* Representation of Graphs:

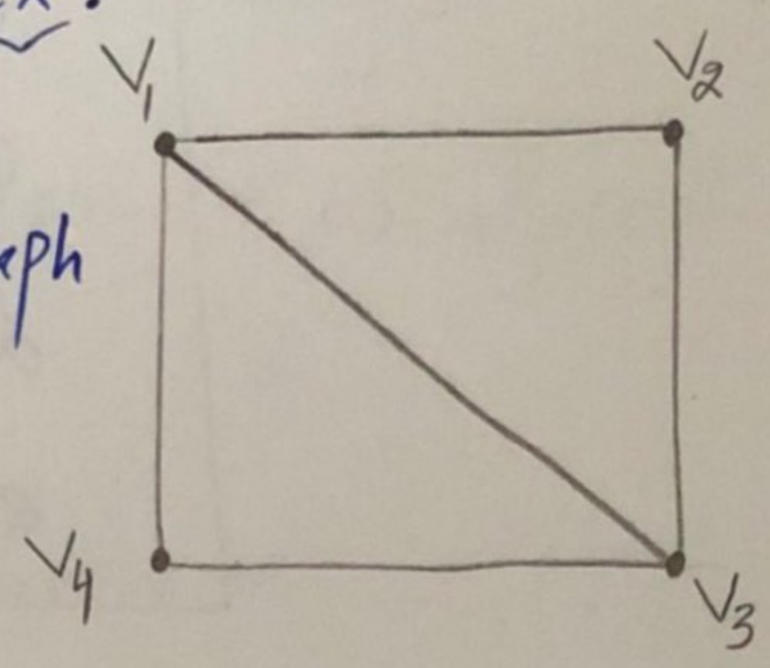
(1) Adjacency matrix
(2) Incidence matrix

] ⟶ Matrix Representation

(3) Adjacency List (Linked representation/adjacency structure representation)

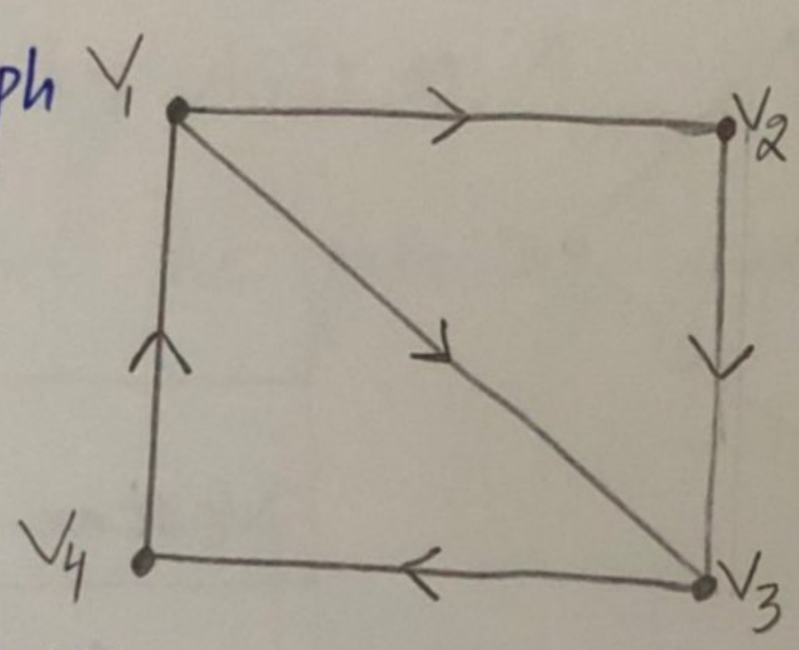
(1) Adjacency Matrix:

(a) Undirected graph



$$A = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 1 & 1 & 1 \\ v_2 & 1 & 0 & 1 & 0 \\ v_3 & 1 & 1 & 0 & 1 \\ v_4 & 1 & 0 & 1 & 0 \end{array}$$

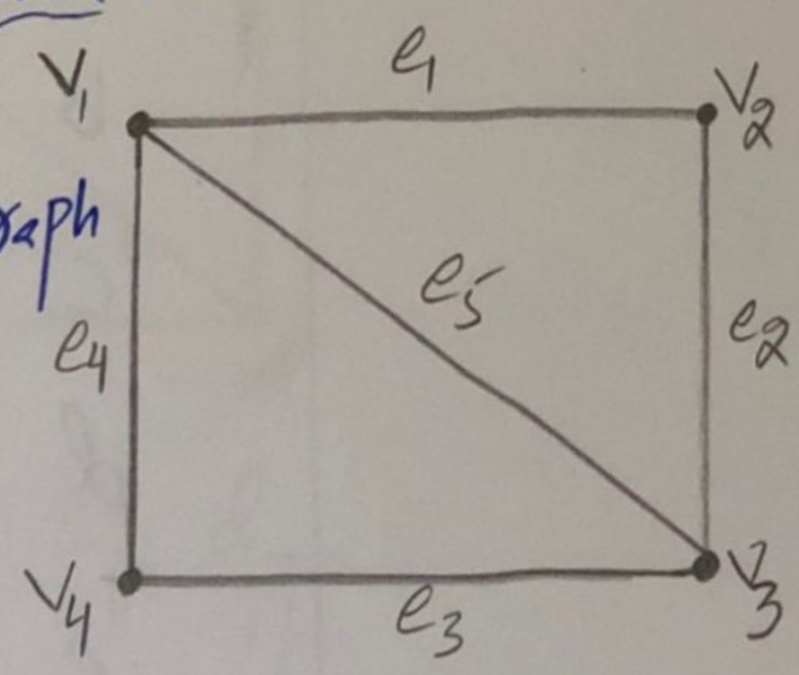
(b) Directed graph



$$A = \begin{array}{c|cccc} & v_1 & v_2 & v_3 & v_4 \\ \hline v_1 & 0 & 1 & 1 & 0 \\ v_2 & 0 & 0 & 1 & 0 \\ v_3 & 0 & 0 & 0 & 1 \\ v_4 & 1 & 0 & 0 & 0 \end{array}$$

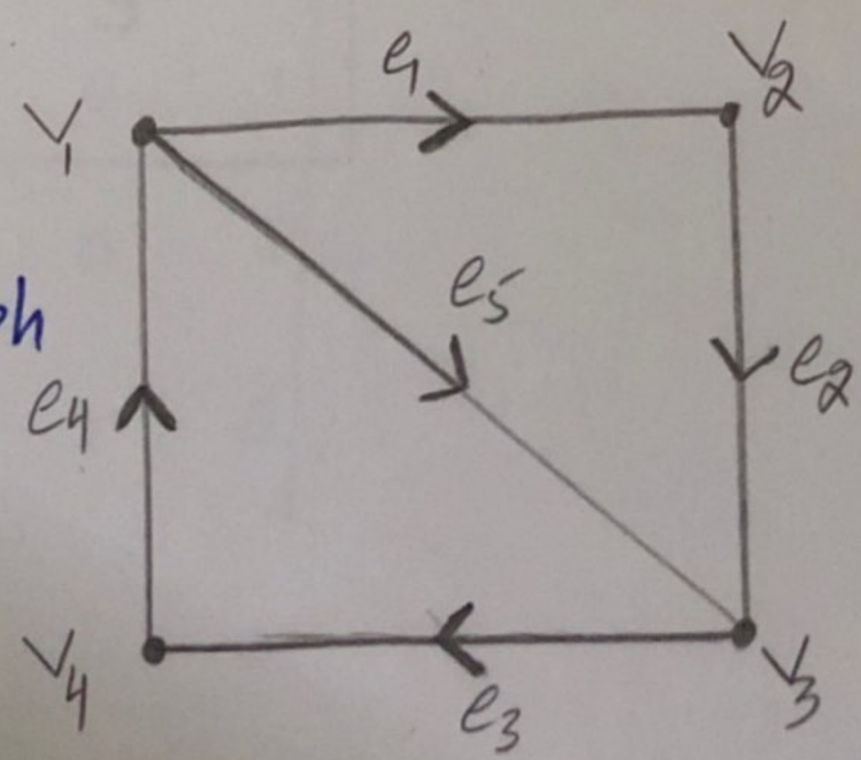
(2) Incidence Matrix:

(a) Undirected graph



$$\begin{array}{c|ccccc} & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline v_1 & 1 & 0 & 0 & 1 & 1 \\ v_2 & 1 & 1 & 0 & 0 & 0 \\ v_3 & 0 & 1 & 1 & 0 & 1 \\ v_4 & 0 & 0 & 1 & 1 & 0 \end{array}$$

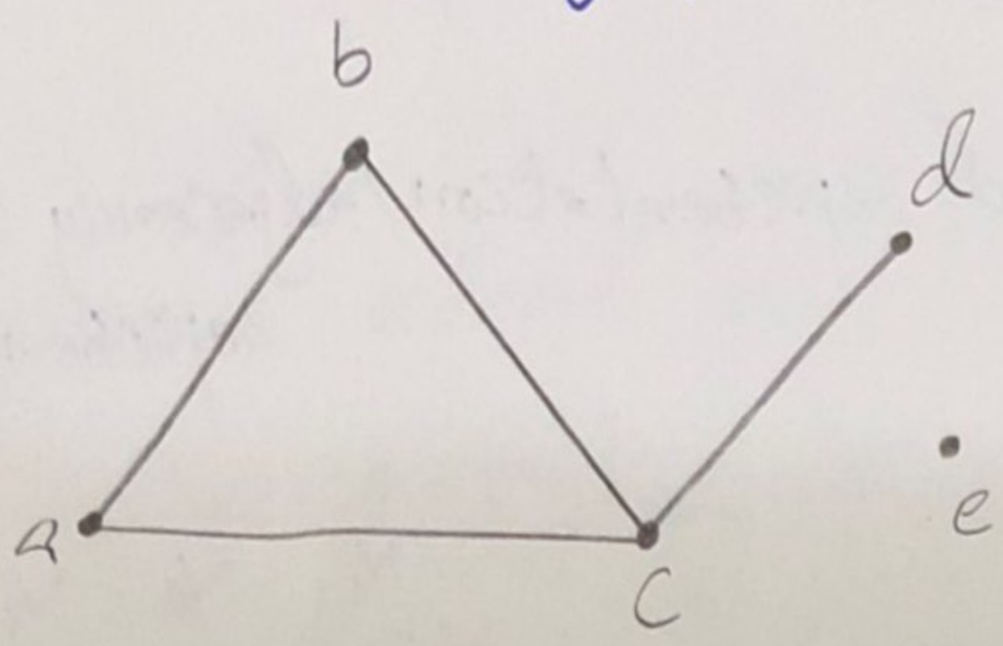
(b) Directed graph



$$\begin{array}{c|ccccc} & e_1 & e_2 & e_3 & e_4 & e_5 \\ \hline v_1 & 1 & 0 & 0 & -1 & 1 \\ v_2 & -1 & 1 & 0 & 0 & 0 \\ v_3 & 0 & -1 & 1 & 0 & -1 \\ v_4 & 0 & 0 & -1 & 1 & 0 \end{array}$$

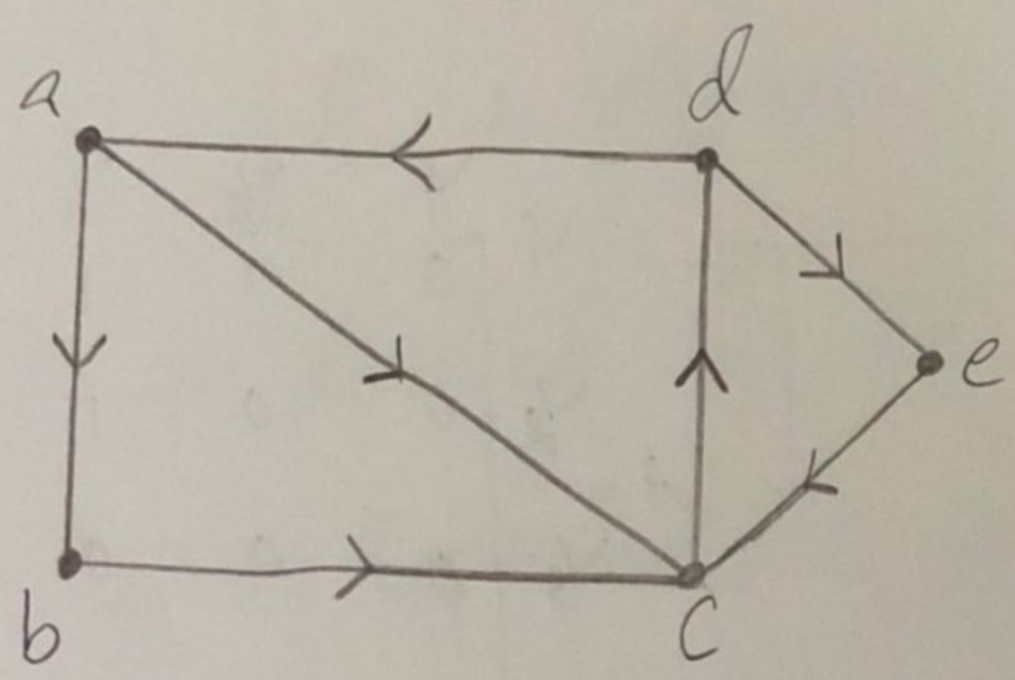
(3) Adjacency List :

(a) Undirected graph



Vertex	Adjacency List
a	b, c
b	a, c
c	a, b, d
d	c
e	ϕ

(b) Directed graph



Vertex	Adjacency List
a	b, c
b	c
c	d
d	a, e
e	c

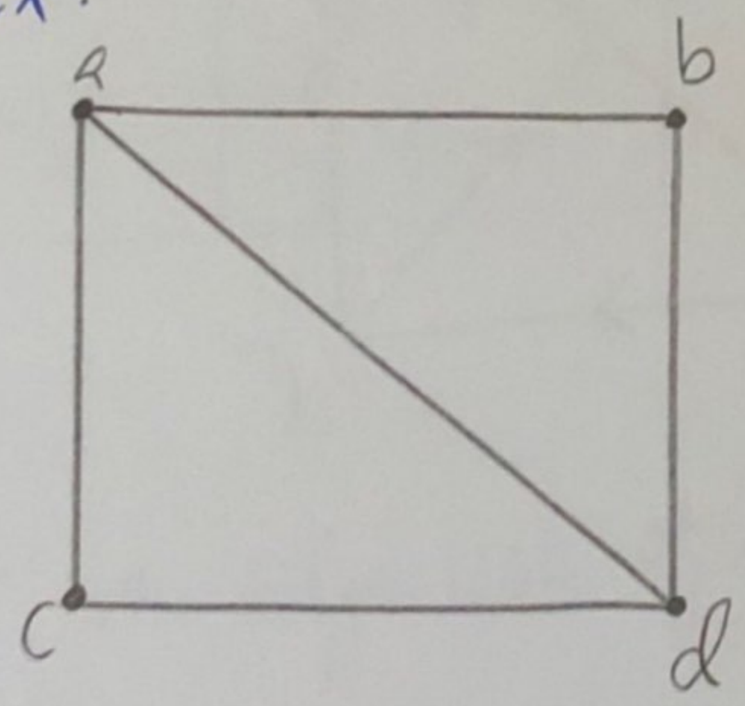
* Matrix Representation of Graphs and Digraphs :

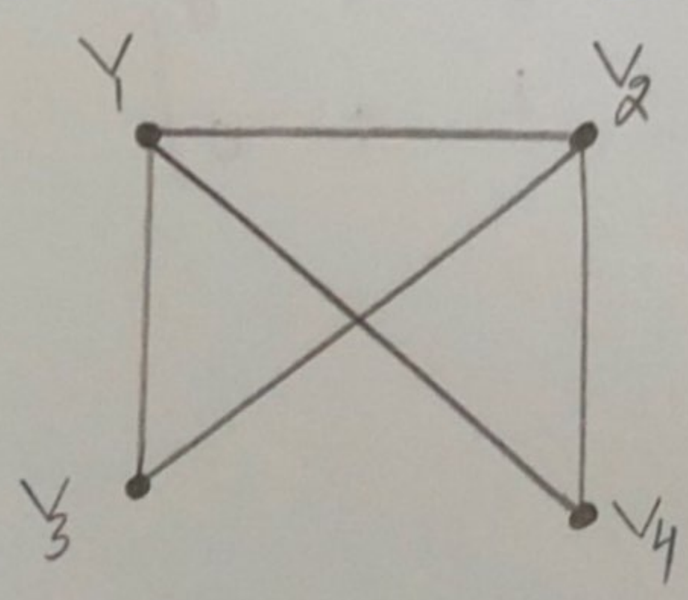
⇒ **Adjacency Matrix** : Let $G(V,E)$ be a simple graph with n vertices ordered from v_1 to v_n, then the adjacency matrix $A = [a_{ij}]_{n\times n}$ of G is an $n\times n$ symmetric matrix defined by the elements.

$$a_{ij} = \begin{cases} 1 & ; \text{ When } v_i \text{ is adjacent to } v_j \\ 0 & ; \text{ otherwise} \end{cases}$$

It is denoted by $A(G)$ or A_G.

Example: Represent the following graph with an adjacency matrix.





Sol :

$$A(G) = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

$$A_G = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

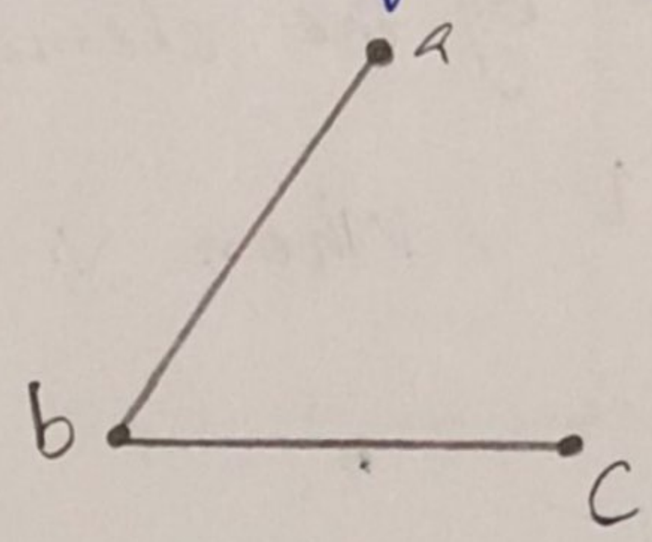
Scanned with CamScanner

Example: Draw a graph of the given adjacency matrix.

(a) $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ (b) $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

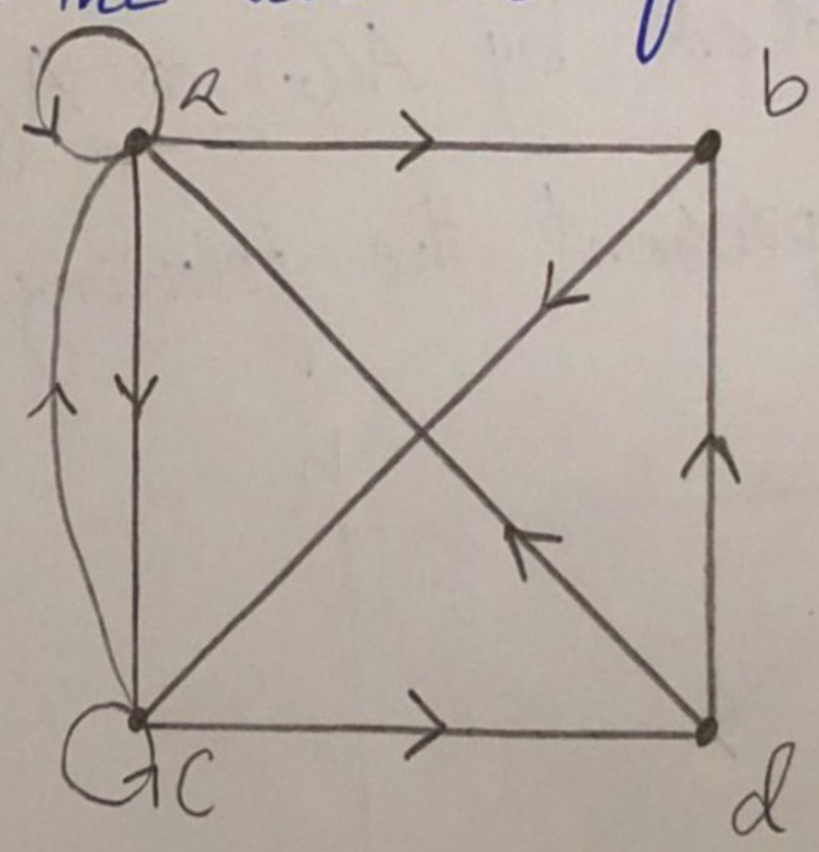
Sol: (a) Let a, b, c are the vertices of the given graph.

$$\begin{array}{c c} & \begin{array}{ccc} a & b & c \end{array} \\ \begin{array}{c} a \\ b \\ c \end{array} & \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array}$$



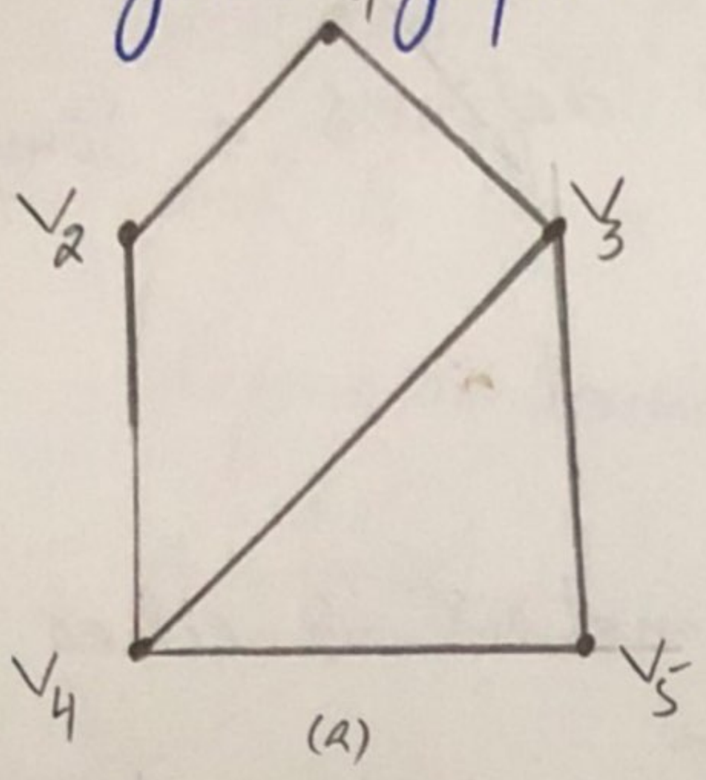
(b) Let a, b, c and d are the vertices of the given graph.

$$\begin{array}{c c} & \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{array}{c} a \\ b \\ c \\ d \end{array} & \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

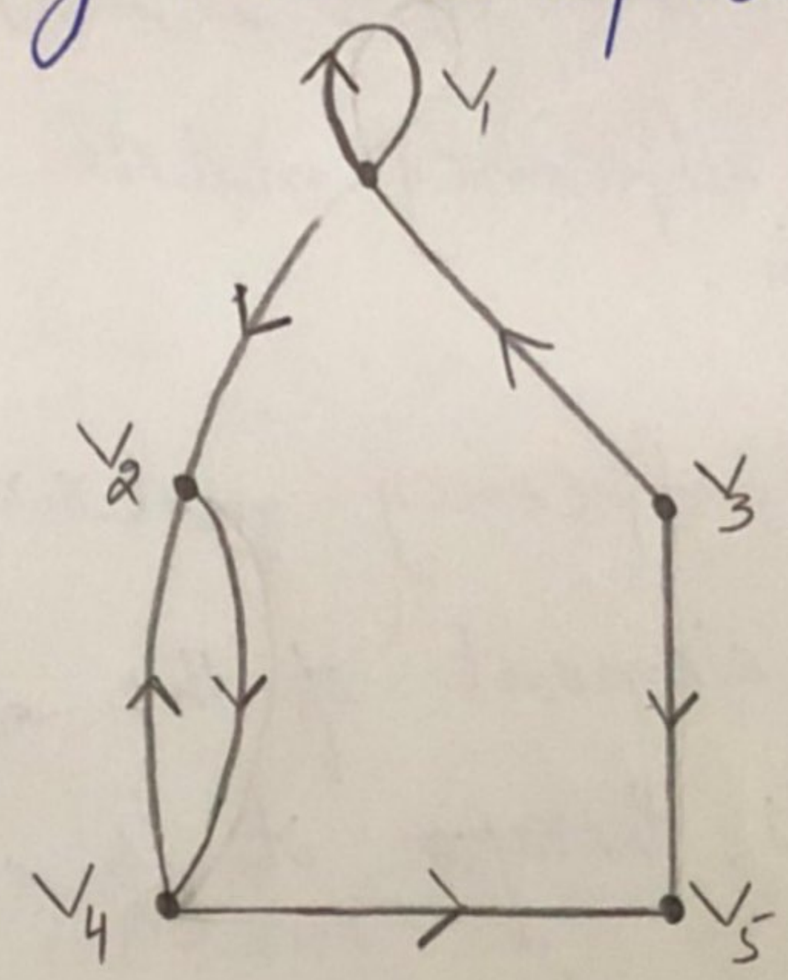


Scanned with CamScanner

Example: Use an adjacency list to represent the given graph.



