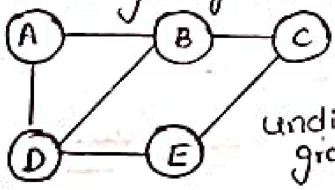
# UNIT-4 : GRAPH THEORY.

A Graph G is a collection of points called vertices and collection of lines called edges, each of which joins either in pair of vertices or a single vertex.

A graph G is an ordered pair G = (V, E) where V is non empty set of vertices and E is the non-empty set of edges.

## UNDIRECTED GRAPH :

An undirected graph is a type of graph where the edges have no specified direction assigned to them. Nodes are unordered pairs in the def.ⁿ of every edge.

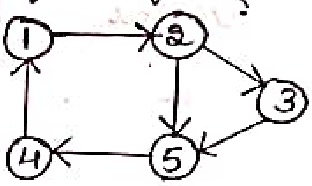


undirected graph

[Application :- Undirected graph are used to model social networks, Traffic flow optimization]

## DIRECTED GRAPH :

A graph in which edge has direction. The nodes are ordered pairs in the definition of every edge.

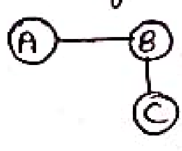


Directed graph.

## BASIC TERMINOLOGIES OF A GRAPH.

- Adjacent vertices :- Vertices are adjacent if there is an edge between them (or)

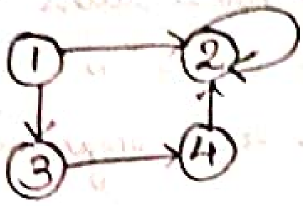
  Two vertices are said to be adjacent if there is an edge between these two vertices.




A&B
B&C are adjacent vertices

A&C are not adjacent.

- **Loop** :- A loop (also called a self-loop) is an edge that connects a vertex to itself.

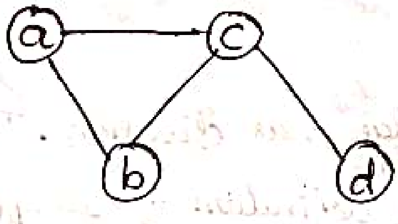



- **Degree of a vertex** :-

The number of edges incident with a vertex $v$ of a graph, with self-loops counted twice is called the degree of the vertex $v$, denoted by $deg(v)$.

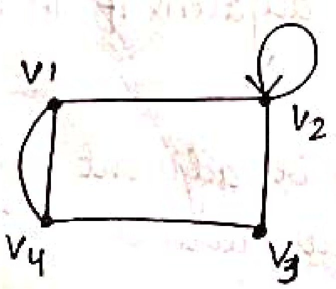
If the degree of a vertex is even, then the vertex is called an even vertex and if the degree of a vertex is odd then the vertex is called an odd vertex.

eg:-




| Vertex | Degree | Even/Odd |
|---|---|---|
| a | 2 | Even |
| b | 2 | Even |
| c | 3 | Odd |
| d | 1 | Odd. |

eg:-

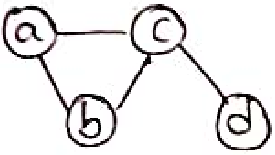



$deg(v_1) = 3$
$deg(v_2) = 4$
$deg(v_3) = 2$
$deg(v_4) = 3$

# THE HAND SHAKING LEMMA :

The sum of the degrees of the vertices of a graph is equal to twice the number of edges.

$$\sum_{v \in V} deg(v) = 2|E|$$




| Vertex | Degree |
|---|---|
| a | 2 |
| b | 2 |
| c | 3 |
| d | 1 |

The sum of all the degree's of all vertices = 8.

The number of edges = 4

$\therefore 8 = 2(4)$

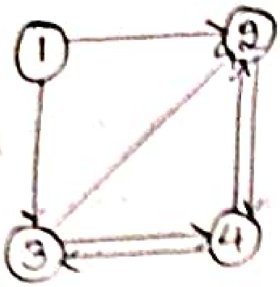
Proof:- Since the degree of a vertex is the number of edges incident with that vertex, the sum of the degree counts the total number of times an edge is incident with a vertex. Since every edge is incident with exactly two vertices, each edge gets counted twice, once at each end.

Thus the sum of the degrees equal twice the number of edges.

- Indegree of a graph & Outdegree of a graph

Indegree of a vertex is defined as the number of incoming edges incident on a vertex in a directed graph.

Outdegree of a vertex is defined as the number of outgoing edges from a vertex in a directed graph




indegree(1) = 0 Outdegree(1) = 2

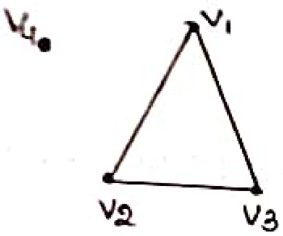
indegree(2) = 3 Outdegree(2) = 1

indegree(3) = 2 Outdegree(3) = 2

indegree(4) = 2 Outdegree(4) = 2

- **Isolated Vertex :-**

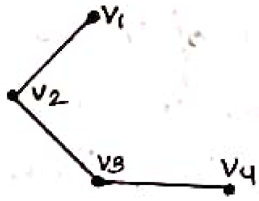
A vertex of degree zero in a graph is called an Isolated vertex.




The vertex V4 is an isolated vertex.

- **Pendant vertex :-**

Let G be a graph, A vertex v of G is called a pendant vertex if and only if v has degree 1.

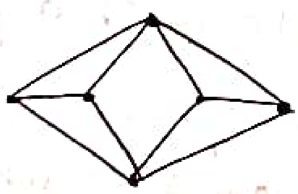



The vertices $v_1$ and $v_4$ are pendant vertices.

- **Order of a graph :-**

If G=(V,E) is a finite graph, then the number of vertices, denoted by |V| is called the order of G.

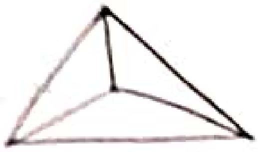
eg:-



Order - 6.

- **Size of a graph :-**

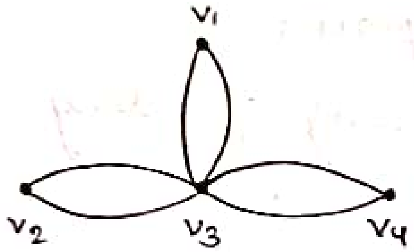
If G=(V,E) is a finite graph, then the number of edges in G is called the size of G. It is denoted by |E|.



