storage requirements of computer programs and data. There are many symbol manipulation applications in which this unpredictability is particularly acute. It will be shown that a list structure can be used to represent a directed graph. The representations of a general graph structures are based not only nature of the data but also on the operations which are to be performed on the data.

### ❖ METHOD-6: TYPES OF TREE TRAVERSAL AND ITS ALGORITHMS

T	1	Explain representation of a binary tree by linked allocation technique with example.	
H	2	Define: tree traversal, pre-order traversal, in-order traversal and post-order traversal.	
C	3	Write algorithm on pre-order traversal.	
C	4	Write algorithm on post-order traversal.	
H	5	Discuss application of list structures and graphs.	