(a)



Sol: An Edge List of (a)

Vertex	Adjacency vertices
v_1	v_2, v_3
v_2	v_1, v_4
v_3	v_1, v_4, v_5
v_4	v_2, v_3, v_5
v_5	v_3, v_4

Sol: An Edge List of (b)

Vertex	Adjacency vertices
v_1	
v_2	
v_3	
v_4	
v_5	

Scanned with CamScanner

→ Properties of adjacency matrix:

1. An adjacency matrix Completely defines a Simple graph.

2. The adjacency matrix is Symmetric.

3. Any element of the adjacency matrix is either 0 or 1, therefore it is also Called as, bit matrix or boolean matrix.

4. The ith row in the adjacency matrix is determined by the edges which originate in the node V_i.

5. If the graph G is Simple, the degree of the Vertex V, equals the number of 1's in the ith row (or ith Column) of A_G.

6. Given an $n \times n$ Symmetric boolean matrix A, we Can find a Simple graph G Such that A is the adjacency matrix of G.

7. G is null $\longleftrightarrow$ A(G) is the zero matrix of order n.

⇒ **Incidence Matrix**: Let G be a graph with n vertices and V be a set of vertices $\{v_1, v_2, \ldots, v_n\}$ and E be a set of edges $\{e_1, e_2, \ldots, e_n\}$ that will define a matrix $n \times m$.

$A = [m_{ij}]_{n \times m}$ where

$$m_{ij} = \begin{cases} 1 & ; \text{when } v_i \text{ is incident with } e_j \\ 0 & ; \text{otherwise} \end{cases}$$

It is denoted by I_G.

Example: Find the incidence matrix for the following graph

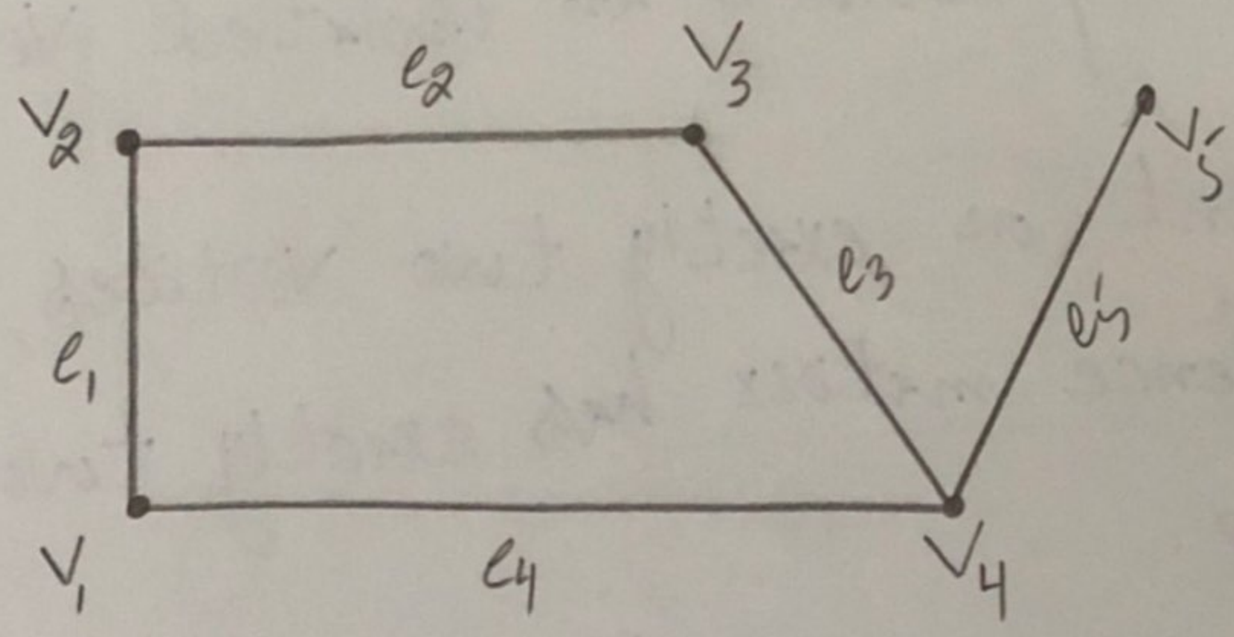


Fig: (1)

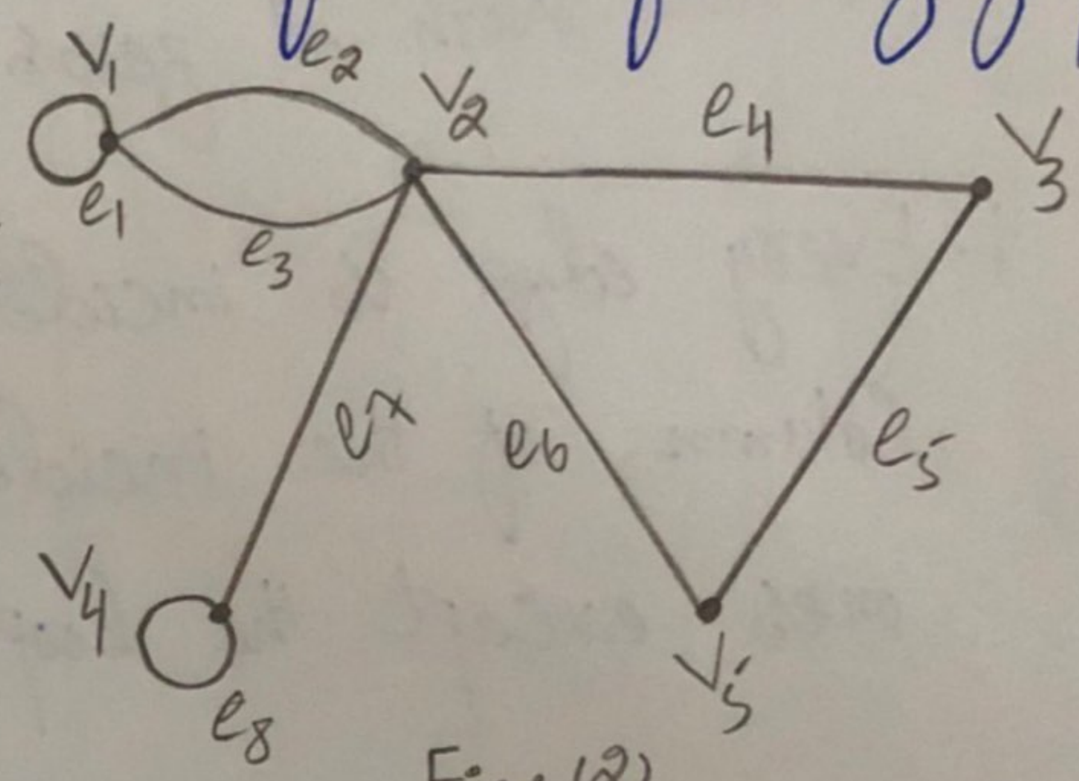


Fig: (2)

Sol: Fig: (1)

The incidence matrix for the given graph is

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Sol: Fig: (2)

The incidence matrix for the graph is

$$
\begin{array}{c|cccccccc}
 & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 \\
\hline
v_1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
v_2 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
v_3 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
v_4 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
v_5 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0
\end{array}
$$

→ Properties of incidence matrix:

1. The incidence matrix Contains only 0 and 1.

2. The number of 1's in each row equals to the degree of the corresponding vertex.

3. A row with all zeros represents an isolated vertex

4. Every edge is incident on exactly two vertices, each Column of the incidence matrix has exactly two ones except the loop.

5. The parallel edges in a graph produce indential Columns in its incidence matrix.

6. Permutation of any two rows or Columns in an incidence matrix Corresponds to relabeling the vertic and the edges of the Same graph.

* Walk, Trial, Path and Circuit in a Graph:

⟹ **Walk**: A walk in a graph G is a finite alternating sequence of vertices and edges.

$$W = v_0 e_1 v_2 e_2 - - - - - - - v_{n-1} e_n v_n$$

beginning and ending with vertices such that each edge e_i in the sequence is incident with vertices v_{i-1} and v_i. we say that the above walk "W" is a walk from v_0 to v_n. In any walk initial vertex and end vertex are called "terminal vertex".

Example:

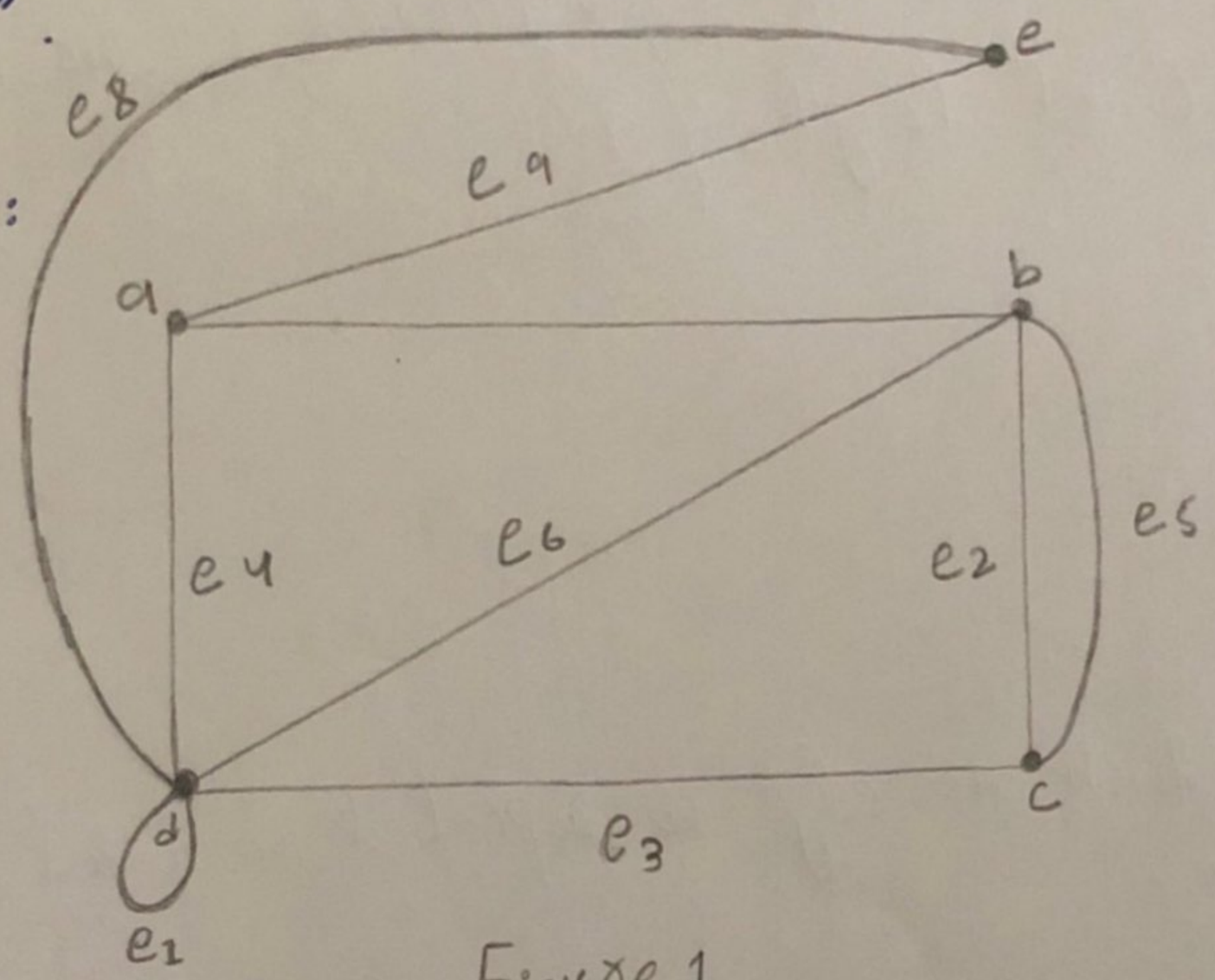


Figure 1

Sol:

$W_1 = a e_4 b e_2 c e_3 d e_8 e$

$W_2 = a e_4 e e_8 d e_7 d e_3 c e_5 b e_6 d$

$W_3 = d e_7 d e_8 e e_9 a e_4 d e_6 b e_2 c e_5 b$

Note: In a walk vertex and edges may be repeated number of times.

→ **Open Walk**: A walk in which first and last vertex are different is called "open walk".

→ **Closed Walk**: A walk in which first and last vertex are same is called "Closed walk".

In Figure (1) open walk are:

In Figure (1) Closed walk are:

$W = ae_1be_2ce_3de_8ee_9a$

$W = de_7de_8ee_9ae_4de_6be_2ce_5be_6d$

→ **Length of Walk**: The number of edges occurred in the walk is called length of walk.

Length of $W_4 = 5$

Length of $W_5 = 8$

⇒ **Trial**: A walk in which no edge is repeated is called a tr

⇒ **Closed Trial (Tour)**: A trial whose first and last vertex is same is called a Closed trial or Tour.

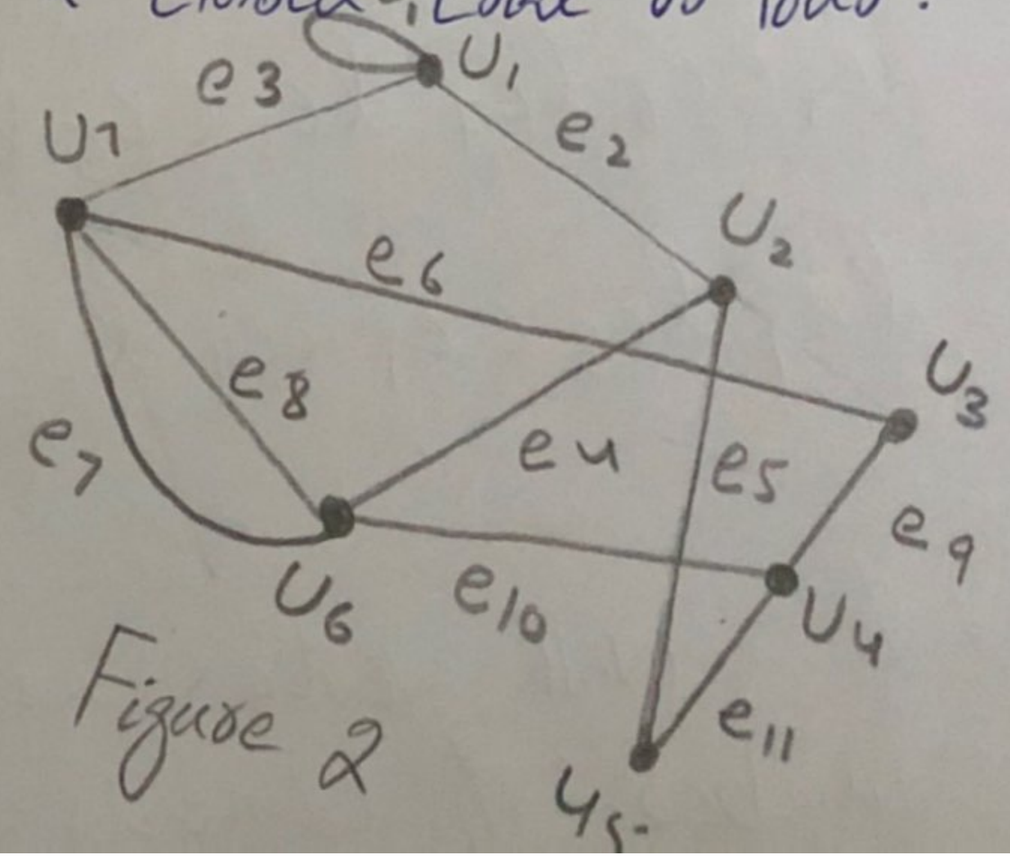


Figure 2

Sol:

$T_1 : u_1 e_1 u_2 e_5 u_5 e_{11} u_4 e_9 u_3 e_6 u_7$

$T_2 : u_6 e_{10} u_4 e_{11} u_5 e_5 u_2 e_4 u_6 e_7 u_7$

$T_3 : u_7 e_8 u_6 e_{10} u_4 e_9 u_3 e_6 u_7 e_3 u_1$

$T_4 : u_1 e_1 u_2 e_2 u_5 e_{11} u_4 e_9 u_3 e_6 u_7 e_3 u_1$

$T_5 : u_2 e_5 u_5 e_{11} u_4 e_{10} u_6 e_7 u_7 e_3 u_1 e_2 u_2$

In above five walks, T_1, T_2, T_3 are "trails" and T_4, T_5 are "closed trails" or tour.

Note: In trial edge is not repeated but vertex may be repeated.

Scanned with CamScanner

⇒ **Path** : A walk is said to be path, in which no vertex is repeated.

⇒ **Cycle** or **Circuit** : A closed walk in which no vertex is repeated except terminal vertices is called a cycle or circuit.

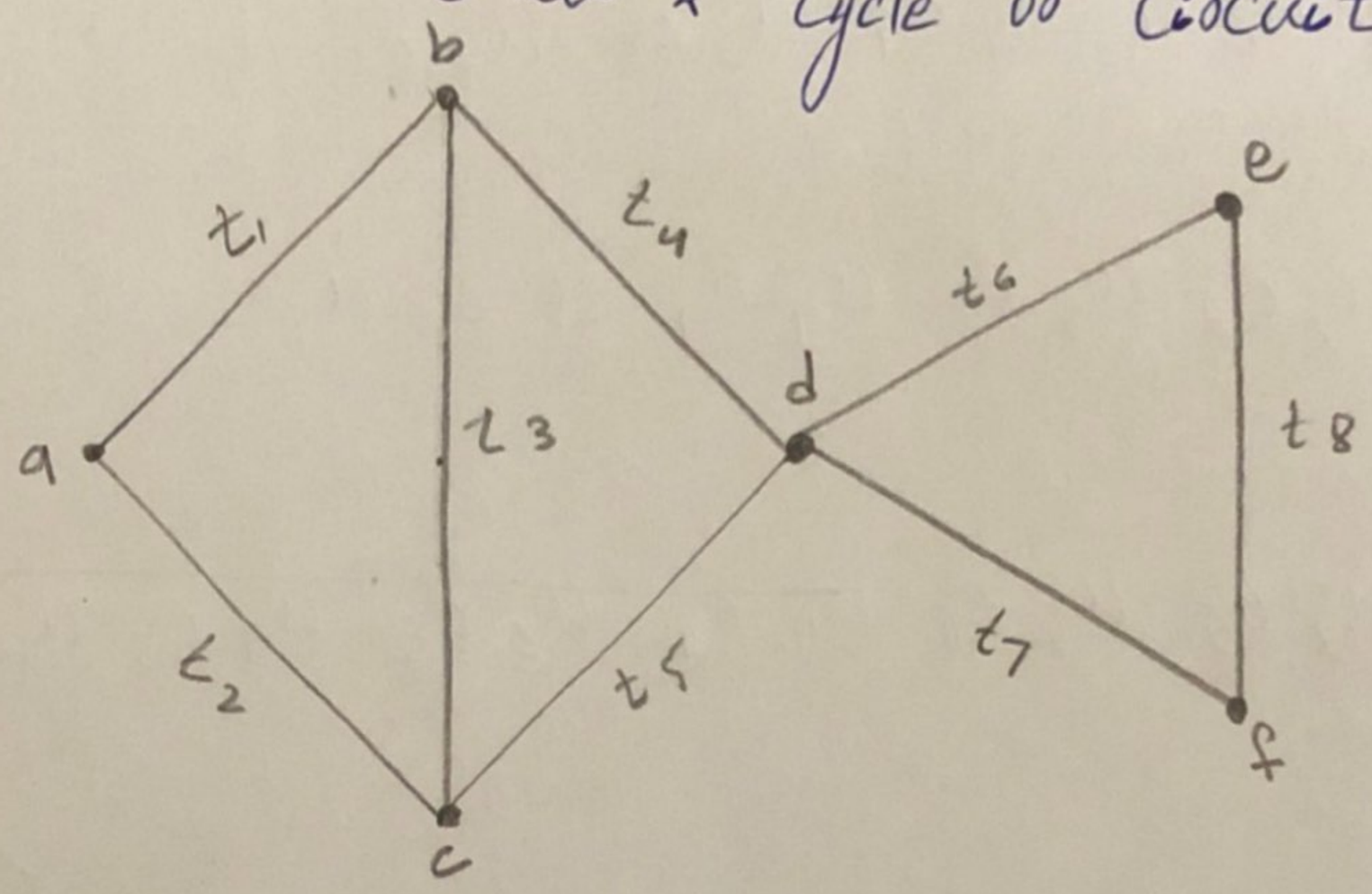


Figure 3

Sol:

$P_1 = a t_1 b t_3 c t_5 d t_6 e t_8 f$

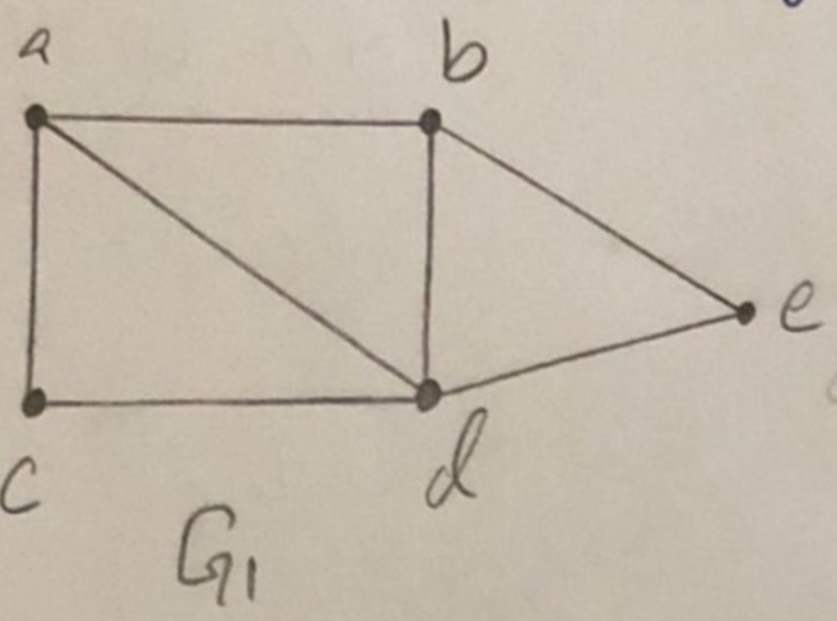
$P_2 = a t_2 c t_5 d t_6 e t_8 f$; $P_3 = c t_3 b t_4 d t_7 f$

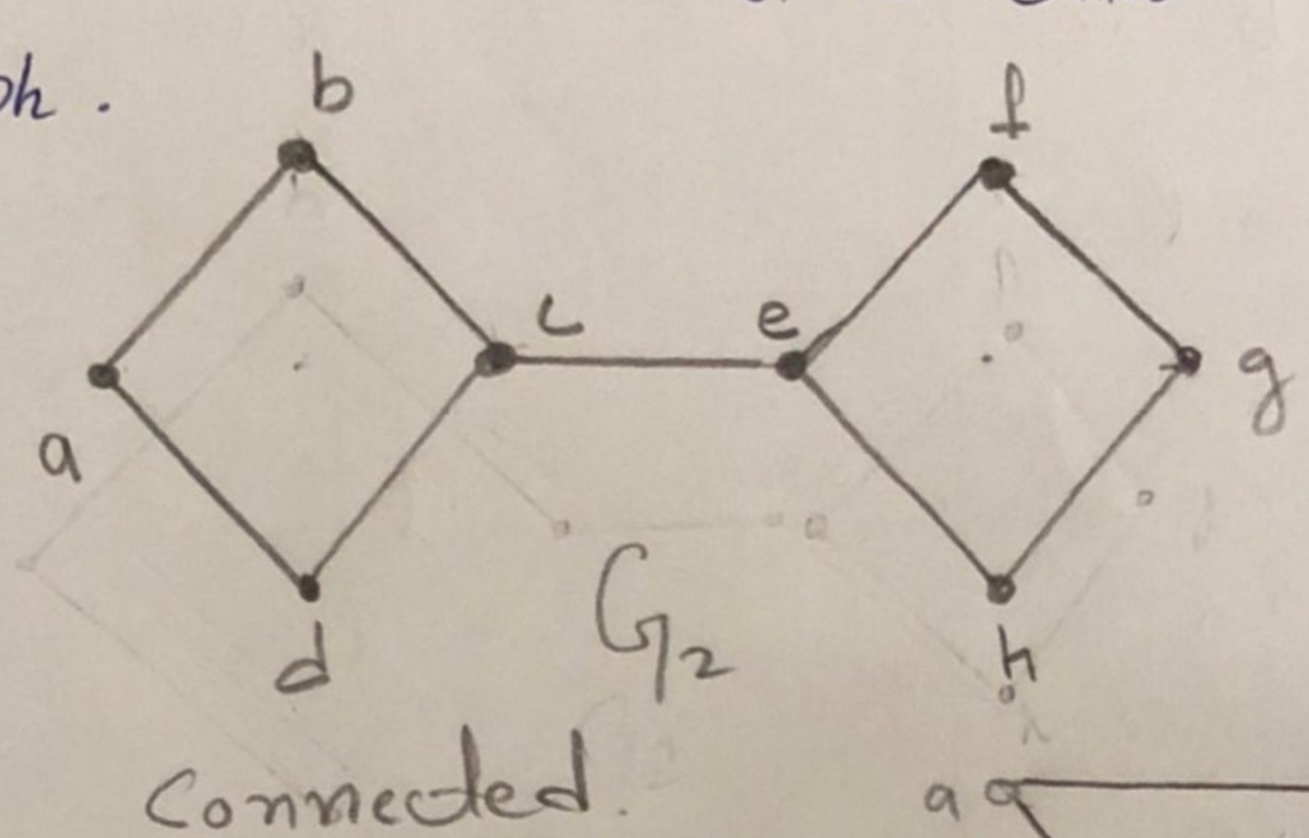
$P_4 = a t_2 c t_5 d t_4 b t_1 a$; $P_5 = e t_8 f t_7 d t_6 e$