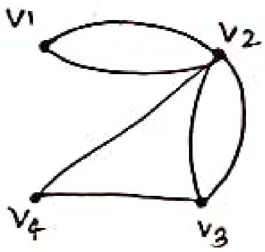
Size of the graph = 6.

- **Degree Sequence of a graph:**

The degree sequence of a graph is the sequence of the degrees of the vertices of the graph in non-increasing order.



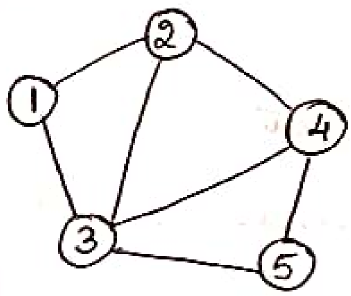

{2, 2, 2, 6}




{2, 2, 3, 5}

- **Walk**

A walk is a sequence of vertices and edges of a graph i.e. if we traverse a graph then we get a walk.




- Edge & vertices can both be repeated.

Here 1 → 2 → 3 → 4 → 2 → 1 → 3 is a walk.

**Types of walks.**

**Open walk :-** A walk is said to be an open walk if the starting & ending vertices are different.

1 → 2 → 3 → 4 → 5 → 3 is an openwalk.

Closed walk :-

A walk is said to be closed walk if the starting and ending vertices are same.

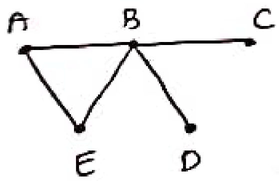
1 → 2 → 3 → 4 → 5 → 3 → 1 is a closed walk

NOTE:

- The number of edges which is covered in a walk is known as the length of the walk.
- The walk will be known as the Trivial walk if the length of the walk is zero
- In case of the open walk and closed walk, the edges and vertices can be repeated.

Trails :-

A trail in a graph is a walk that uses no edge more than once.




D → B → E → A is a trail.

C → B → A → E → B → D is a trail.

Path :- A path in a graph is a walk that visits no vertex more than once and no edges more than once.

(or) An open walk in which no vertex and edge appears more than once is called a simple path or a path.

D → B → E → A - Path, trail, walk

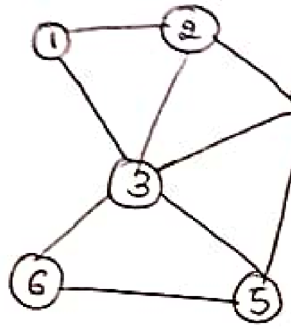
D → B → E → A → B - Not a path, (trail, walk)

## Circuit:

edges can repeat, but vertex can repeat

A closed walk in which no vertex (except its terminal vertices) appear more than once is called a circuit.

ex:-




Here $1 \to 2 \to 4 \to 5 \to 3 \to 1$ is a circuit

[A circuit can be described as a closed walk where no edge is allowed to repeat. In the circuit, the vertex can be repeated]

**Cycle** :- A cycle is a closed path in a graph, that starts and ends at the same vertex, and where no edges & no vertices are repeated.

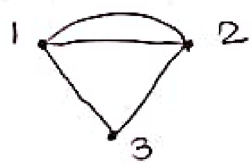
ex:- In the above graph,

(1) → (2) → (4) → (3) → (1) is a cycle.

## TYPES OF GRAPHS

1. **Simple graph** :-
   A graph is called simple graph if the graph is undirected and does not contain any self loops or multiple edges.

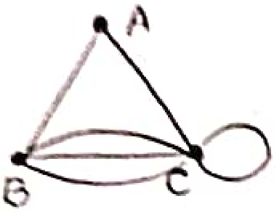

2. **Multi-graph** :-
   A graph having no self-loops but having parallel edge(s) in it is called as a multi-graph.



3. <u>Pseudo Graph</u> :-

A graph having a parallel edges and having self loop(s) in it is called a pseudo graph.

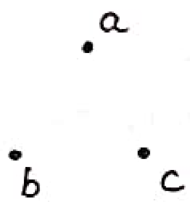
ex:-




4. <u>Null graph</u> :-

A null graph <u>does not contain any edges</u> in it.

eg:-




5. <u>Trivial graph</u> :-

A graph having only <u>one vertex</u> in it is called as a trivial graph. It is the smallest possible graph.

ex:-


6. <u>Sub graph</u> :-

A subgraph is a graph that is entirely contained within another graph.

(a) (b)

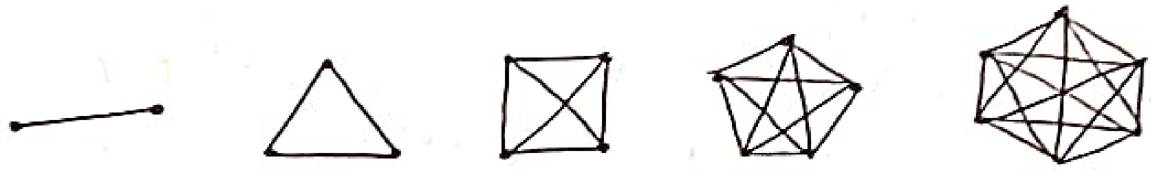
The graph (b) is a subgraph of graph (a)

7. Complete graph :-

A simple graph G is complete, if every vertex in G is connected with every other vertex.

The complete graph with 'n' vertices is denoted by $K_n$ & has $\frac{n(n-1)}{2}$ undirected edges.

The graphs $K_n$ for n = 2, 3, 4, 5, 6 ...



$K_2$ $K_3$ $K_4$ $K_5$ $K_6$

8. Regular graph.

A graph where each vertex has the same number of neighbours. That is, every node or vertex has the same degree. A regular graph with vertices of degree k is called a k-regular graph.

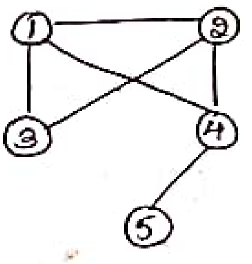
2-regular graph 2-regular graph

9. Connected and strongly connected graph.

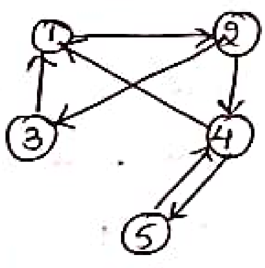
An undirected graph is called connected if for every pair of vertices there exists a path between them.

For a directed graph, for every pair of vertices (i, j) there exists a path from i to j and from j to i, then the graph is strongly connected.

ex:-




Connected graph

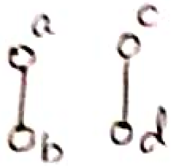



Strongly connected graph

10. <u>Disconnected graph</u>

A graph in which there does not exist any path between at least one pair of vertices is called as a disconnected graph.

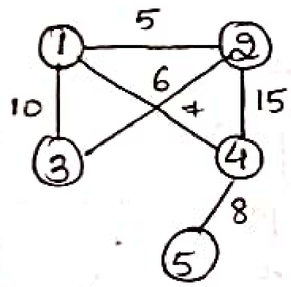
ex:-




11. <u>Weighted Graphs</u>

A graph G is said to be weighted graph if G consists of a set V of vertices, a set E of edges and weights are assigned to every edge.

ex:-

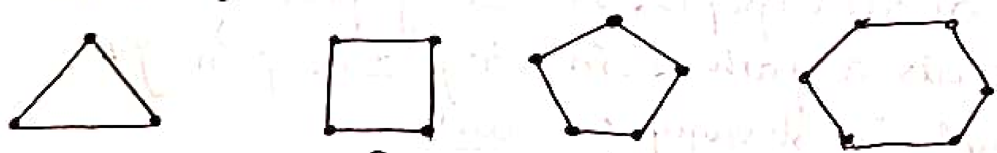



12. <u>Cycle graph</u>:-

A simple graph of 'n' vertices ($n \geq 3$) and 'n' edges forming a cycle of length 'n' is called as a cycle graph

* In a cycle graph, all the vertices are of degree 2.

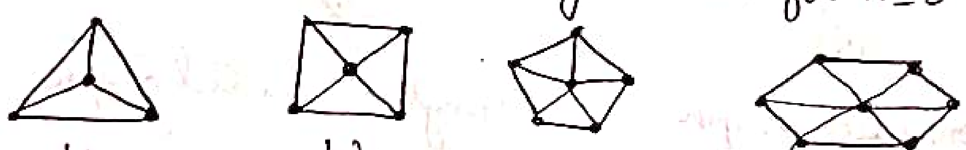
eg:-



$C_3$ $C_4$ $C_5$ $C_6$

13. <u>Wheels</u>:- A wheel $W_n$ is obtained when we add an additional vertex to a cycle $C_n$, for $n \geq 3$.

ex:-

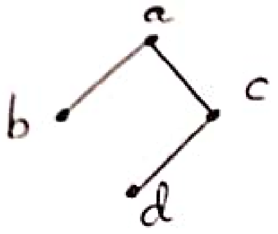


$W_3$ $W_4$ $W_5$ $W_6$

14. Acyclic graph :-

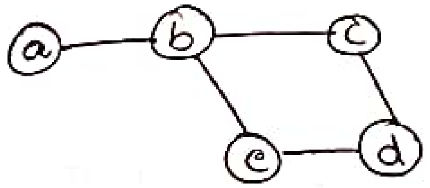
A graph which is not containing any cycle in it is called as an acyclic graph.

ex:-



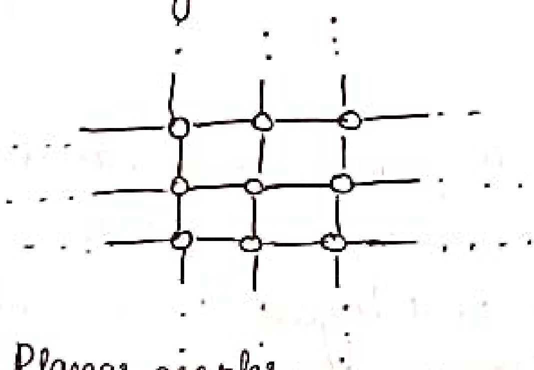

15. Finite graph :-

A graph consisting of finite number of vertices and edges is called a finite graph.




16. Infinite graph :-

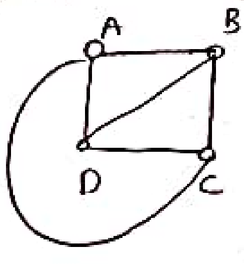
A graph consisting of infinite number of vertices and edges is called as an infinite graph.

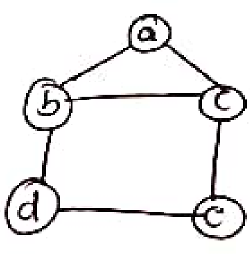


17. Planar graphs :-

A graph is said to be planar if it can be drawn in a plane so that no edges intersect except at their endpoints.

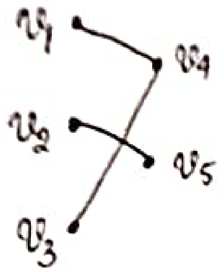
ex:-



18. <u>Bipartite graph</u> :-

A bipartite graph is a graph where its vertices can be divided into two disjoint sets, and each edge connects a vertex in one set to a vertex in the other set.

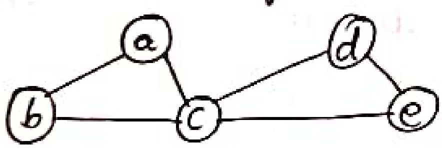



$V = \{v_1, v_2, v_3, v_4, v_5\}$

$V_1 = \{v_1, v_2, v_3\}$ $V_2 = \{v_4, v_5\}$

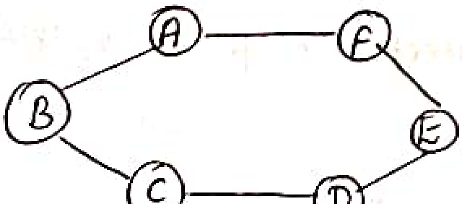
19. <u>Euler graph</u> :-

Euler graph is a connected graph in which all the vertices are of even degree.