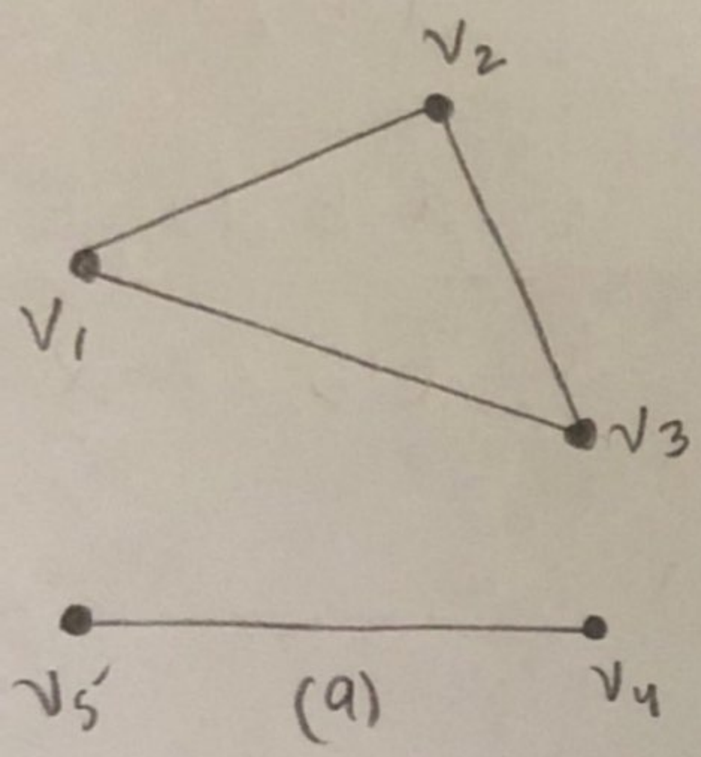
In above all walks, P_1, P_2 and P_3 are "paths" and P_4, P_5 are "circu

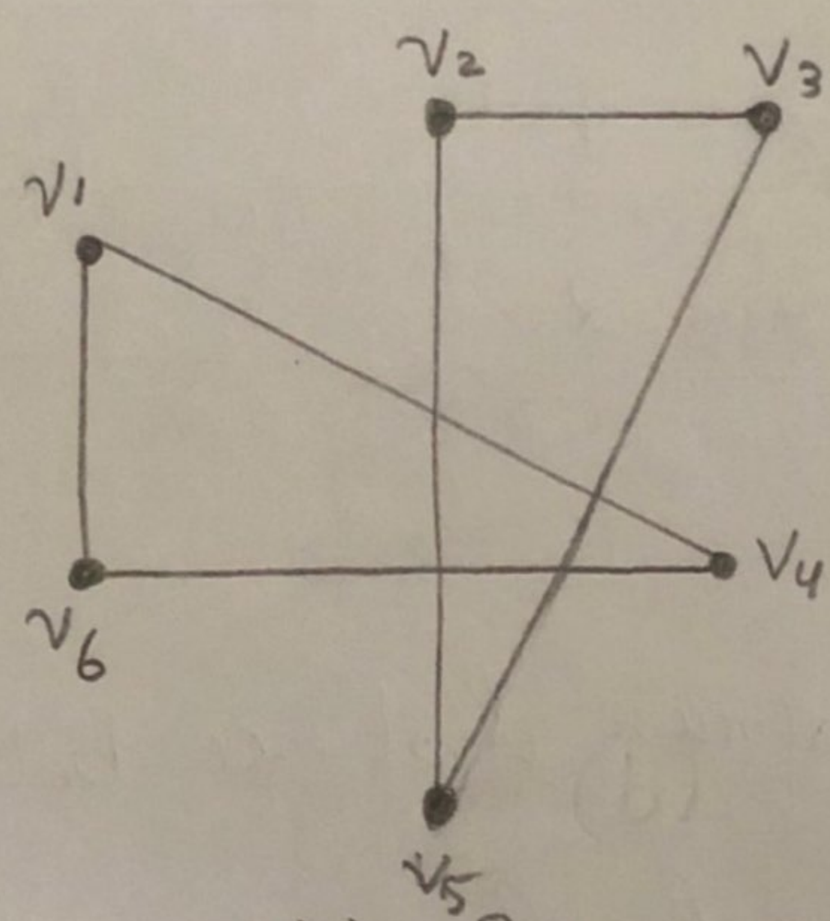
⇒ **Connected vertices** : A vertex u is said to be connected to a vertex v in a graph G if there is a path in G fr vertex u to vertex v.

⇒ **Connected Graph**: A graph G is said to be connected graph if there is at least one path between every pair of vertices in G otherwise G is called disconnected graph.

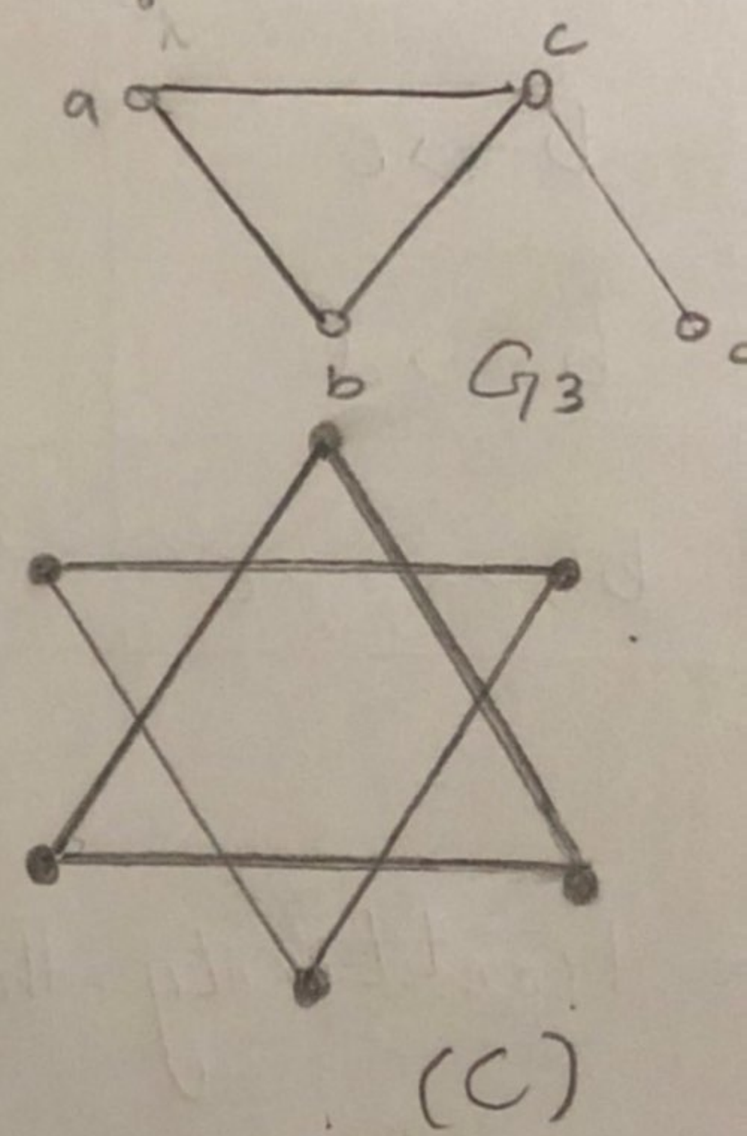




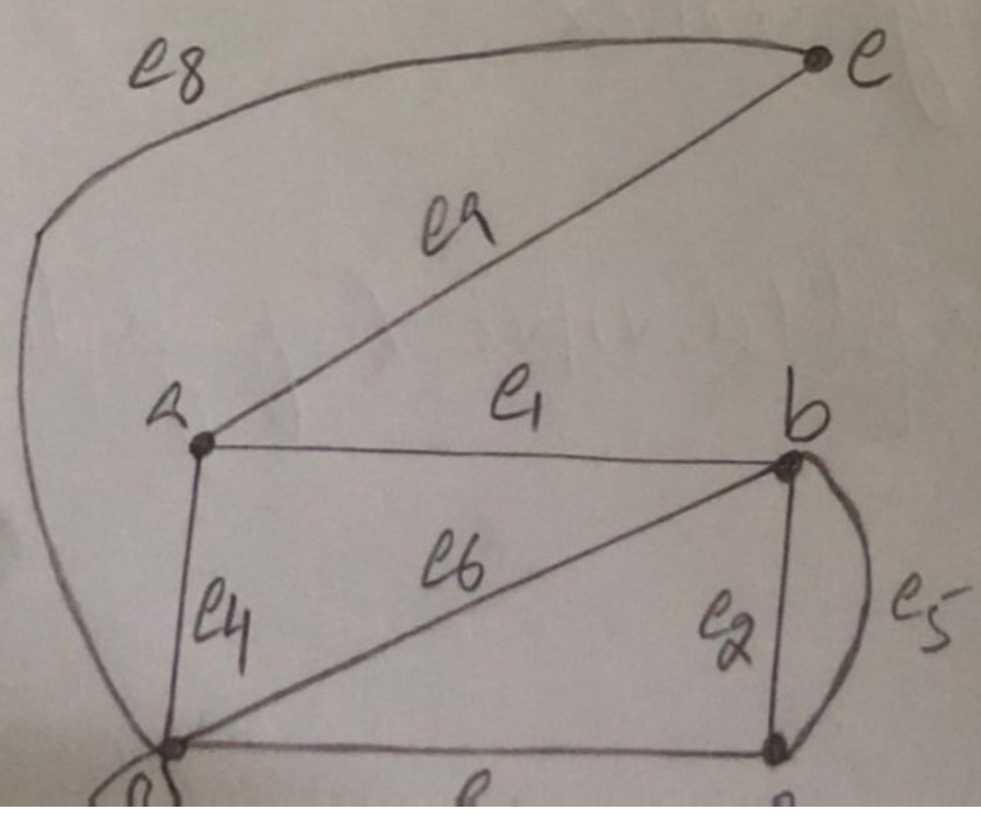




(b) Disconnected graph



⇒ **Distance between two vertices**: Let G be a connected graph and v be any two vertices of graph G, then the smallest path between u and v and it is denoted by $d(u, v)$.

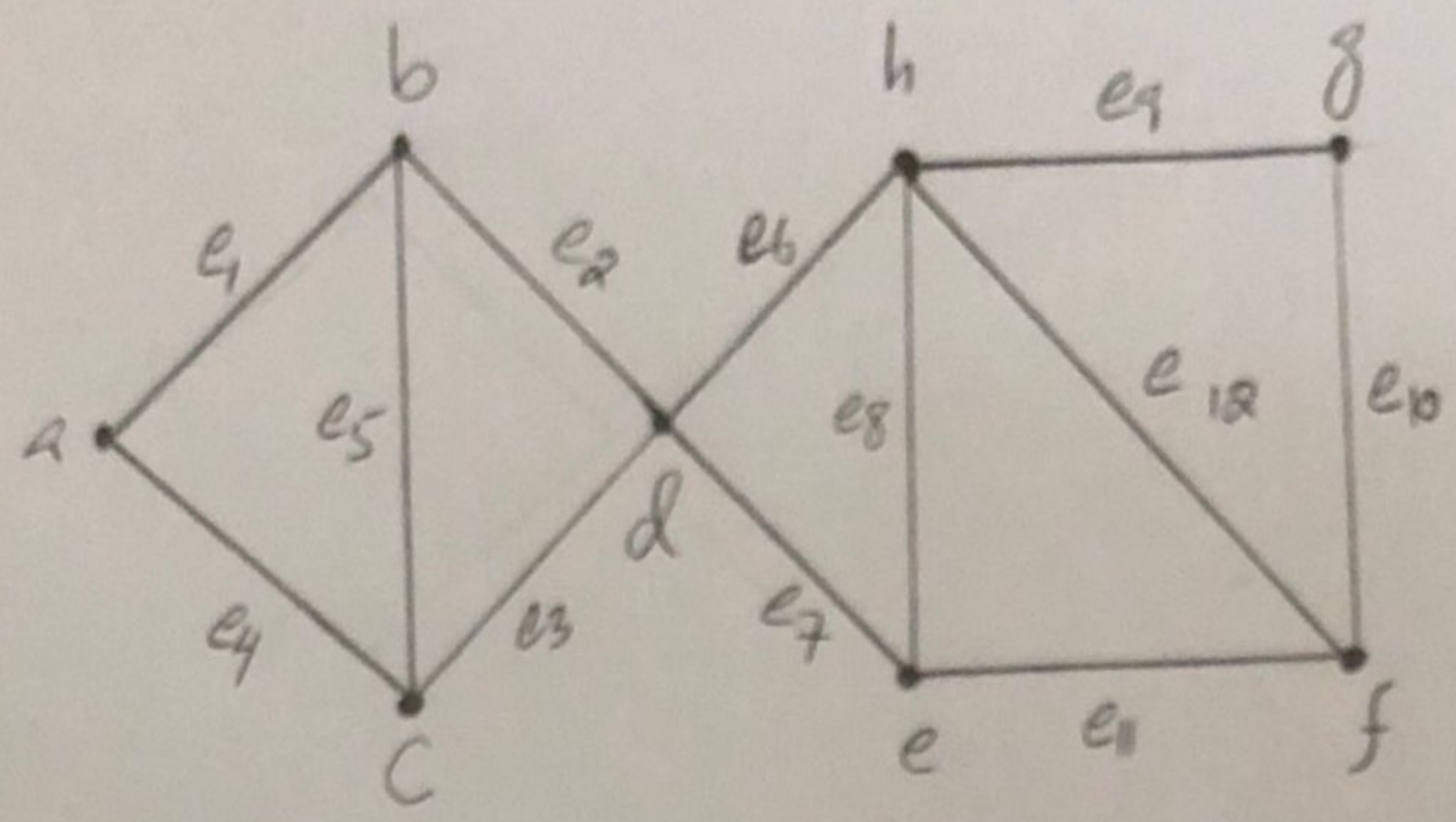


Smallest path	Number of edges occurred	Distance
$a \longrightarrow b$	1	$d(a,b)=1$
$a \rightarrow b \rightarrow c$	2	$d(a,c)=2$
$a \longrightarrow d$	1	$d(a,d)=1$
$a \longrightarrow e$	1	$d(a,e)=1$
$b \rightarrow a$	1	$d(b,a)=1$
$b \rightarrow c$	1	$d(b,c)=1$
$b \rightarrow d$	1	$d(b,d)=1$
$b \rightarrow a \rightarrow e$	2	$d(b,e)=2$

⟹ Eccentricity: The maximum distance between a vertex to all other vertices is considered as the eccentricity of vertex. Eccentricity of vertex u is denoted by $e(u)$.

i.e. the eccentricity of the vertex u is the maximum distance from u to any vertex.

$$e(u) = \max\{d(u,v) \mid v \in V \text{ of graph } G\}$$

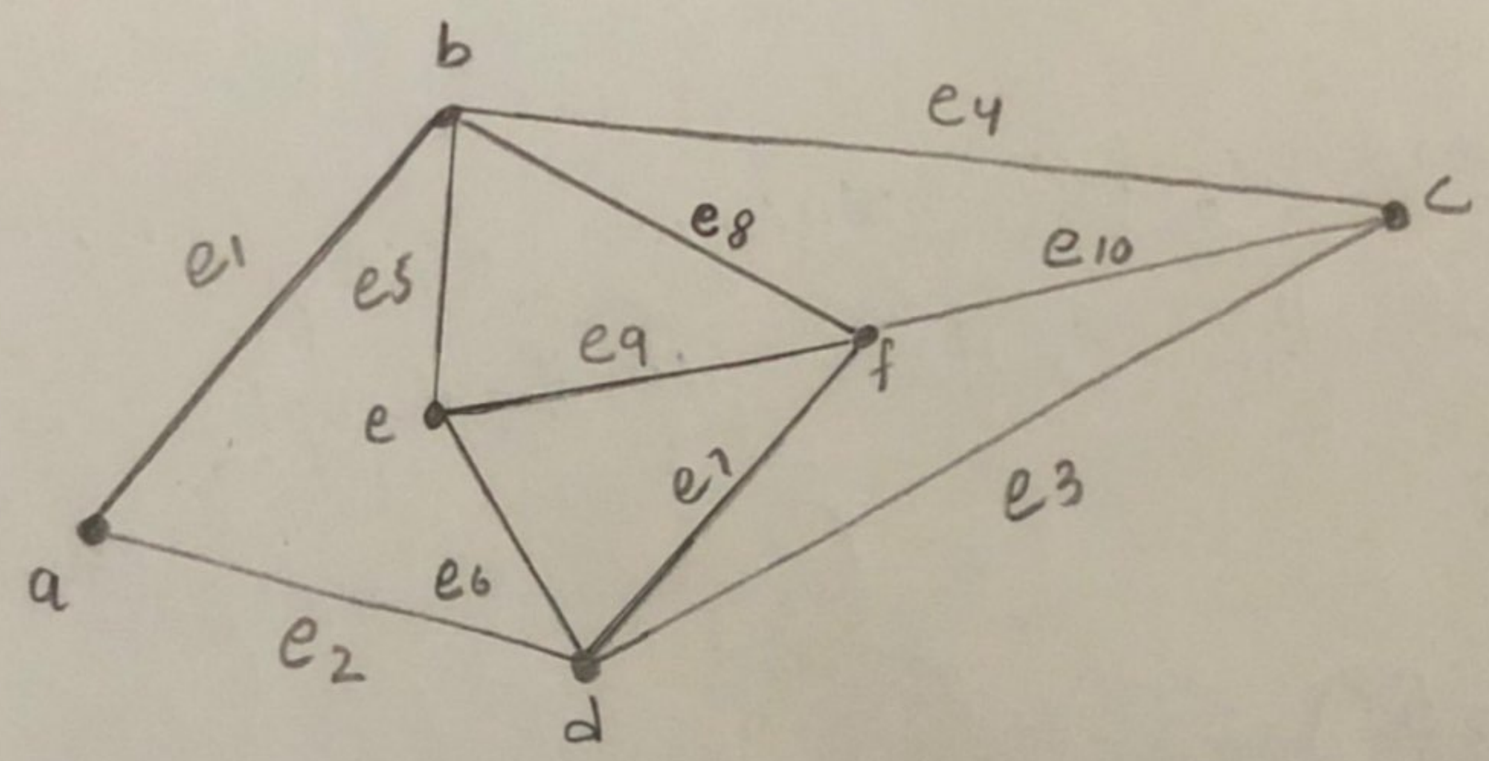

b
h
g
e_9
e_1
e_2
e_6
e_{12}
e_{10}
e_8
a
e_5
d
e_4
e_3
e_7
c
e
e_{11}
f

⇒ Centre: A vertex having minimum eccentricity is called "Centre" of a graph.

⇒ Radius: The minimum eccentricity is called the "radius" of the graph.

⇒ Diameter: Maximum eccentricity of graph is called "diameter" of the graph.

Example: Find the eccentricity, centre and radius of the following graph.



Sol:

(i) Eccentricity:

$e(a) = 2$, $e(b) = 2$, $e(c) = 2$, $e(d) = 2$, $e(e) = 2$, $e(f) = 2$

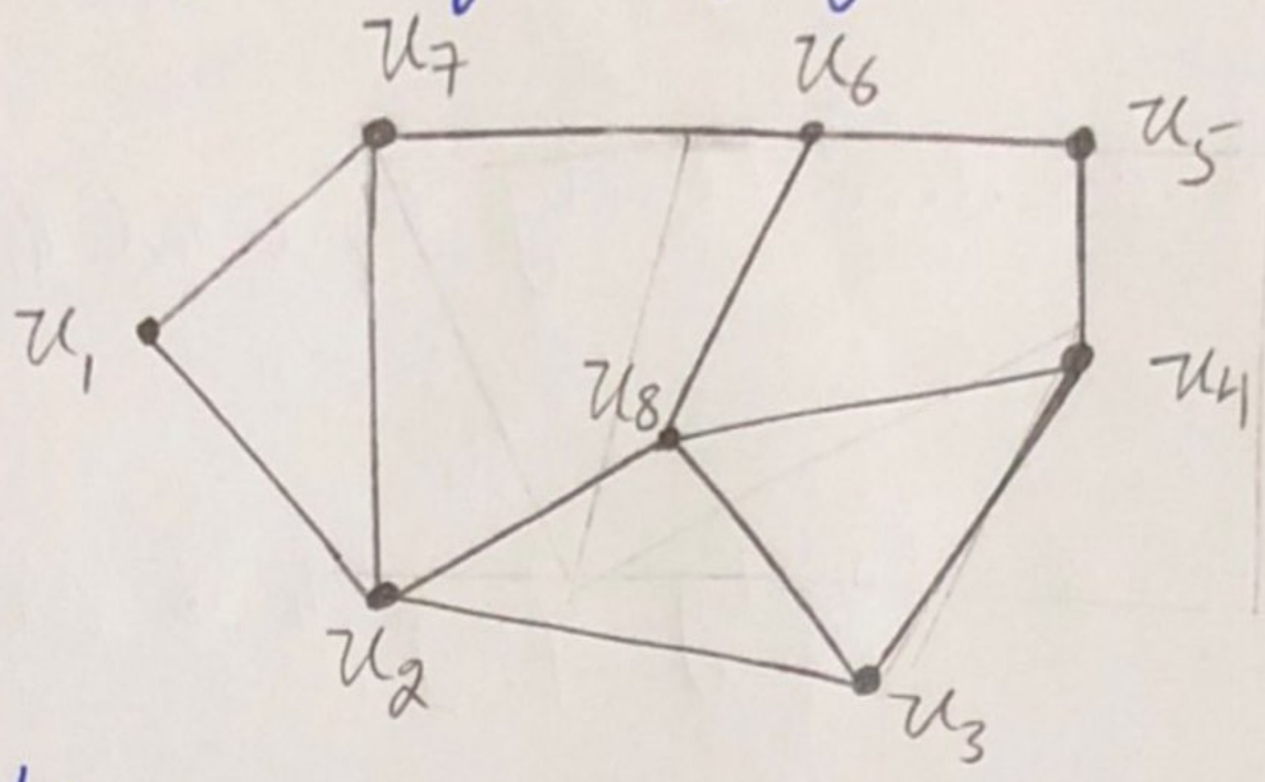
(ii) Centre:

Centre is the vertex which has minimum eccentricity.
i.e; Vertex a, b, c, d, e and f all are centres of the graph

(iii) Radius:

Minimum eccentricity is the radius of the graph i.e, 2.

Example: Find the eccentricity, Centre, radius and diameter of the following graph.



Sol:

(i) Eccentricity:

$e(u_1) = 3$, $e(u_2) = 3$, $e(u_3) = 2$, $e(u_4) = 3$, $e(u_5) = 3$,

$e(u_6) = 2$, $e(u_7) = 3$

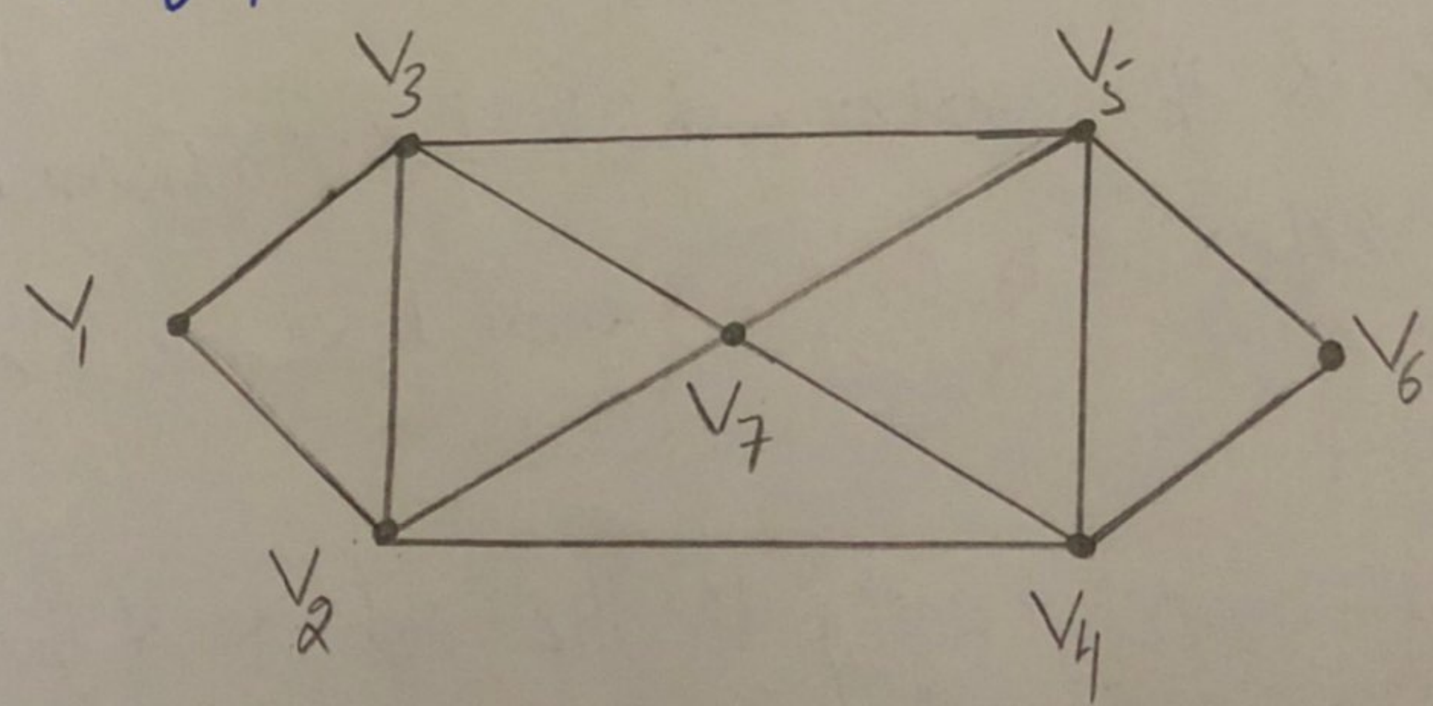
(ii) Centre:

vertex u_3 and u_6 has minimum eccentricity. Hence u_3 and u_6 are the Centres of the graph.

(iii) Radius: 2

(iv) Diameter: 3

Example: Find the eccentricity, Centre, radius and diameter of the following graph.



Sol:

(i) Eccentricity:

$e(V_1) = 3$, $e(V_2) = 2$, $e(V_3) = 2$, $e(V_4) = 2$, $e(V_5) = 2$, $e(V_6) = 3$,

$e(V_7) = 2$

(ii) Centre:

Vertex v_2, v_3, v_4, v_5 and v_7 has minimum eccentricity. Hence v_2, v_3, v_4, v_5 and v_7 are the Centres of the graph.

(iii) Radius:

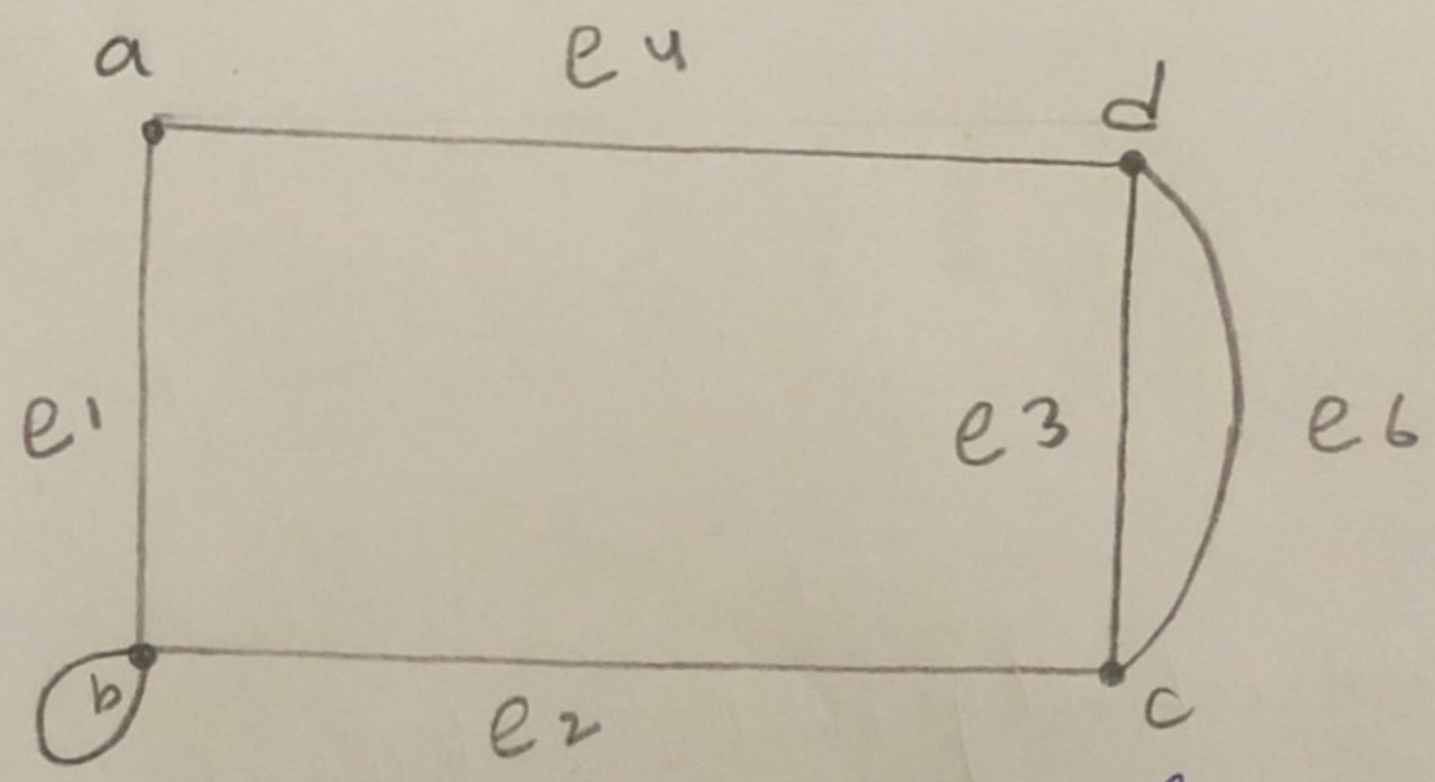
Radius $= 2$

(iv) Diameter:

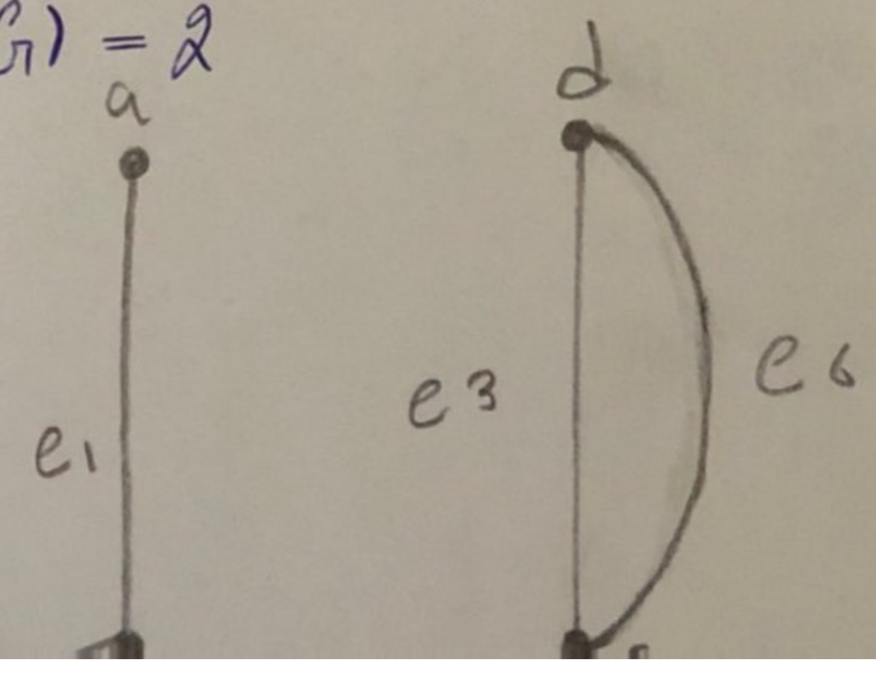
Diameter $= 2$

⟹ Edge Connectivity: The edge Connectivity of the graph G is the minimum number of edges that need to be deleted, Such that the Graph G gets disconnectd and it is denoted by $\lambda(G)$.

Example:

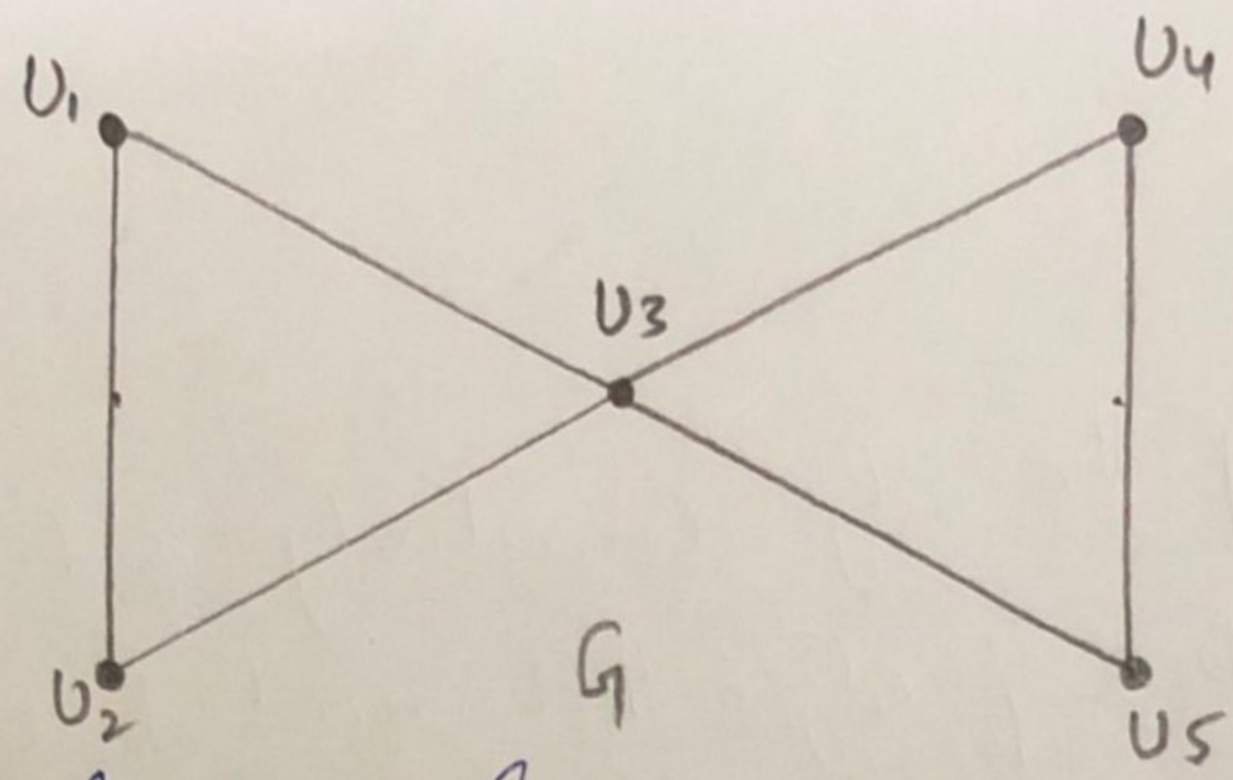


In the above graph G, minimum two edge should be removed then only graph will be disconnected. In G, edge e_2 and e_4 Should be removed then G is a disconnected graph. i.e. $\lambda(G) = 2$

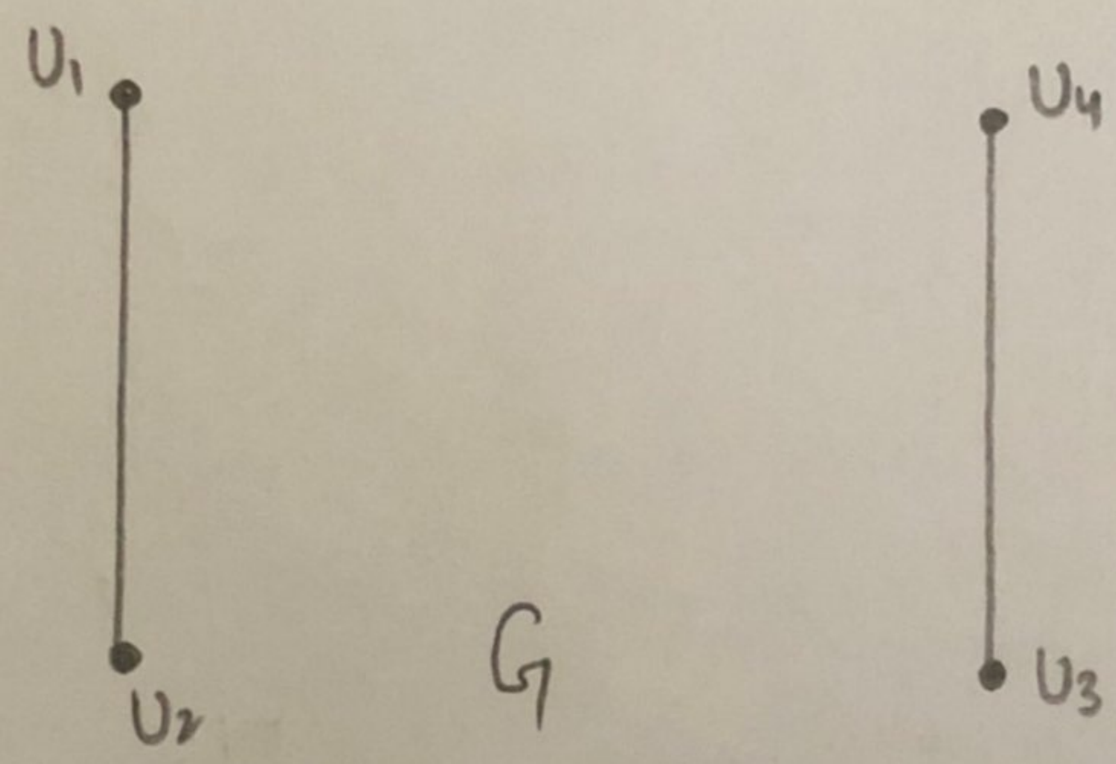


⇒ Vertex Connectivity : The vertex Connectivity of the graph G is the minimum number of vertices that need to be deleted such that the graph G gets disconnected and it is denoted by K(G).

Example :

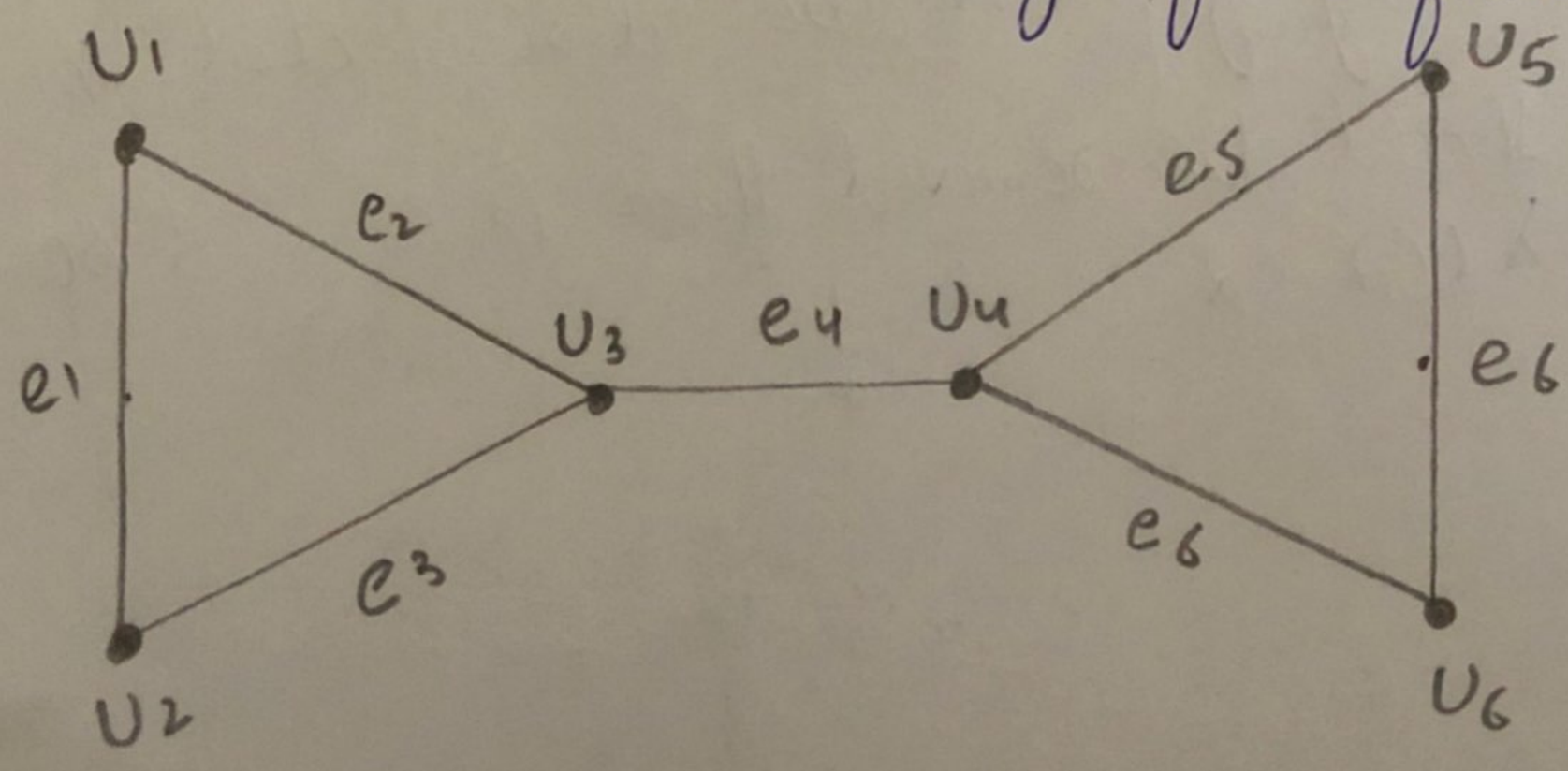


In above graph G, if we delete only one vertex u_3 then the graph is disconnected.



∴ The vertex Connectivity is 1 i.e. K(G) = 1

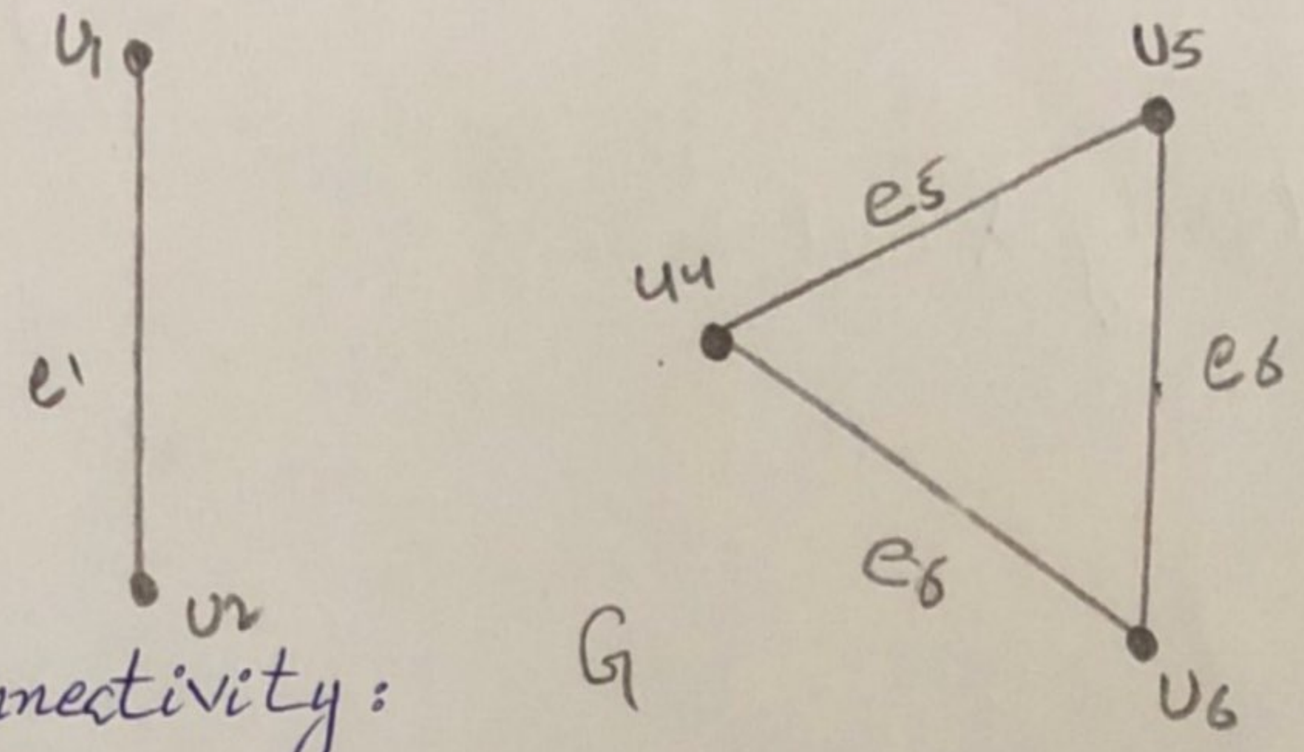
Example : Find the vertex Connectivity and edge connectivity of the following graph.



Sol:

Vertex Connectivity:

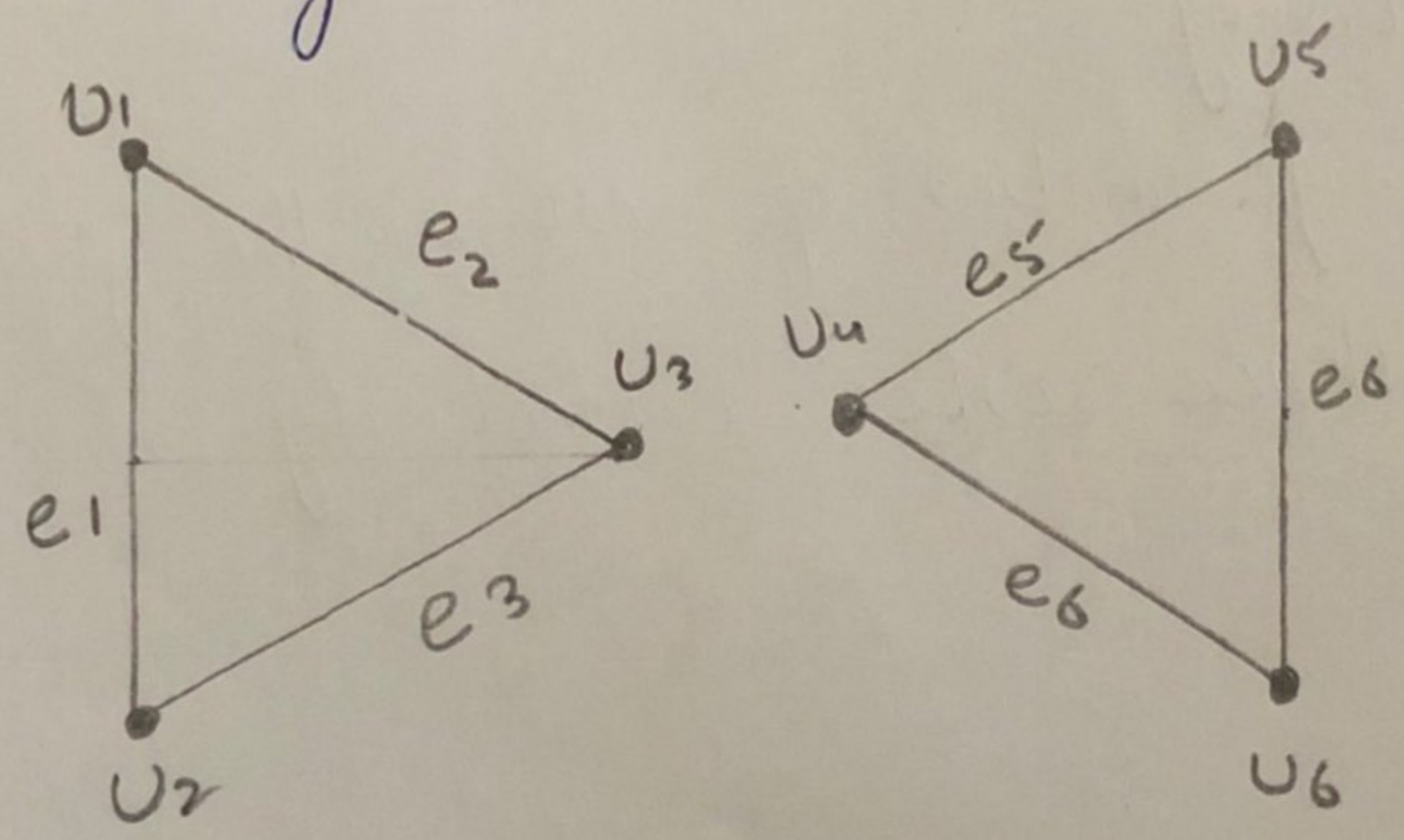
Minimum one vertex v_3 or (v_4) should be removed then graph is disconnected. Hence vertex connectivity $K(G) = 1$.