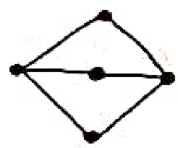



20. <u>Hamiltonian graph</u> :-

A Hamiltonian graph is a graph which has a closed path that visits each vertex exactly once, ending on the same vertex as it began. This closed path is also called a Hamiltonian cycle.

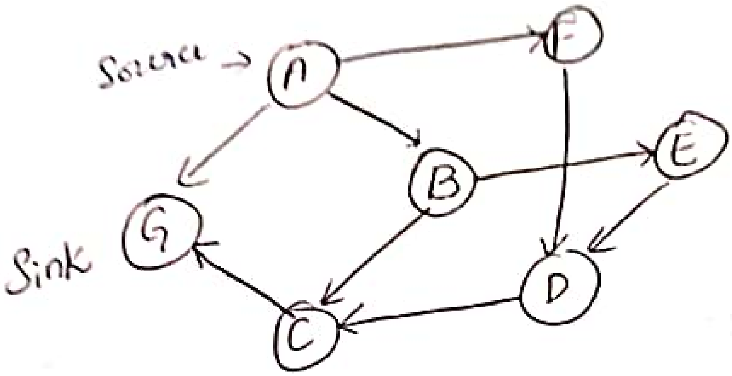



Hamiltonian



non - Hamiltonian .

21. Directed acyclic graph :- A directed graph that has no cycles.



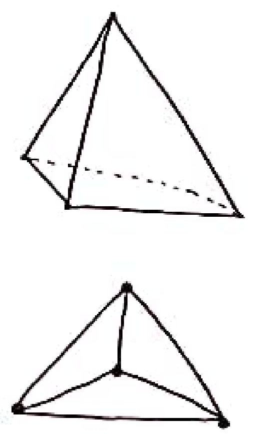

- A vertex v is known as a source if it has a positive out-degree but zero in-degree.
- A vertex v is known as a sink if it has a positive indegree but a zero out-degree.
- Every DAG has atleast one source, and atleast one sink.

22. Platonic graph :- The graphs of the platonic solids have been called platonic graphs.

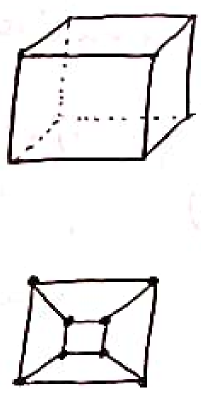
There are five platonic graphs.

1. Tetrahedral or Tetrahedron - 4 vertices & 6 edges
2. Cube - 8 vertices & 12 edges
3. Octahedron - 6 vertices & 12 edges
4. Dodecahedron - 20 vertices & 30 edges
5. Icosahedron - 12 vertices & 30 edges.

ex:-



Tetrahedron



Cube

# Representation of Graphs

The three important ways to represent a graph.

(1) Adjacency matrix

(2) Incidence matrix

(3) Adjacency list.

## Adjacency matrix :-

The Adjacency matrix is a matrix with 'n' rows and 'n' columns. Each entry in the matrix represents 1, if there exists an edge (i,j) between vertex i and vertex j and 0 when there is no edge. Therefore, the adjacency matrix contains only 0's and 1's for both directed & undirected graphs.

If a graph is a weighted graph then instead of 1's and 0's, we can store weight of the edge.

For an undirected graph G,

$$A(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \text{ (or) } (j,i) \in E \\ 0 & \text{otherwise.} \end{cases}$$

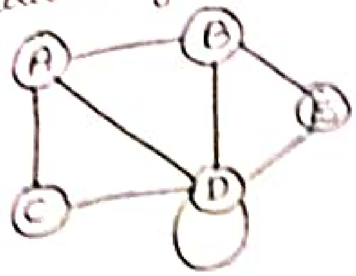
For a directed graph G,

$$A(i,j) = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise.} \end{cases}$$

For a weighted graph G,

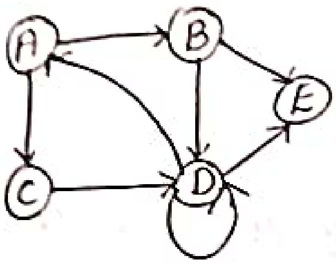
$$A(i,j) = \begin{cases} W(i,j) & \text{if } (i,j) \in E \\ \infty & \text{otherwise.} \end{cases}$$

Undirected graph.



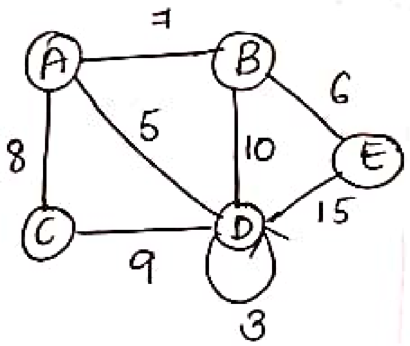

| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 0 | 1 | 1 |
| C | 1 | 0 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 1 | 1 |
| E | 0 | 1 | 0 | 1 | 0 |

Directed graph.



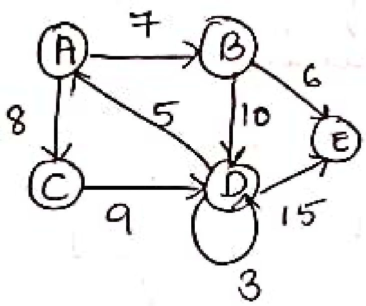

| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 | 0 |
| B | 0 | 0 | 0 | 1 | 1 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 1 | 0 | 0 | 1 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

Un-directed weighted graph.




| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | ∞ | 7 | 8 | 5 | ∞ |
| B | 7 | ∞ | ∞ | 10 | 6 |
| C | 8 | ∞ | ∞ | 9 | ∞ |
| D | 5 | 10 | 9 | 3 | 15 |
| E | ∞ | 6 | ∞ | 15 | ∞ |

Directed weighted graph.




| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | ∞ | 7 | 8 | ∞ | ∞ |
| B | ∞ | ∞ | ∞ | 10 | 6 |
| C | ∞ | ∞ | ∞ | 9 | ∞ |
| D | 5 | ∞ | ∞ | 3 | 15 |
| E | ∞ | ∞ | ∞ | ∞ | ∞ |

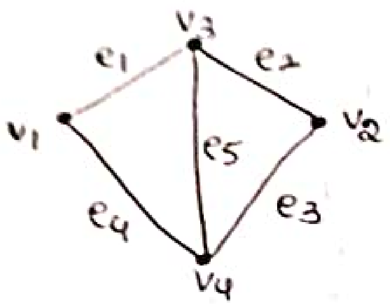
Incidence matrix :-

If an Undirected graph G consists of 'n' vertices and 'm' edges, then the incidence matrix is an $n \times m$ matrix $C = [c_{ij}]$ and defined by

$$c_{ij} = \begin{cases} 1, & \text{if the vertex } v_i \text{ incident by edge } e_j \\ 0, & \text{otherwise} \end{cases}$$

NOTE :- The number of ones in an incidence matrix of the undirected graph (without loops) is equal to the sum of the degrees of all the vertices in a graph.

ex :- Find the incidence matrix $M_I$




The undirected graph consists of 4 vertices & 5 edges.