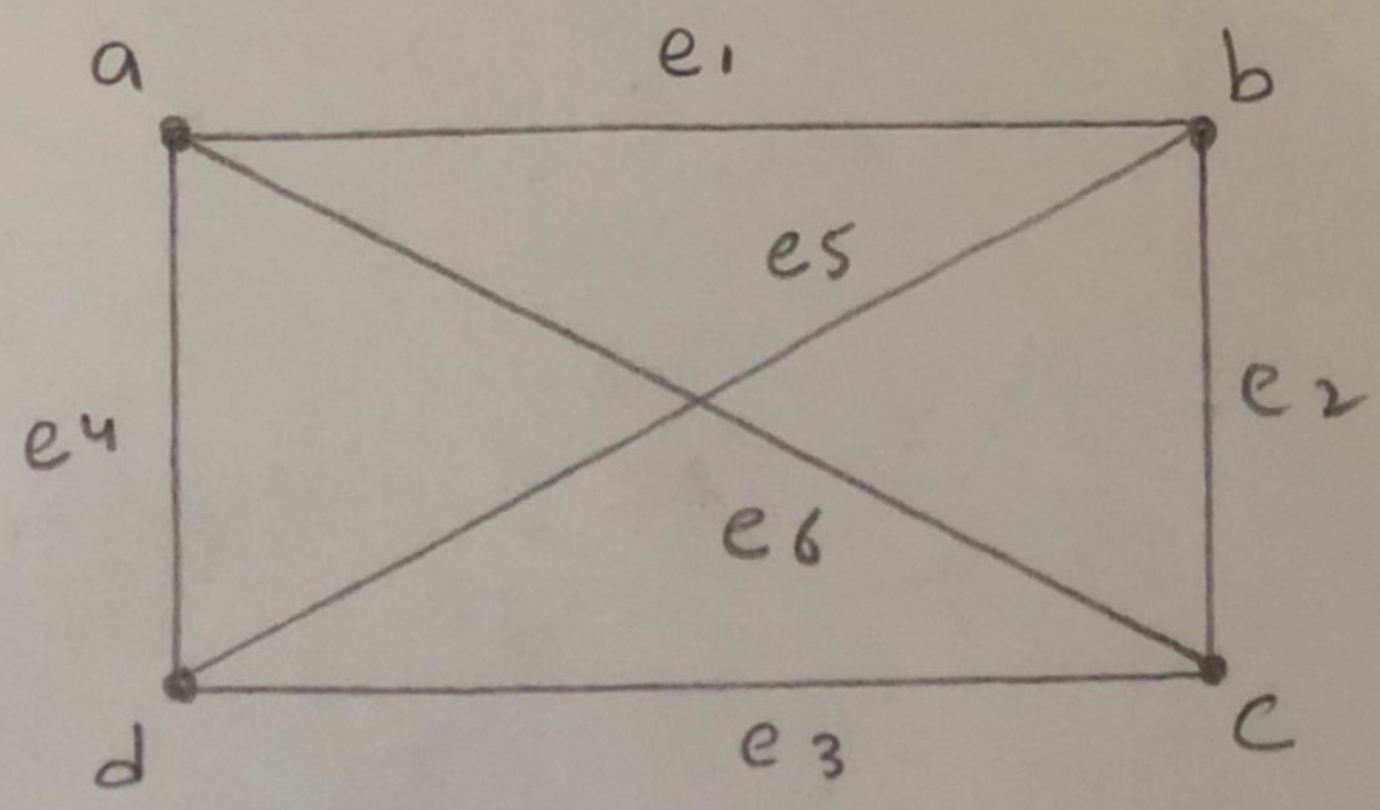
Edge Connectivity:

By deleting only one edge e_4 the graph G is disconnected.

$\therefore$ Edge Connectivity $\lambda(G) = 1$



Example: Find the edge and vertex connectivity of the following graph.

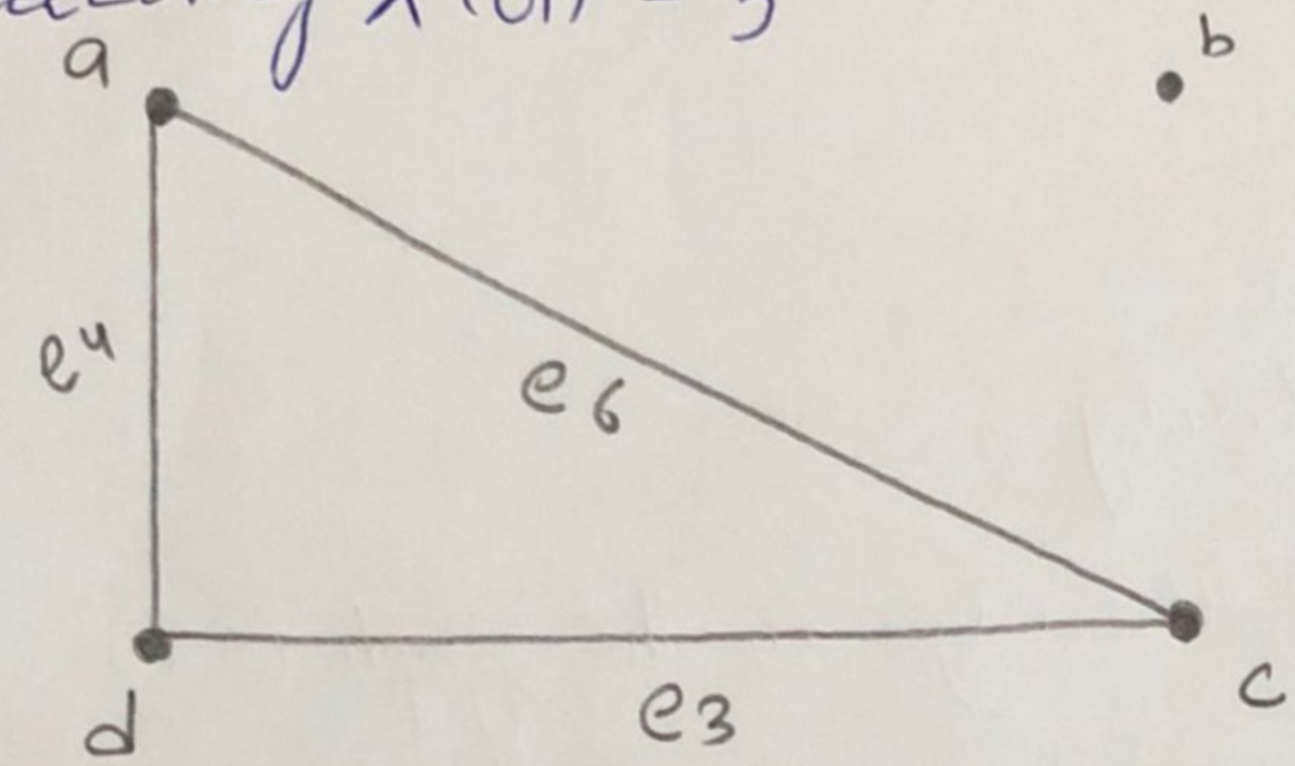


Sol:

Edge Connectivity:

By deleting minimum three edges e_1, e_2 and e_5 or (e_1, e_4, e_6 or e_3, e_4, e_5 or e_2, e_3, e_6) the graph is disconnec ed

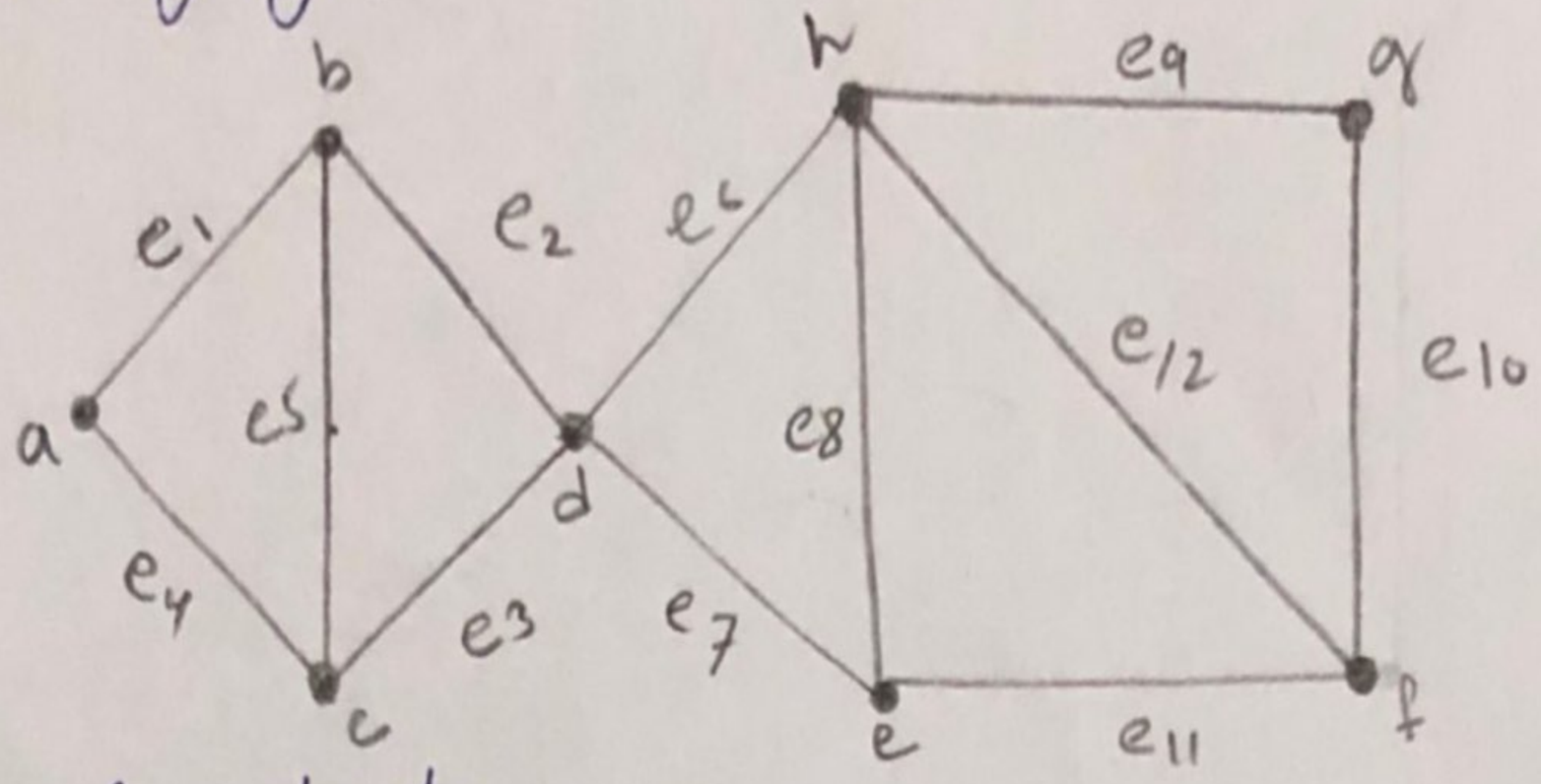
$\therefore$ Edge Connectivity $\lambda(G) = 3$



Vertex Connectivity:

By deleting any number of vertices, the graph is not disconnected. Hence vertex Connectivity is 0, i.e. $K(G) =$

Example: Find the edge and vertex Connectivity of the following graph.

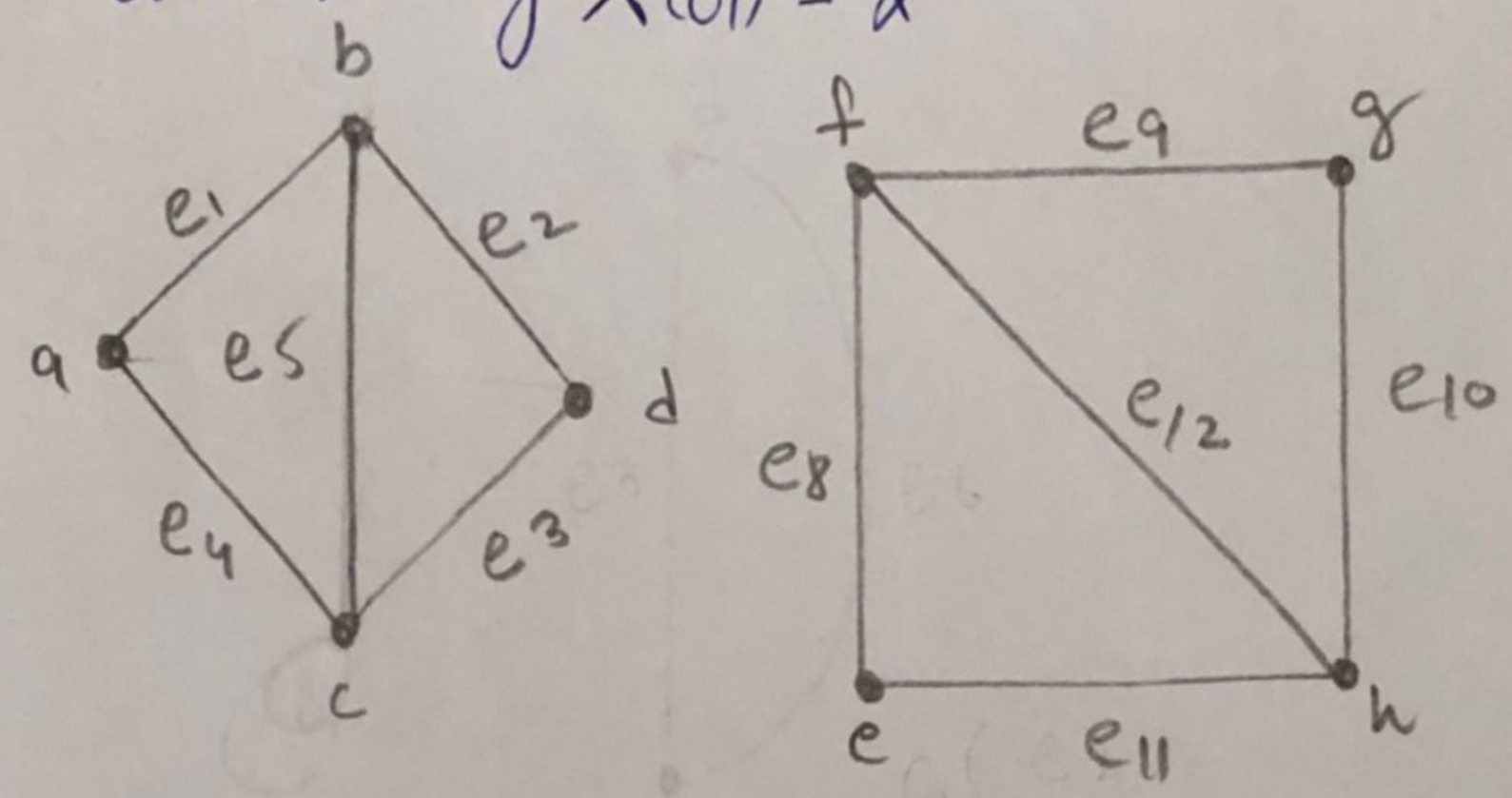


Sol:

Edge Connectivity:

By deleting minimum two edges e_6, e_7 or (e_2, e_3 or e_1, e_4 or e_9, e_{10}) the graph is disconnected.

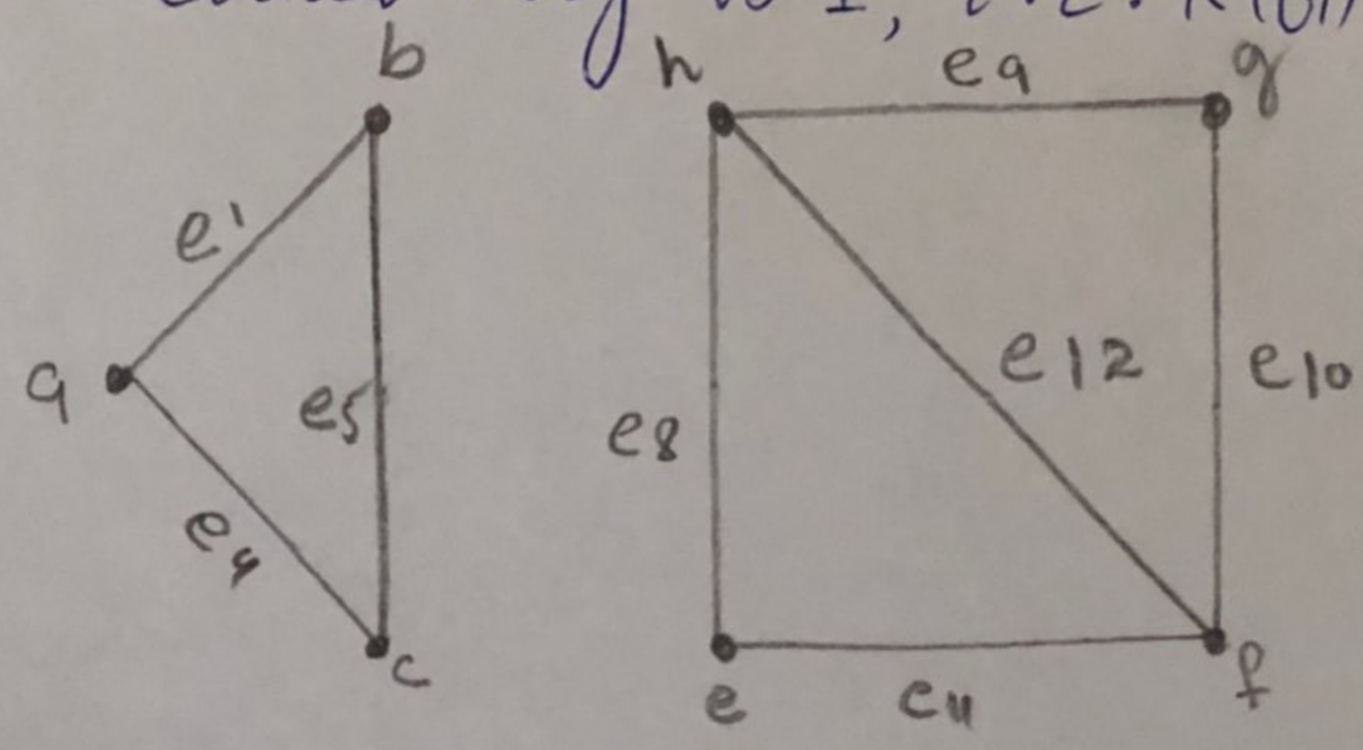
$\therefore$ Edge Connectivity $\lambda(G) = 2$



Vertex Connectivity:

By deleting only one vertex 'd' the graph is disconnected.

Hence Vertex Connectivity is 1, i.e. $K(G) = 1$.



⇒ Eulerian Graph:

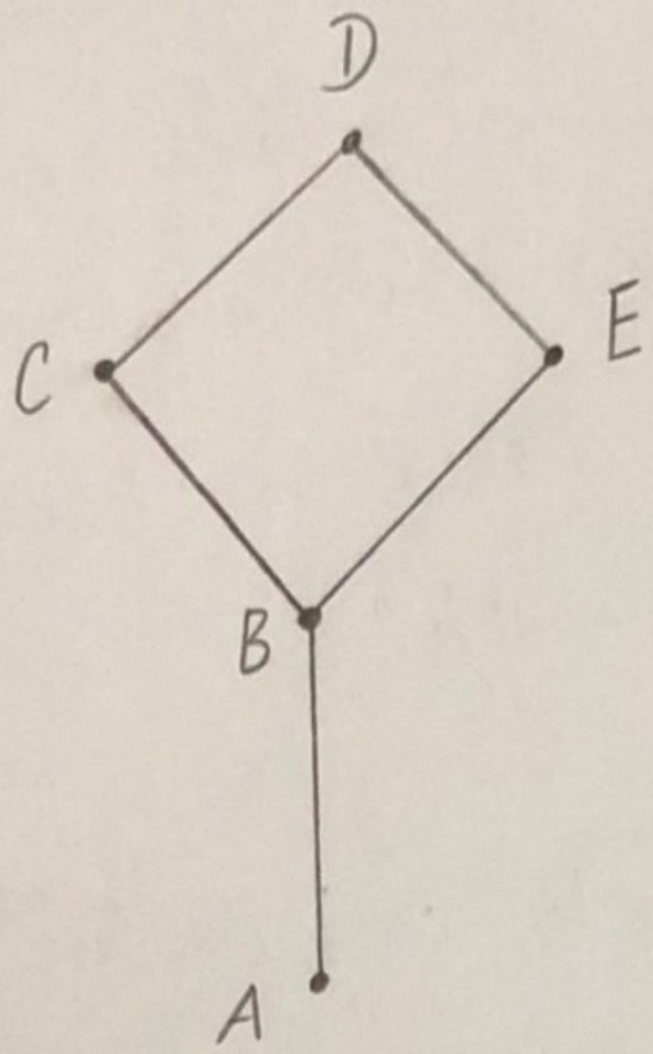
→ Traversable Graph: An Eulerian trial or Euler walk in an undirected graph is a walk that uses each edge exactly once. If an Euler trial exists in a given graph G, then G is called a "traversable graph" or "a Semi-Eulerian graph".

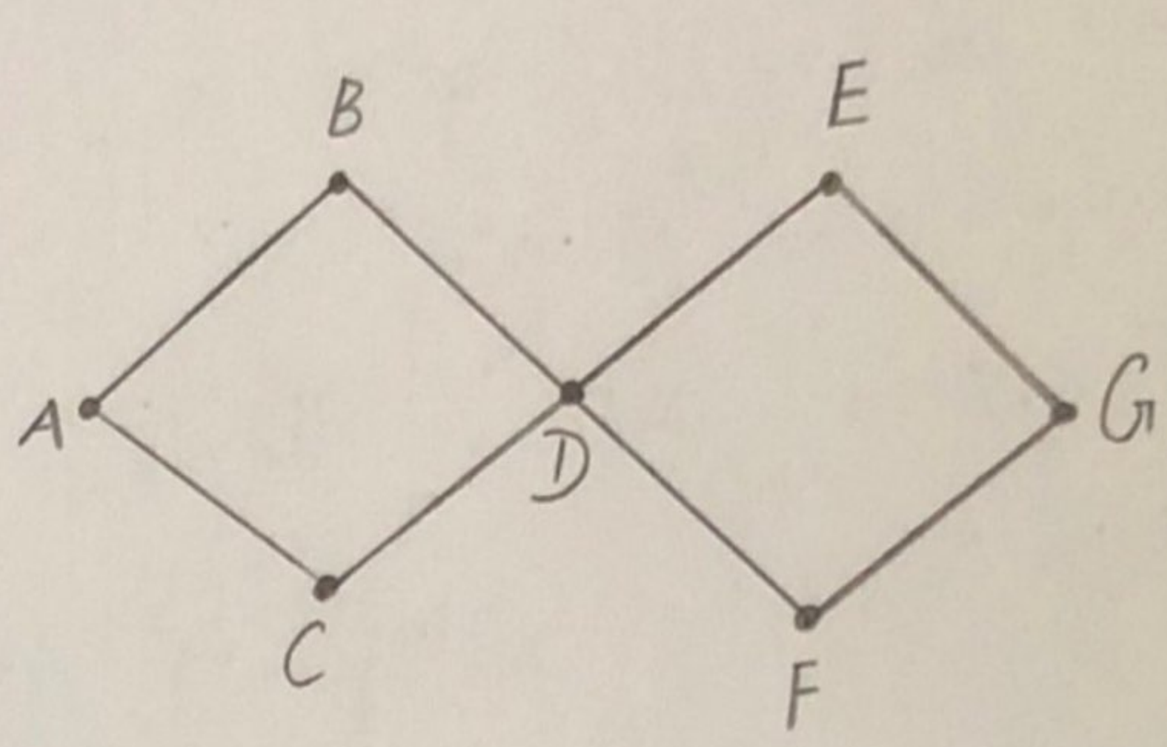
→ Eulerian Graph: An Eulerian cycle or Eulerian circuit or Euler tour in an undirected graph is a cycle that uses each edge exactly once. If such an Euler cycle exists in the graph concerned, then the graph is called an "Eulerian graph" or "a unicursal graph".