Therefore, the incidence matrix is an 4×5 matrix.

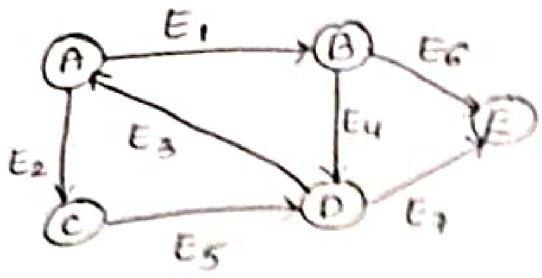
$$M_I = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

- If a directed graph G consists of n vertices & m edges, then the incidence matrix is an $n \times m$ matrix $C = [c_{ij}]$ and defined by

$$c_{ij} = \begin{bmatrix} 1, & \text{if } V_i \text{ is the initial vertex of edge } e_j \\ -1, & \text{if } V_i \text{ is the final vertex of edge } e_j \\ 0, & \text{if } V_i \text{ is not incident on edge } e_j \end{bmatrix}$$

NOTE :- The number of ones in an incidence matrix is equal to the number of edges in the graph.

ex:-

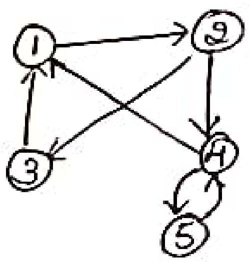


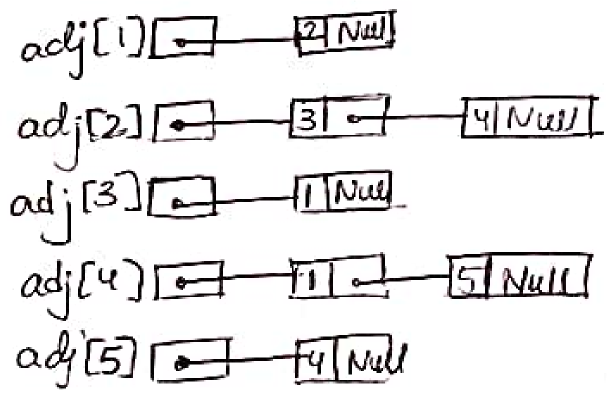

| | $E_1$ | $E_2$ | $E_3$ | $E_4$ | $E_5$ | $E_6$ | $E_7$ |
|---|---|---|---|---|---|---|---|
| A | 1 | 1 | -1 | 0 | 0 | 0 | 0 |
| B | -1 | 0 | 0 | 1 | 0 | 1 | 0 |
| C | 0 | -1 | 0 | 0 | 1 | 0 | 0 |
| D | 0 | 0 | 1 | -1 | -1 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 | -1 | -1 |

## Adjacency list

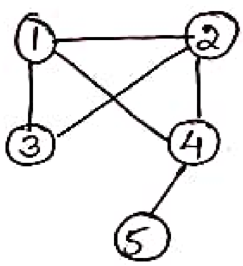
In adjacency list representation, the adjacency information is stored in the form of linked list.

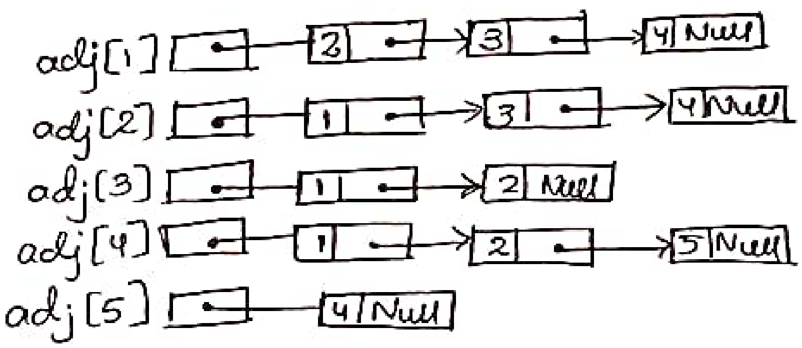
(a) Directed graph




Adjacency list




Adjacency list.

# GRAPH ISOMORPHISM

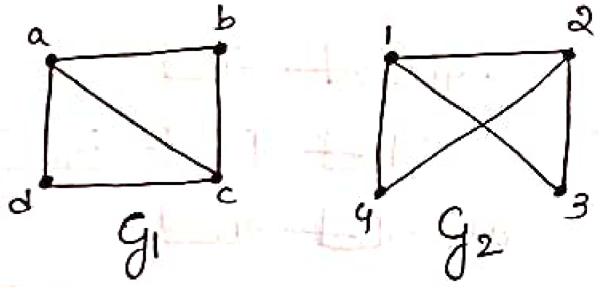
Two graphs G and H are isomorphic if there is a one-to-one correspondance between their vertices such that two vertices are adjacent in G if and only if their corresponding vertices are adjacent in H.

## Graph isomorphism conditions.

For any two graphs $G_1$ and $G_2$ to be isomorphic, necessary conditions must be satisfied.