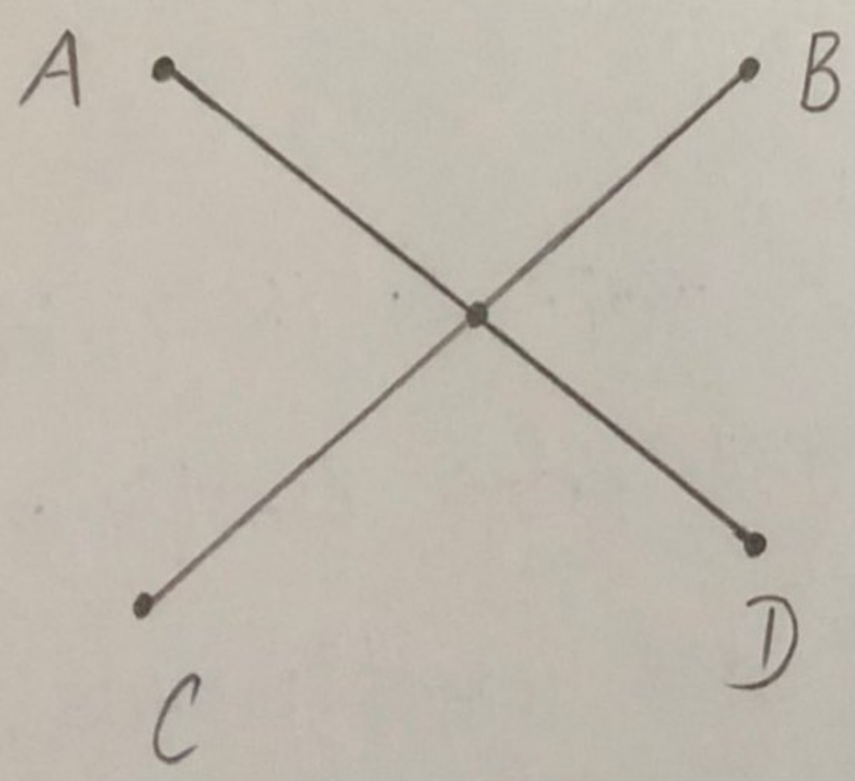
→ Euler Path: A path in a graph G is called "Euler Path" if it includes every edge exactly once. Since the path contains every edge exactly once, it is also called Euler trial.

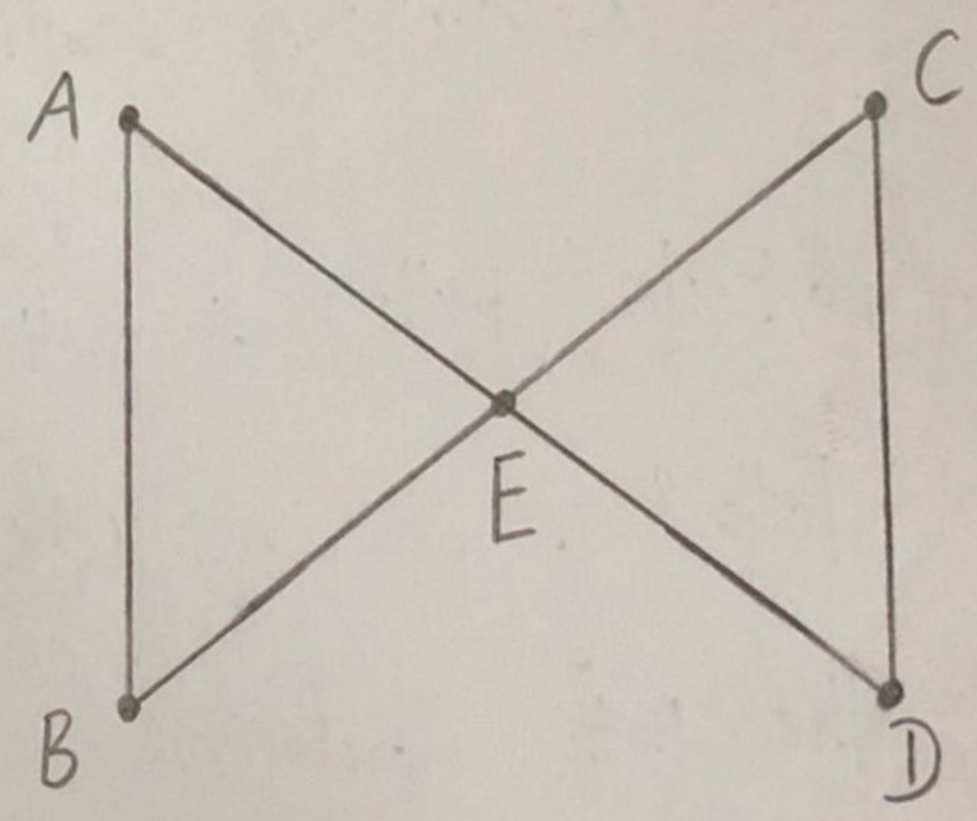
→ Euler Circuit: An Euler path that is circuit is called Euler circuit. A graph which has a Eulerian circuit is called an Eulerian graph.

Scanned with CamScanner









Scanned with CamScanner

⇒ Hamiltonian Graph:

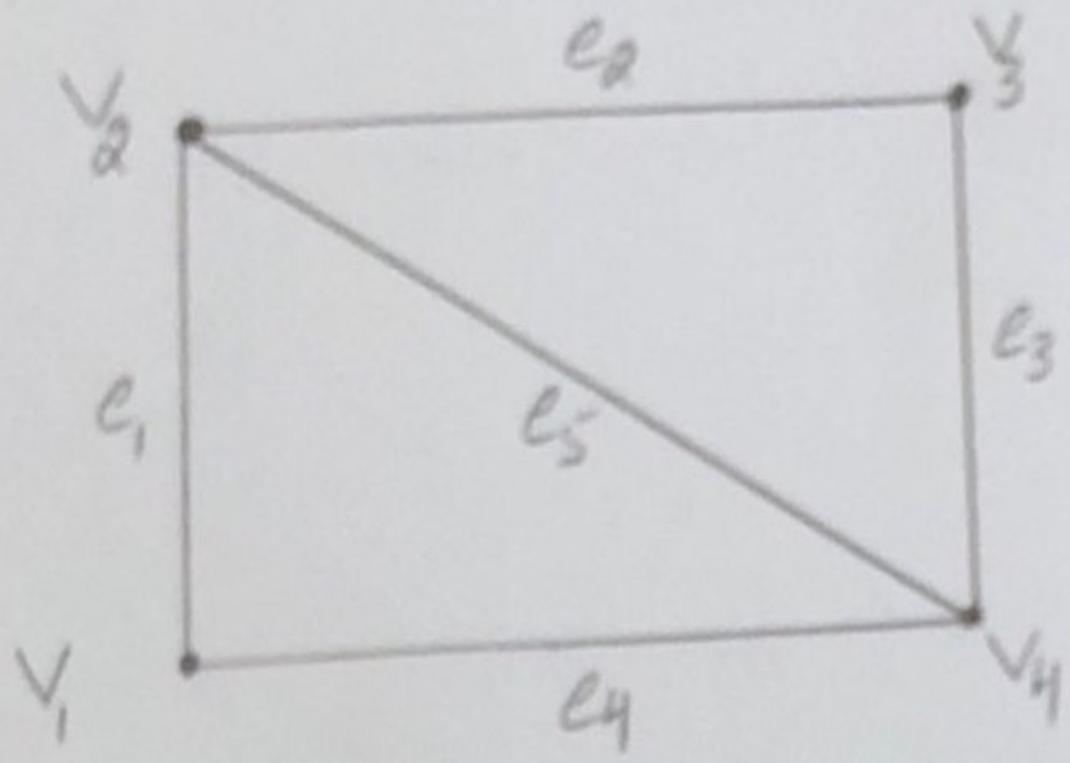
→ Traceable Graph: A Hamiltonian path (or traceable path) is a path in an undirected (or directed graph) that visits each vertex exactly once. A graph that contains a Hamiltonian path is called a "traceable graph".

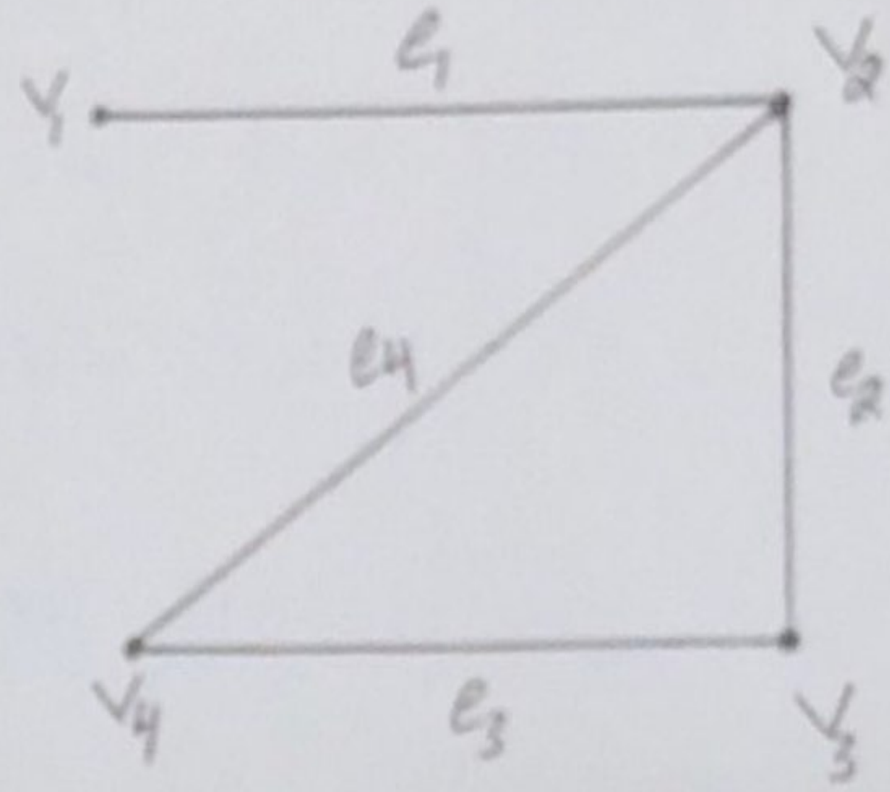
→ Hamiltonian Graph: A Hamiltonian cycle, or a Hamiltonian circuit, or a vertex tour or a graph cycle is a cycle that visits each vertex exactly once (except for the vertex that is both the start and end, which is visited twice). A graph that contains a Hamiltonian cycle is called a "Hamiltonian graph".

→ Hamiltonian Path: A Hamiltonian path is a simple path that contains all vertices of G where the end points may be distinct.

→ Hamiltonian Circuit: A circuit in a graph G that contains each vertex in G exactly once, except for the starting and ending vertex that appears twice is known as "Hamiltonian circuit".

A graph G is called a Hamiltonian graph if it contains a Hamiltonian circuit.

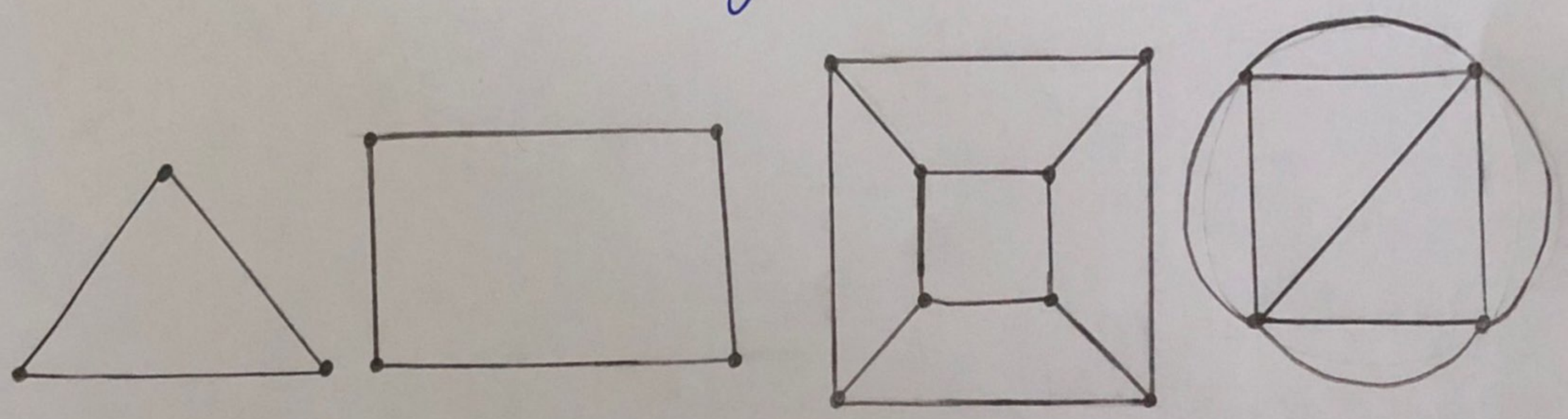

v_2
e_2
v_3
e_1
e_5
e_3
v_1
e_4
v_4

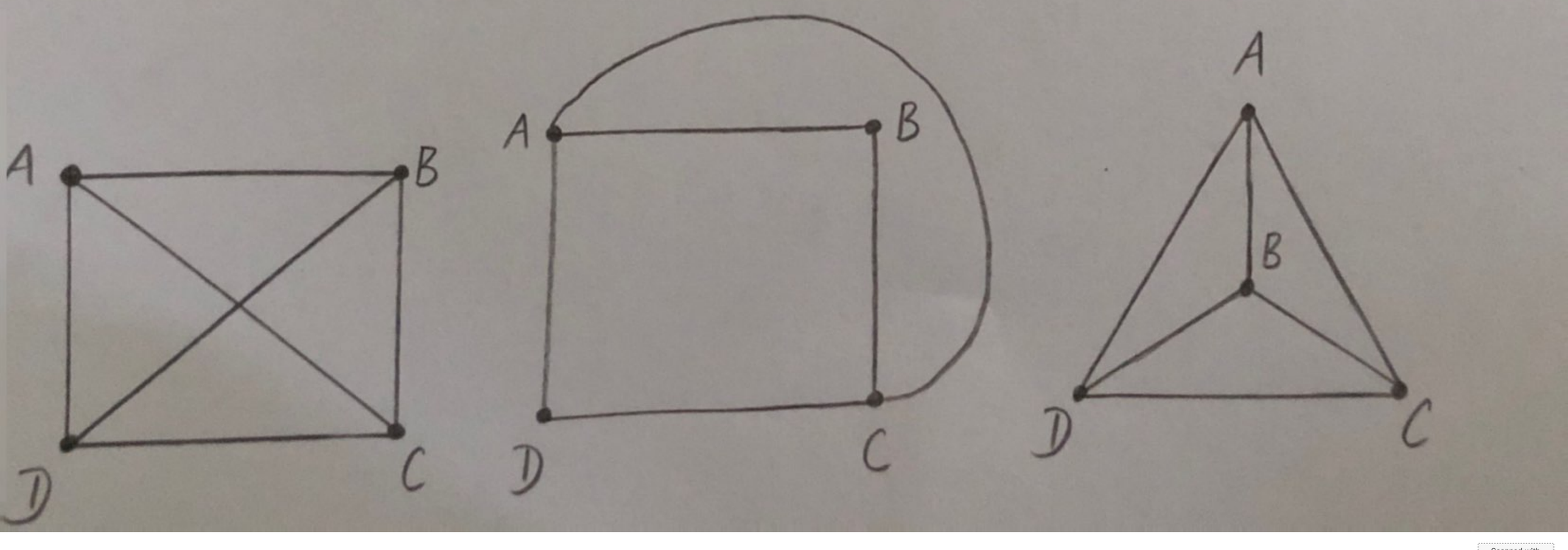

v_1
e_1
v_2
e_4
e_2
v_4
e_3
v_3

* **Planar Graph**: A graph G is said to be "planar graph" if G can be drawn in a plane so that no edges cross.

A graph G is said to be "non-planar" if it cannot be drawn on a plane without crossing its edges.

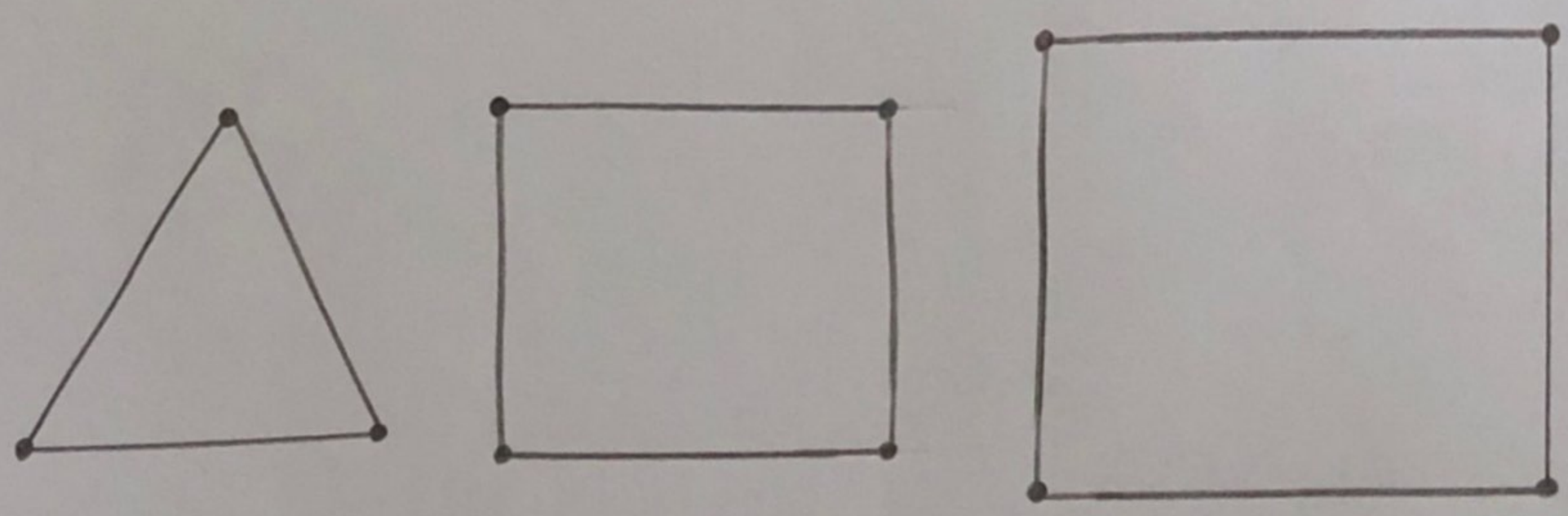
→ **Embedding**: The geometric representation of a graph on any surface such that no edge intersect is called "embedding".



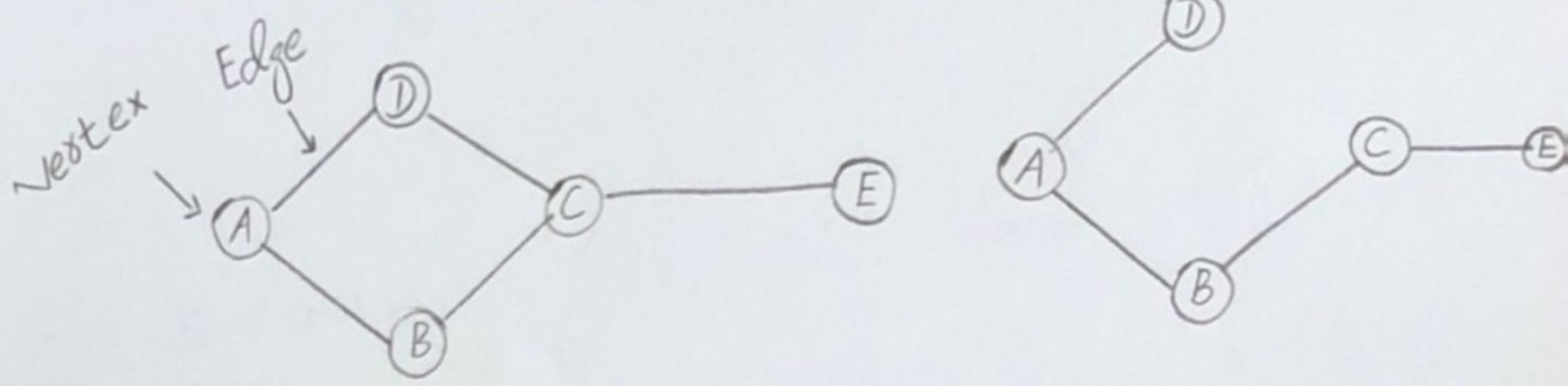


* Graph Colouring: The assign of colours to the vertices of G, one colour to each vertex, so that adjacent vertices are assigned different colours is called the "Proper Colouring of G" or simply "vertex colouring". If G has n colouring then G is called or said to be "n-colourable".

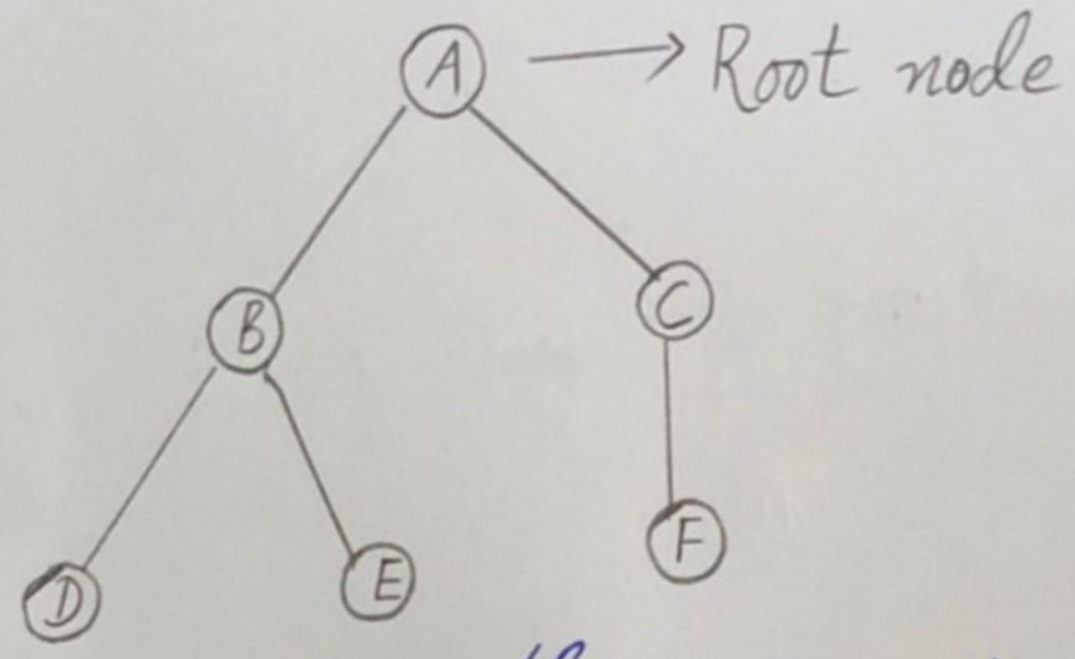
→ The "chromatic number" of a graph G is the minimum number of colours to colour the vertices of the graph G and is denoted by $\chi(G)$.



⇒ **Tree**: A tree is a Connected acyclic graph i.e; a Connected graph having no Cycle. Its edges are Called branches.



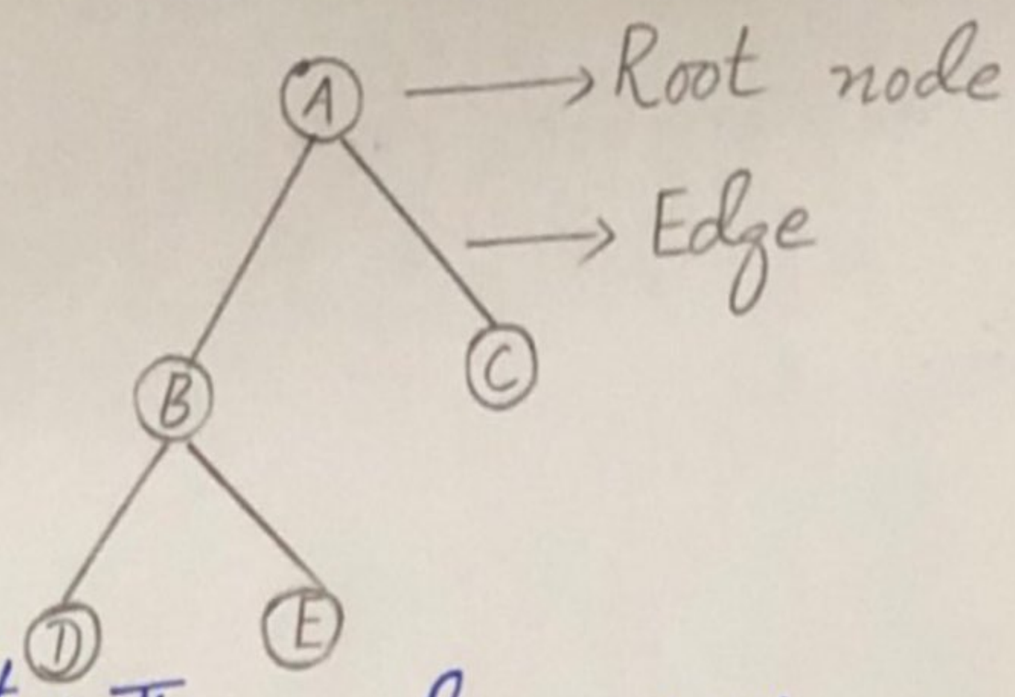
⇒ **Rooted Tree**: A rooted tree is a tree in which a particular vertex is distinguished from the others and is Called the root.