1. Number of vertices in both the graphs must be same.
2. Number of edges in both the graphs must be same.
3. Degree sequence of both the graphs must be same.
4. Mapping should exist.

ex:-




Condition 1:- No. of vertices in $G_1 = 4$
No. of vertices in $G_2 = 4$

Condition 2:- No. of edges in $G_1 = 5$
No. of edges in $G_2 = 5$

Condition 3:- Degree Sequence in $G_1 = \{3, 3, 2, 2\}$
Degree Sequence in $G_2 = \{3, 3, 2, 2\}$

Condition 4 :- Mapping.

| $G_1$ | | $G_2$ | |
|---|---|---|---|
| vertices | Degree of adjacent vertices | vertices | Degree of adjacent vertices |
| a | {3, 2, 2} | 1 | {3, 2, 2} |
| b | {3, 3} | 2 | {3, 2, 2} |
| c | {3, 2, 2} | 3 | {3, 3} |
| d | {3, 3} | 4 | {3, 3} |

a = 1

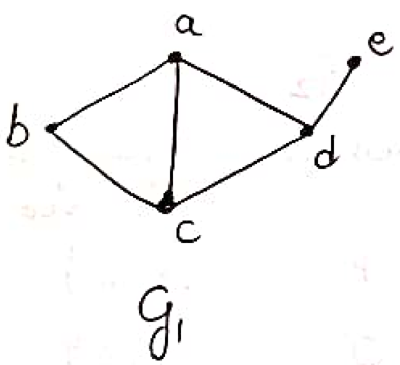
b = 3

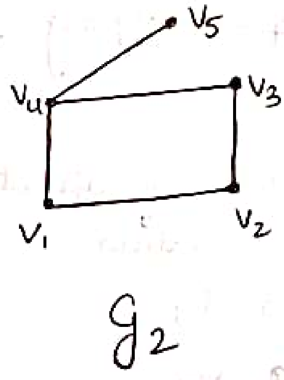
c = 2

d = 4

$\therefore$ $G_1$ & $G_2$ are isomorphic.

ex:- Verify whether two graphs are isomorphic or not.




Condition 1:- No. of vertices in $G_1$ = 5

No. of vertices in $G_2$ = 5

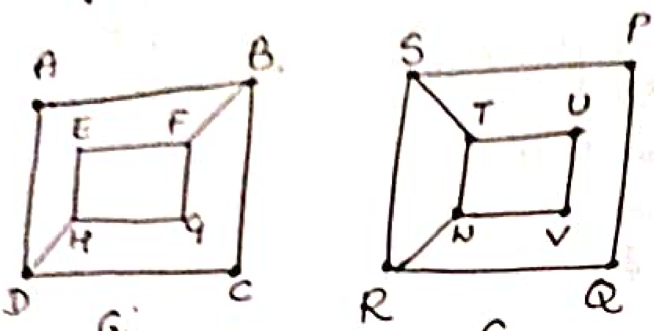
Condition 2:- No. of edges in $G_1$ = 6

No. of edges in $G_2$ = 5

Condition 2 violates.

$\therefore$ $G_1$ and $G_2$ are not isomorphic.

ex:: Examine whether the following graphs are isomorphic (or) not.




$G_1$ $G_2$

Condition 1:- No. of vertices in $G_1 = 8$

No. of vertices in $G_2 = 8$

Condition 2: No. of edges in $G_1 = 10$

No. of edges in $G_2 = 10$.

Condition 3: Degree sequence in $G_1 = \{3,3,3,3,2,2,2,2\}$

Degree sequence in $G_2 = \{3,3,3,3,2,2,2,2\}$

Condition 4:- mapping.

$G_1$

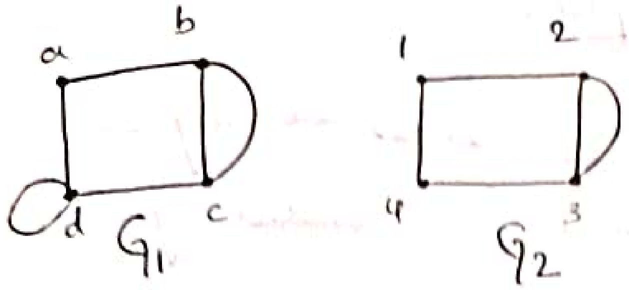
| vertices | Degree of adjacent vertices |
|---|---|
| A | {3, 3} |
| B | {2, 2} |
| C | {3, 3} |
| D | {3, 2, 2} |
| E | {3, 3} |
| F | {3, 2, 2} |
| G | {3, 3} |
| H | {3, 2, 2} |

$G_2$

| Vertices | Degree of adjacent vertices |
|---|---|
| P | {3, 2} |
| Q | {3, 2} |
| R | {3, 3, 2} |
| S | {3, 3, 2} |
| T | {3, 3, 2} |
| U | {3, 2} |
| V | {3, 2} |
| W | {3, 3, 2} |

∴ $G_1$ and $G_2$ are not isomorphic.

ex:-



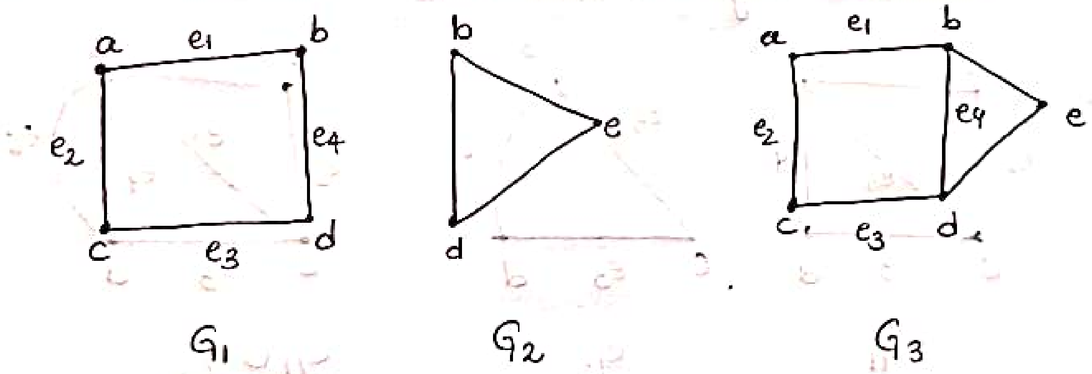

$G_1$ and $G_2$ are not isomorphic, because no. of edges in $G_1$ is 6 and no. of edges in $G_2$ is 5.