* **Tree Terminology** (Some Definitions)

→ **Root**: The first node in a tree is Called a root, it is often Called the top Level node.

→ **Edge**: The Connecting link between any two nodes is Called as edge. In a tree with "n" number of nodes there will be a maximum of "n-1" number of edges.



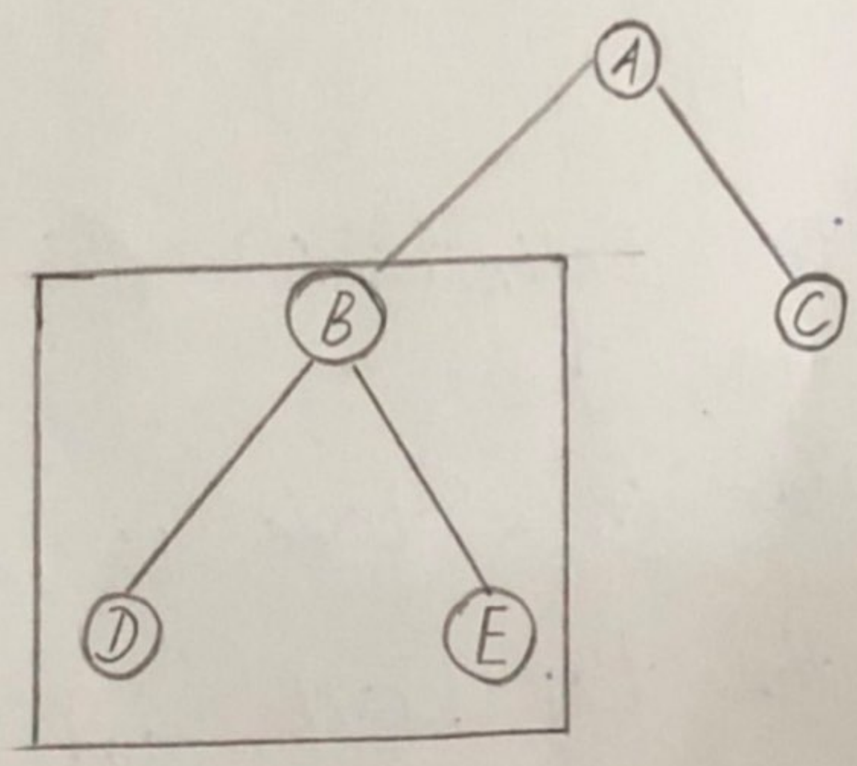
No: of nodes = 5

No: of edges = 4

⇒ **Parent**: The node which is predecessor of any node is called as parent node.

→ **Child**: The node which is descendant of any node is called as child node.

→ **Sibling**: Nodes which belong the same parent are called as siblings.



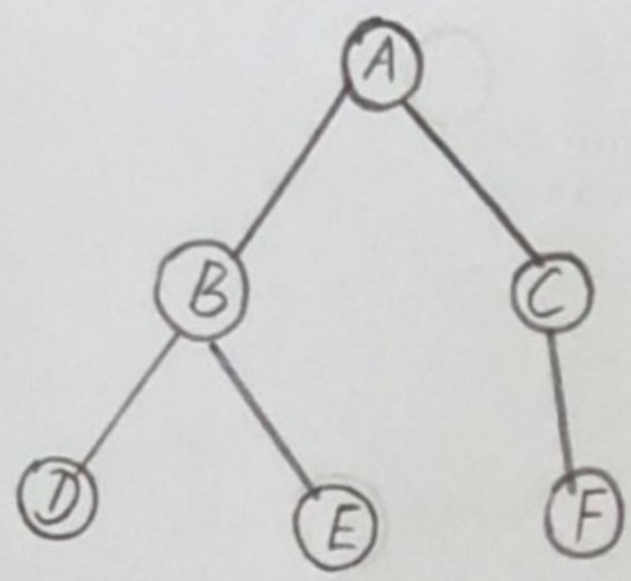
B → Parent node

D, E → Child node

D, E → Sibling

→ **Leaf Node**: The node which does not have a child or node with degree zero is called a leaf node. Leaf node is also called external / terminal nodes.

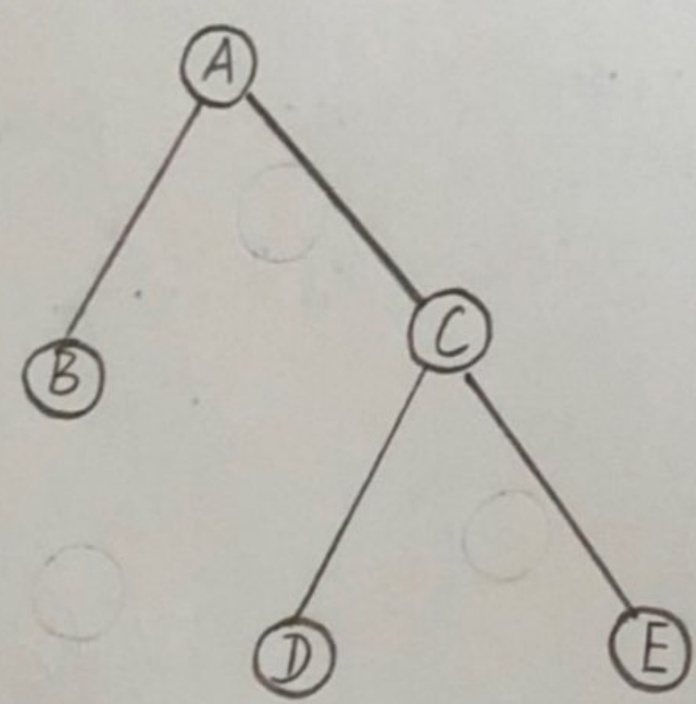
→ **Internal Nodes**: The node which has at least one child is called as internal node.



A, B, C → Internal node

D, E, F → Leaf node

→ **Degree**: The total number of children of a node or number of subtree of a node is called as degree of that node.

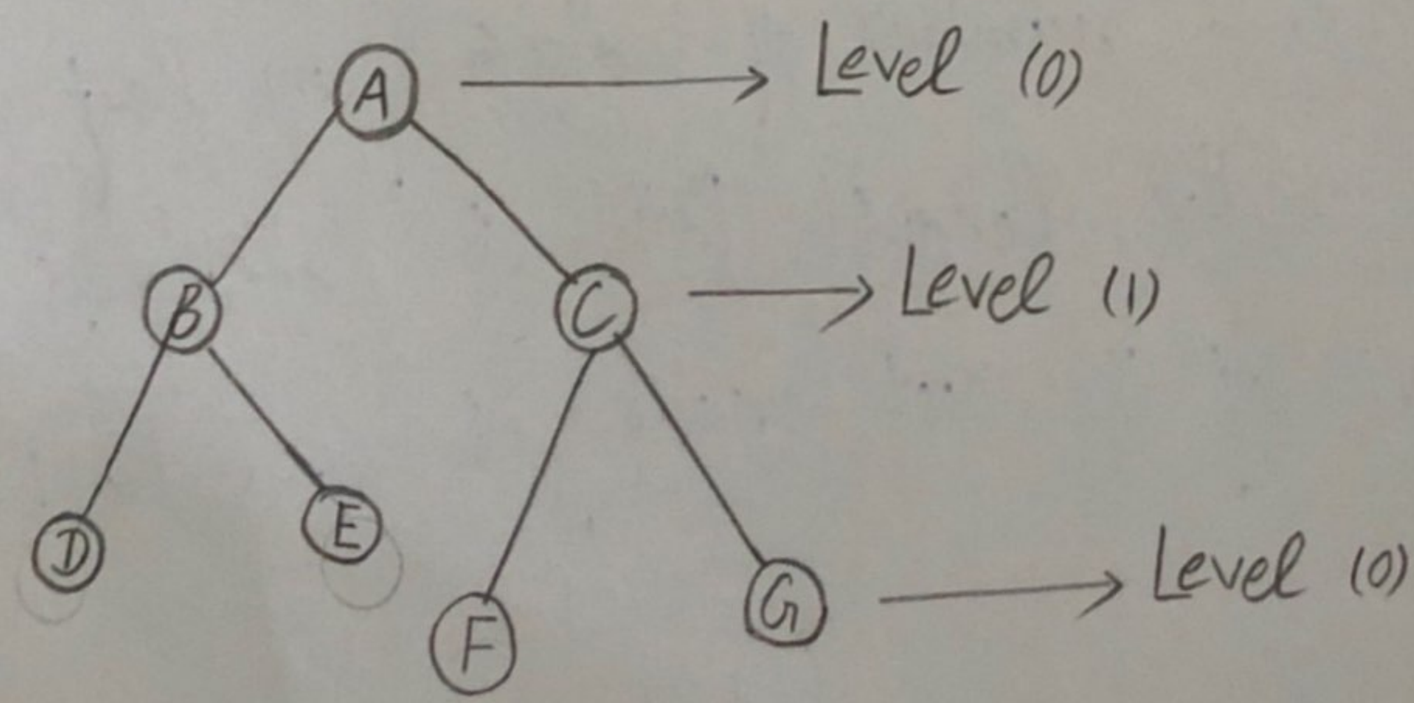


Degree of A → 2

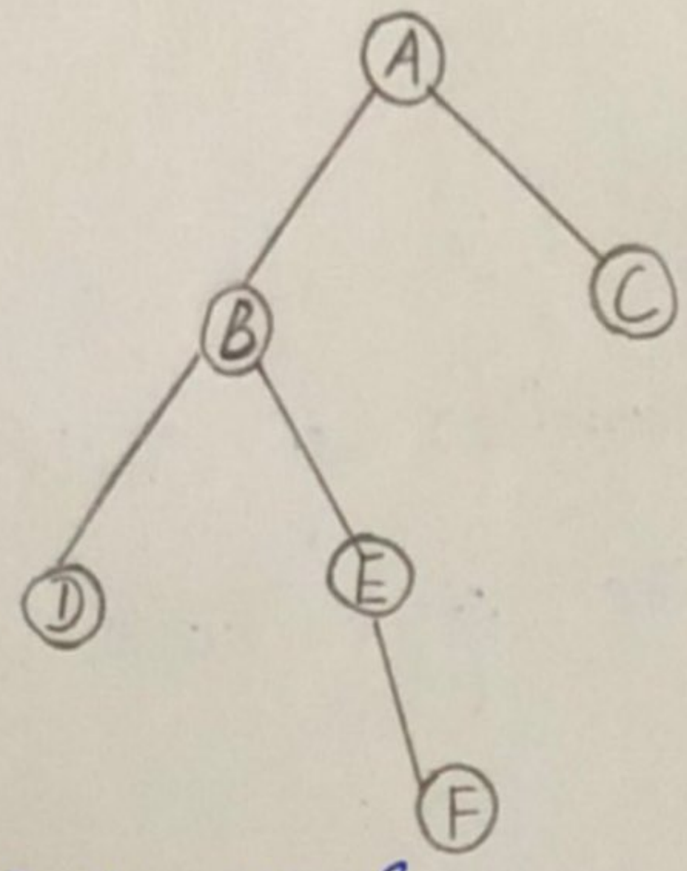
Degree of B → 0

Degree of C → 2

→ **Level**: The root node is said to be at level 0 and the children of root node are at level 1 and the children of root



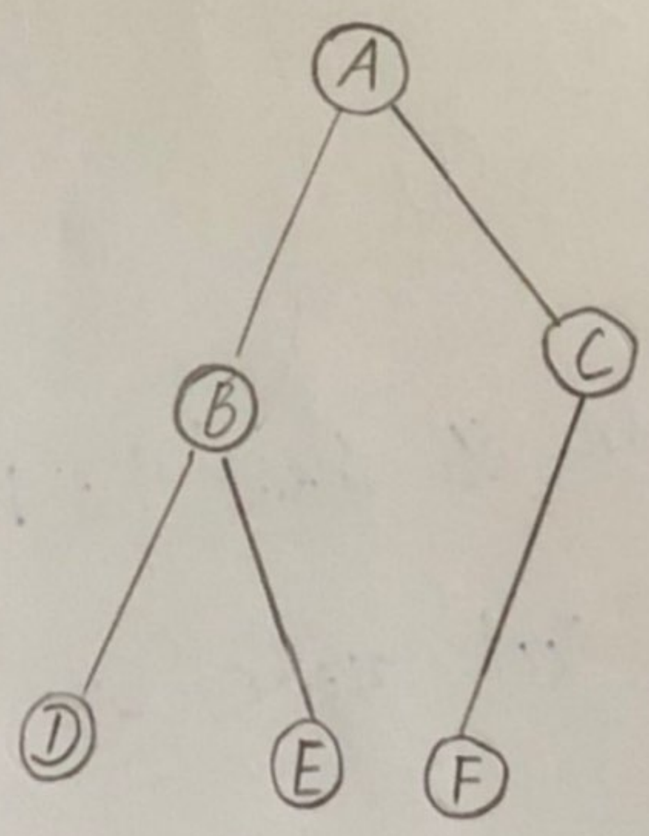
→ Path : The Sequence of nodes and edges from one node to another node is Called as path between that two nodes. Length of a path is total number of nodes in that path.



(A - B - E - F) path

Length = 4

→ Subtree : Each Child from a node forms a Subtree recursively. Every Child node Will form a Subtree on its parent node.



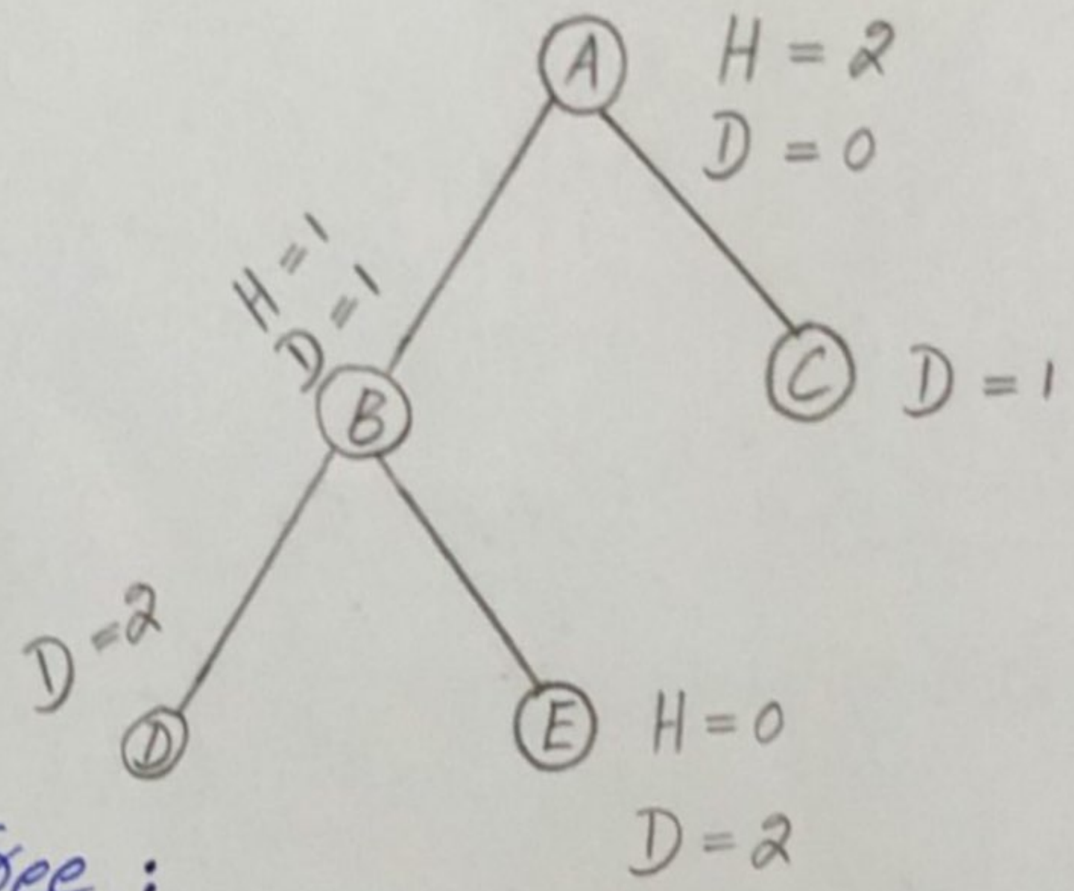
Node A

Left Subtree {B, D, F}

Right Subtree {C, F}

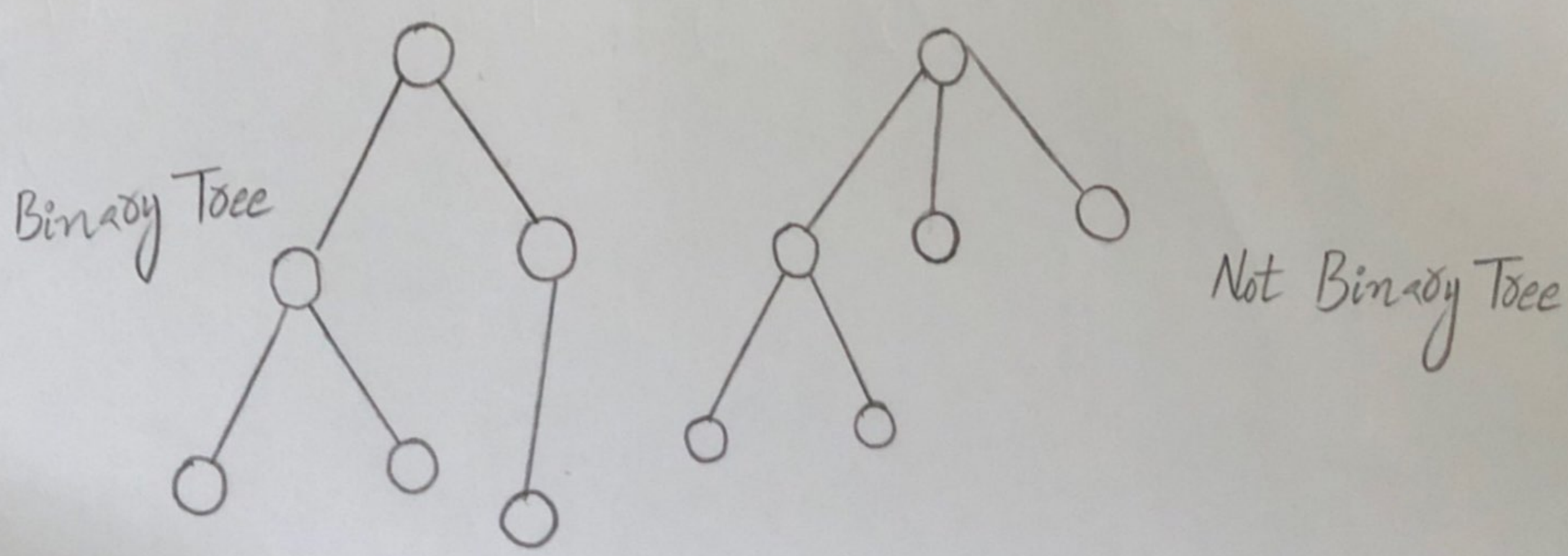
→ Height : The total number of edges from leaf node to Particular node in the longest Path is Called as height of that node. Height of the root node is said to be height of the tree.

→ **Depth**: The total number of edges from root node to a particular node is called as depth of that node. The total number of edges from root node to a leaf node in the longest path is said to be depth of the tree.

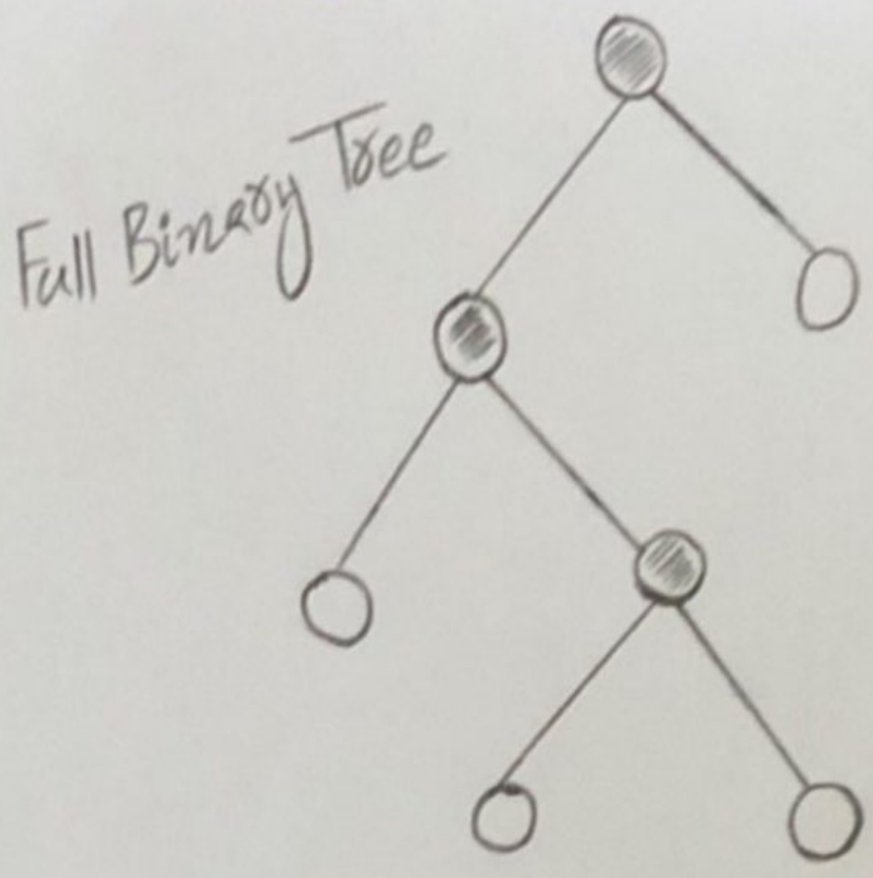


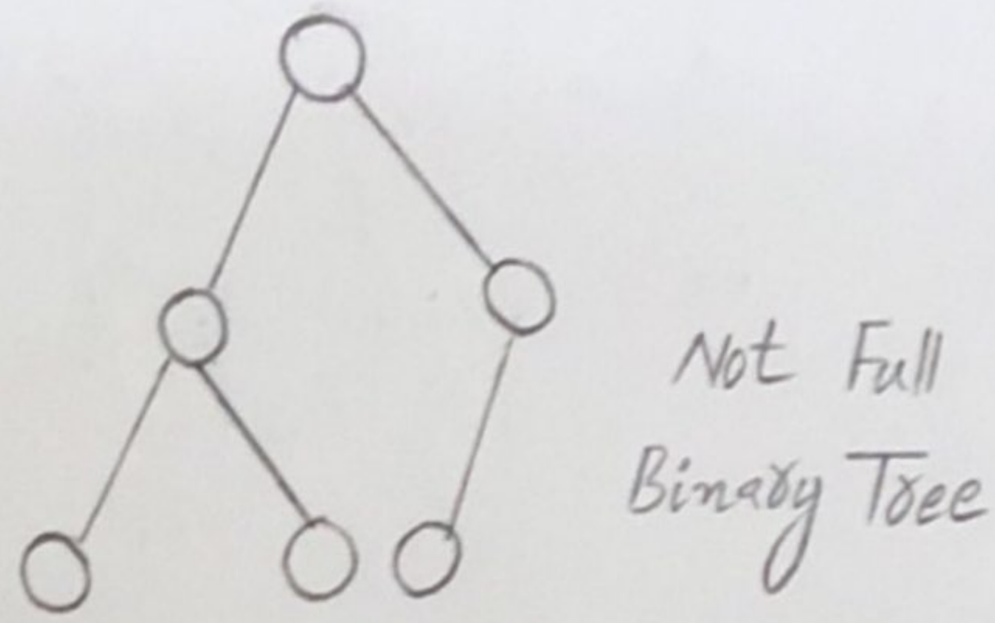
* Types of Tree :

(1) **Binary Tree**: A binary tree is a rooted tree in which each node has atmost two children. Each child in a binary tree is designated either a left child or a right child.

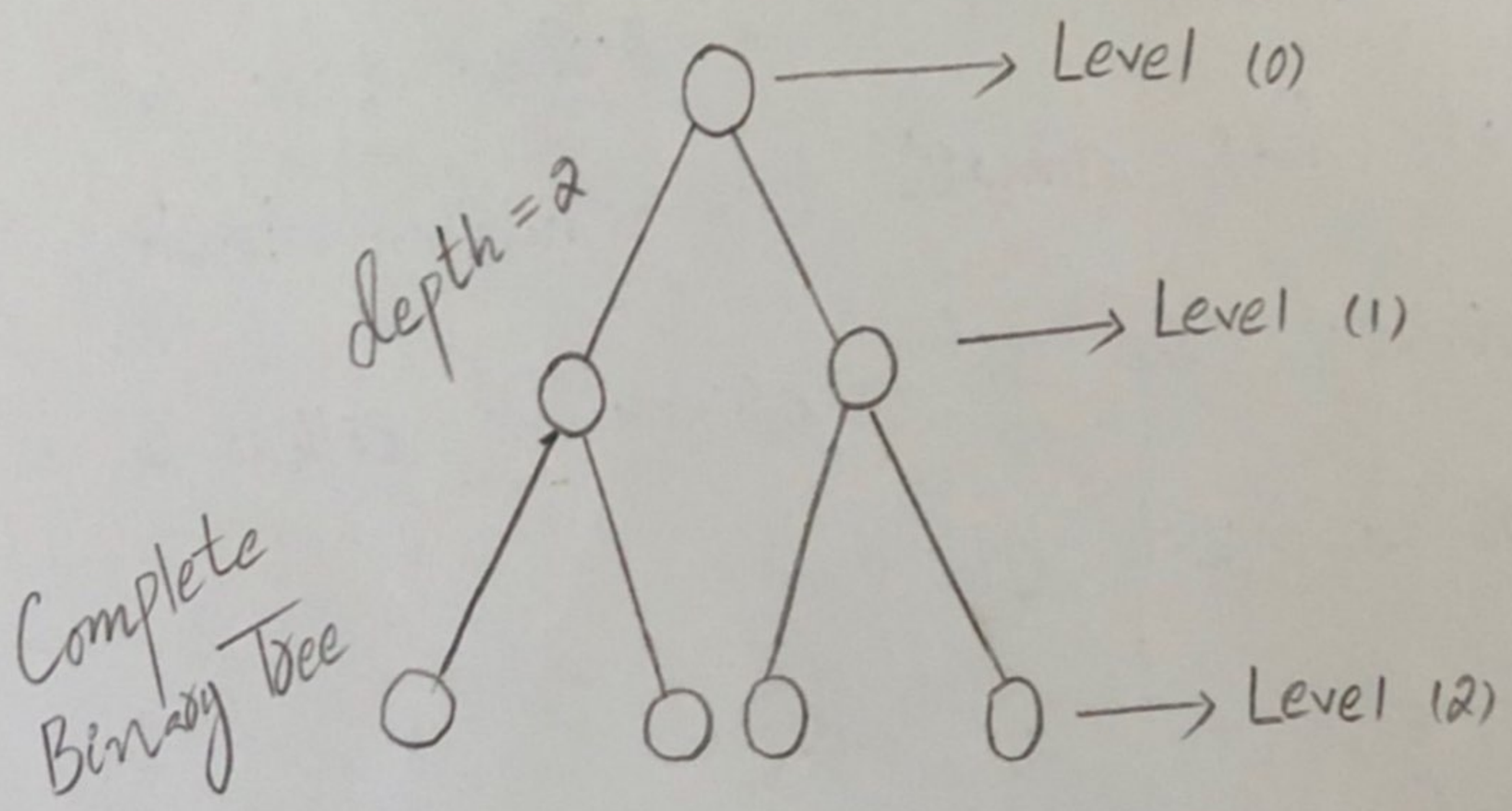


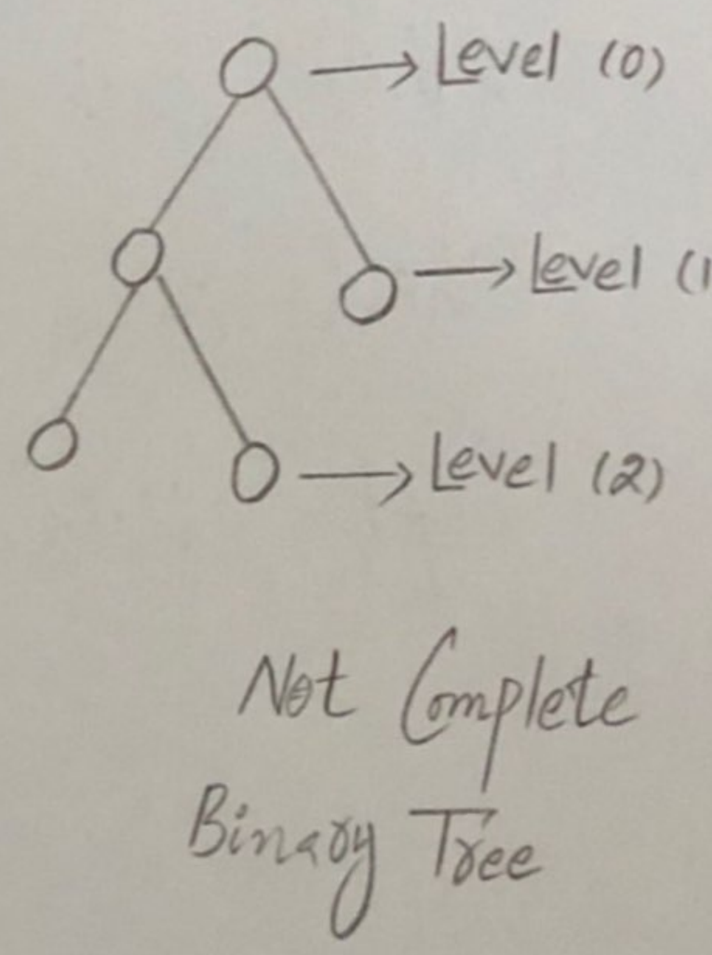
(2) Full Binary Tree / Strictly Binary Tree : A full Binary tree is a tree in which each internal node has exactly two children.





(3) Complete Binary Tree / Perfect Binary Tree : If all the leaves of a full binary tree are at level d, then we call such a tree as a Complete binary tree of depth d.

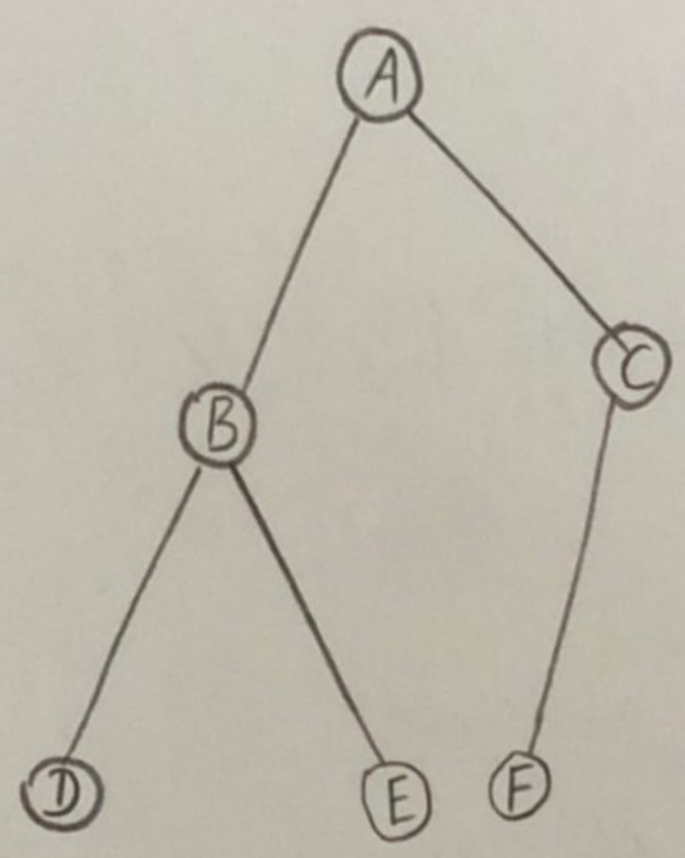


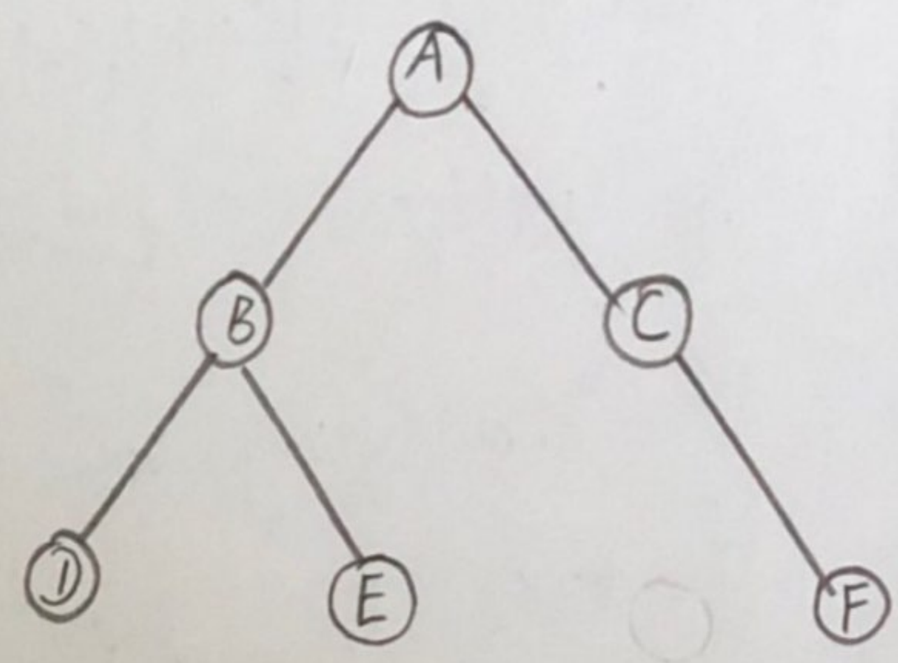


(4) Almost Complete Binary Tree : An almost Complete binary tree is a binary tree that Satisfies the following two (02) properties :

(i) All the levels are Completely filled except possibly the last level.

(ii) The Last level must be strictly filled from left to right.





* Tree Traversal :

A traversal of a tree is a process to traverse a tree in a Systematic way So that each vertex is visited exactly once.

(1) Preorder Traversal:

The preorder traversal of a binary tree is defined recursively as follows:

i. Visit the root

ii. Traverse the left subtree in preorder

iii. Traverse the right subtree in preorder

(2) Postorder Traversal:

The postorder traversal of a binary tree is defined ~~as~~ recursively as follows:

i. Traverse the left subtree in postorder

ii. Traverse the right subtree in postorder

iii. Visit the root

(3) Inorder Traversal:

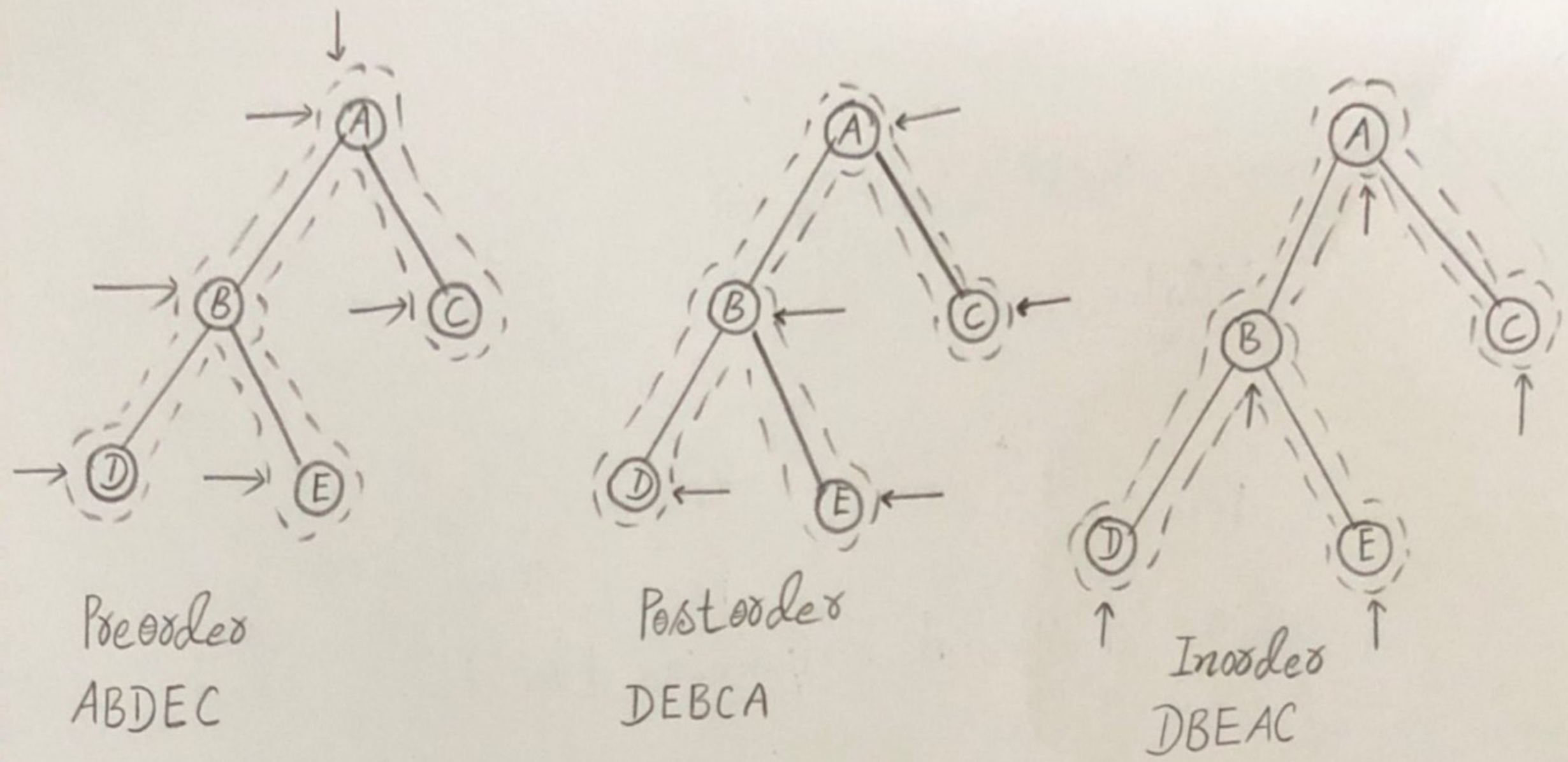
The inorder traversal of a binary tree is defined recursively as follows:

i. Traverse in inorder the left subtree

ii. Visit the root

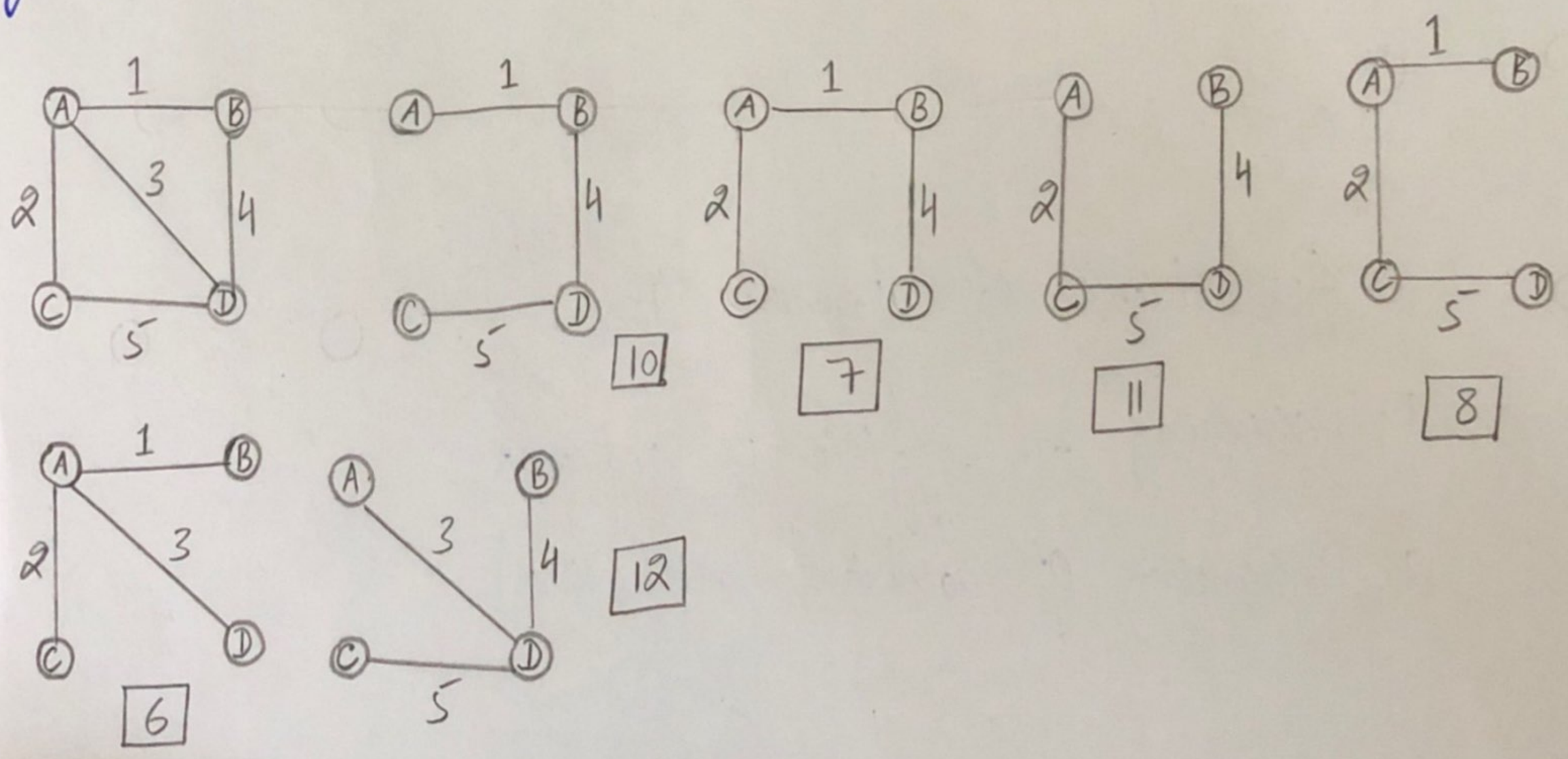
iii. Traverse in inorder the right subtree

Scanned with CamScanner



* Spanning Tree / Minimum Cost Spanning Tree:

→ Spanning Tree: Let $G = (V, E)$ be an undirected connected graph. A subgraph $T = (V, E')$ of G is a spanning tree of G if and only if T is a tree (i.e; no cycle exist in T) and contains all the vertices of G.



→ Minimum-Cost Spanning Tree: Let G be a undirected connected weighted graph. A spanning tree of a graph G with minimum cost is called minimum cost spanning tree.

* Find MCST, Two algorithms are used:

1. Prim's Algorithm
2. Kruskal's Algorithm

→ General MCST algorithm is

```
MCST (G, w)
{
  A = ϕ
  While (A is not a Spanning tree)
  {
  find an edge (u, v) that is
  A = A ∪ {(u, v)}
  }
  return A
}
```

→ Applications of Spanning Tree:

1. Designing an efficient network
2. Designing of routing network

⇒ Prim's Algorithm :

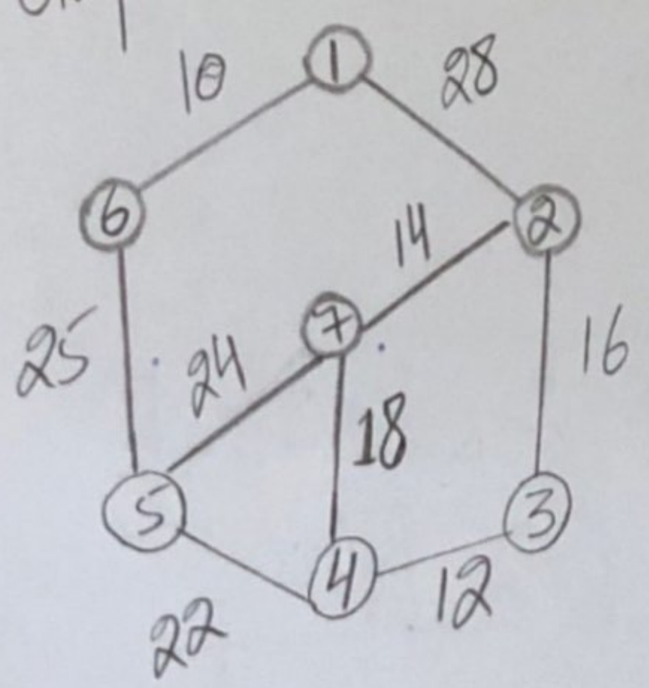
// Input : A undirected Connected weighted graph G = (V, E

// output : A minimum Cost Spanning tree T = (V, E') of G

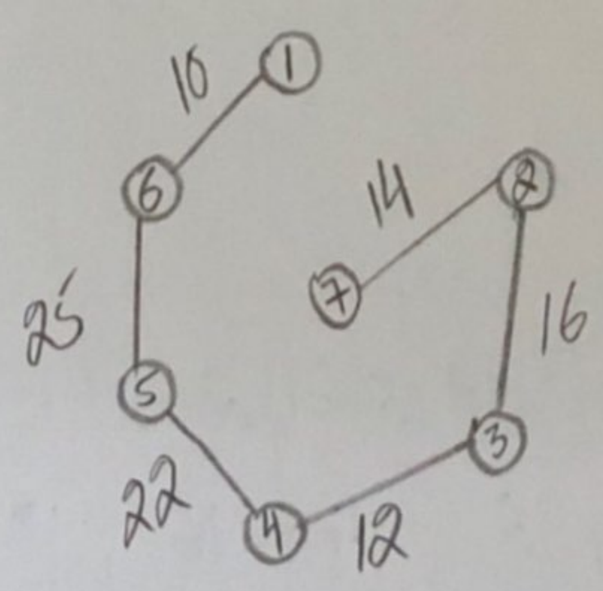
Steps:

1. Select any vertex from graph G.
2. At each Stage, Select the edge of Smallest Weight joinin
   a vertex already included to vertex not yet included
3. Continue until all vertices are included.

Graph



Minimum Cost Spanning Tree



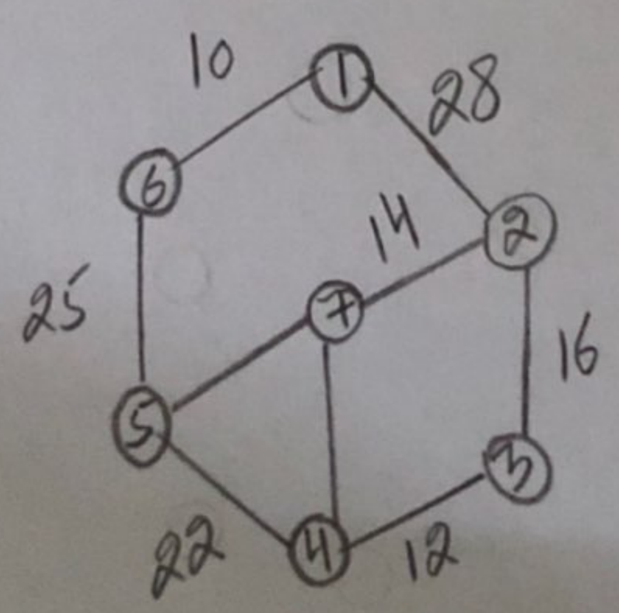
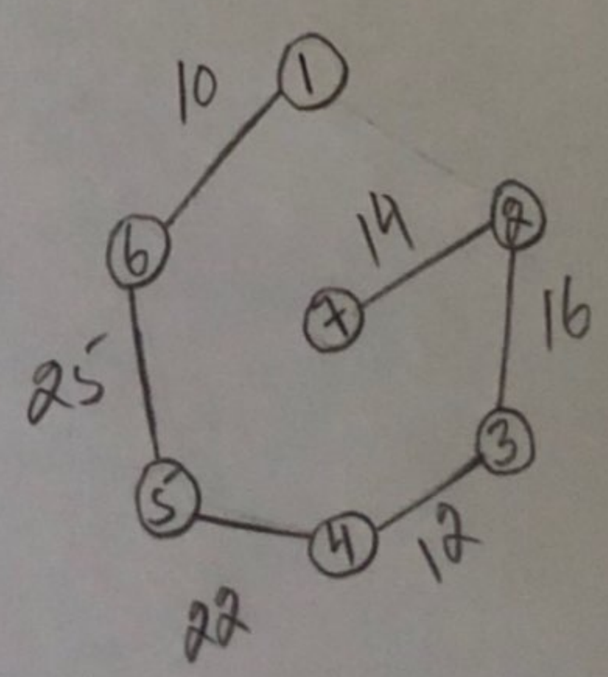
Cost = 99

⇒ Kruskal's Algorithm :

// Input : A undirected Connected weighted graph $G = (V, E$

// output : A minimum Cost Spanning tree $T = (V, E')$ of G.

Steps:

1. Select an edge of minimum weight.
2. At each Stage, Select from the edges not yet Selected the edge of Smallest weight whose inclusion will not Create a Cycle.
3. If G has n vertices, Stop after $n-1$ edges have been Selected otherwise repeat Step 2.

Edges	1-6	3-4	2-7	2-3	4-7	4-5	5-7	5-6	1-2
Weight	10	12	14	16	18	22	24	25	28