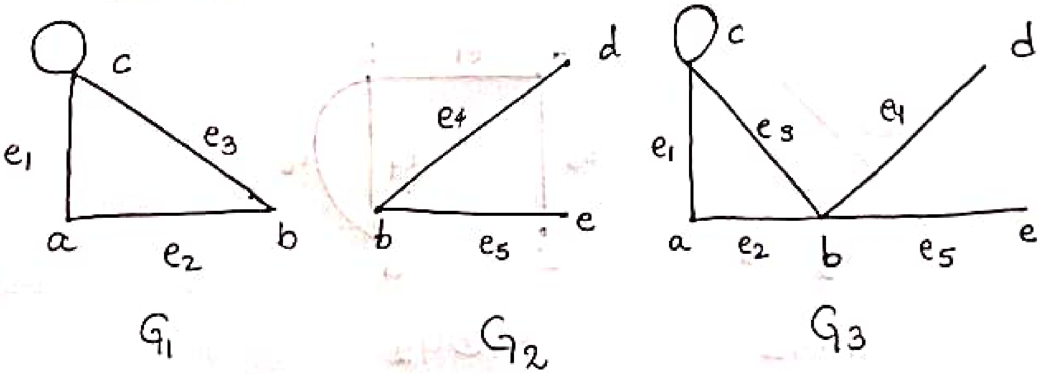
## Operations on Graphs

1. **Union of two graphs :-**

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are any two graphs. Then union of $G_1$ and $G_2$ is denoted by $G_1 \cup G_2 = G_3$, whose vertex set is $V_3 = V_1 \cup V_2$ and edge set is $E_3 = E_1 \cup E_2$

ex:-

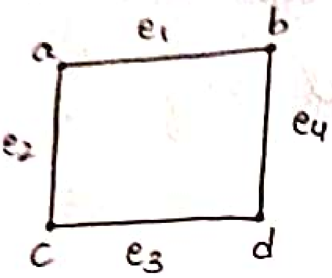


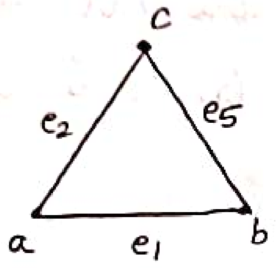
## 2. Intersection of two graphs :-

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are any two graphs. Then intersection of $G_1$ and $G_2$ is denoted by $G_1 \cap G_2 = G_3$, whose vertex set is $V_3 = V_1 \cap V_2$ and edge set is $E_3 = E_1 \cap E_2$

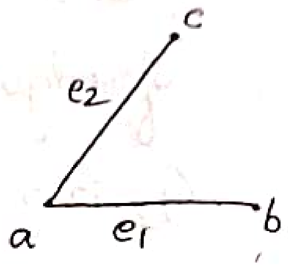
ex:-




$G_1$




$G_2$

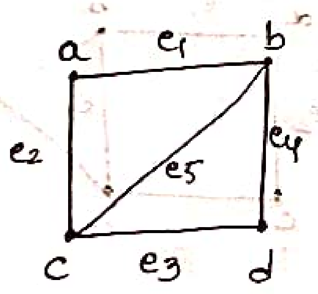



$G_3$.

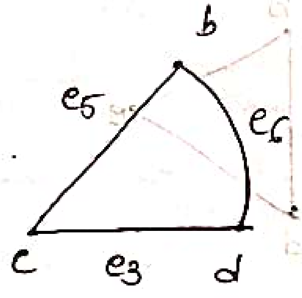
## 3. Ring Sum of two graphs :-

If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are any two graphs then their ring sum $G_3 = G_1 \oplus G_2$ is a graph consisting of the vertex set $(V_1 \cup V_2)$ and edges that are either in $G_1$ or in $G_2$ but not both.

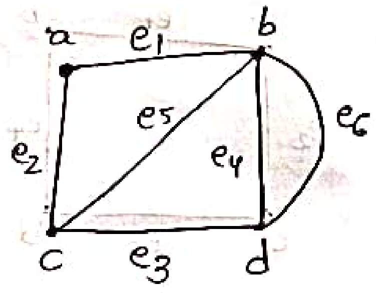
(Common edges in $G_1$ and $G_2$ should not be in $G_3$)



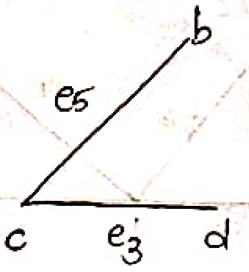

$G_1$



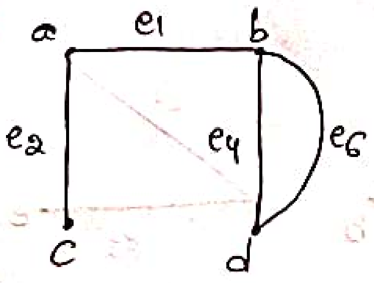

$G_2$




$G_1 \cup G_2$




$G_1 \cap G_2$



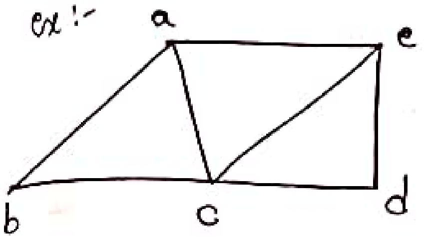

$G_1 \oplus G_2$

## 4. Decomposition of a graph :-

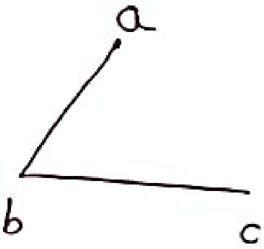
A graph is said to be decomposed into two subgraphs g and h, if $g \cup h = G$ and $g \cap h = \phi$.

i.e., each edge of G occurs either in g or in h but not in both. But some vertex may occur in both g and h.

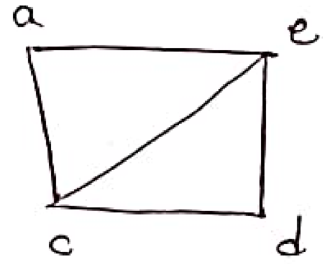
ex :-




G




g

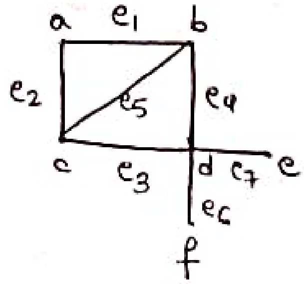



h

## 5. Deletion :-

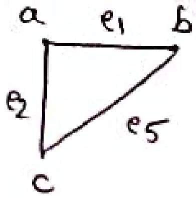
If $v_i$ is a vertex in Graph G then $G - v_i$ denotes a subgraph of G obtained by deleting $v_i$ from G. Deletion of a vertex implies deletion of all edges incident on that vertex.

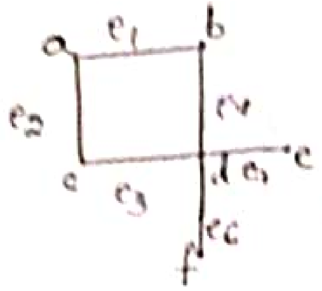
ex :-




After deleting vertex d.

The subgraph of G i.e. $G - d$ is



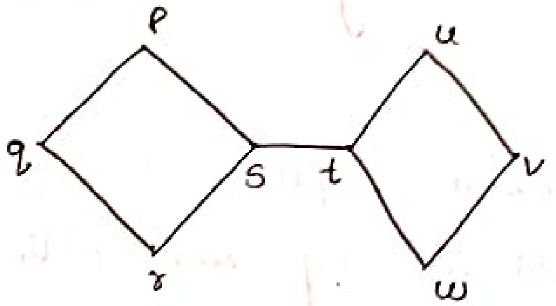



## Cut vertex

Let 'G' be any connected graph, a vertex $V \in G$ is called a cut vertex of G if $(G-V)$ results in a disconnected graph.

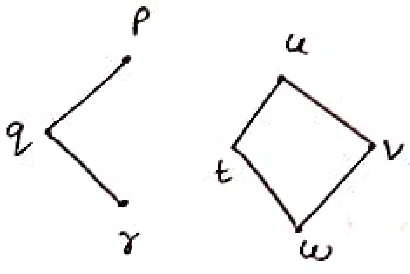
Note:- A connected graph G may have maximum $(n-2)$ cut vertices.

ex:-




If we remove vertex s, the graph becomes disconnected graph

∴ s is a cut vertex.

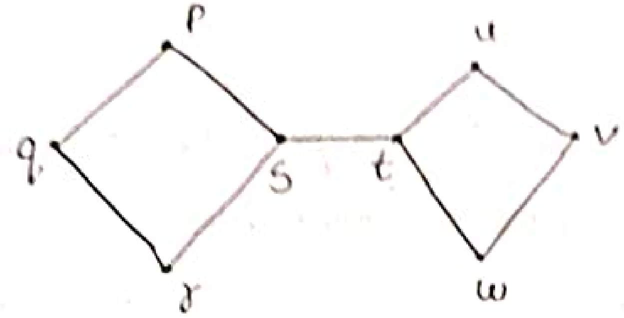



## Cut Edge (Cut Bridge)

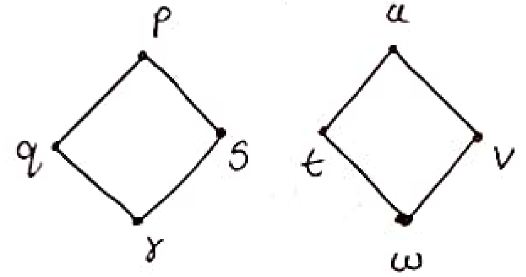
Let 'G' be any connected graph, an edge 'e' of graph G is said to be 'cut edge' of G if $(G-e)$ results in a disconnected graph.

Note:- A connected graph G may have maximum $(n-1)$ cut edges

ex:-



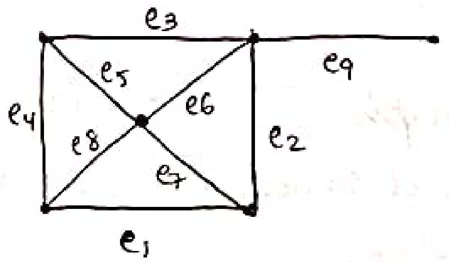

Edge S to T is cut edge which makes the connected graph in to disconnected by splitting the graph into two graphs.




Cut set of a graph :-

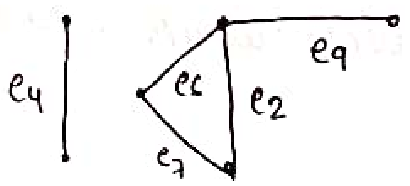
Let G = (V, E) be a connected graph. A subset E' of E is called a cut set of G if deletion of all the edges of E' from G makes G disconnect.

ex:-




Its cut set is $E_1 = \{e_1, e_3, e_5, e_8\}$

after removing the cutset $E_1$ from the graph,

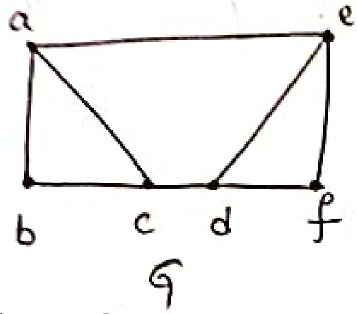


## Vertex connectivity

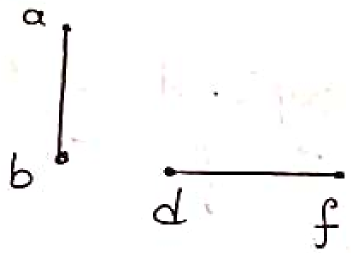
Let G be any connected graph. Then the minimum number of vertices whose removal results in a disconnected graph is known as vertex connectivity.

It is denoted by $K(G)$ (Kappa)

ex:-




Removing vertex e & c results in a disconnected graph.




$\therefore K(G) = 2$

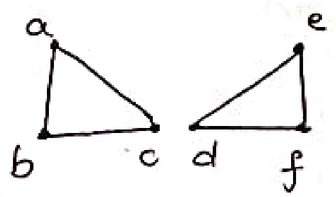
## Edge connectivity

Let G be any connected graph. Then the minimum number of edges whose removal results in a disconnected graph is called edge connectivity.

It is denoted by $\lambda(G)$ (Lamba)

ex:- In the above graph G,

removing edge a to e & c to d results in disconnected graph.

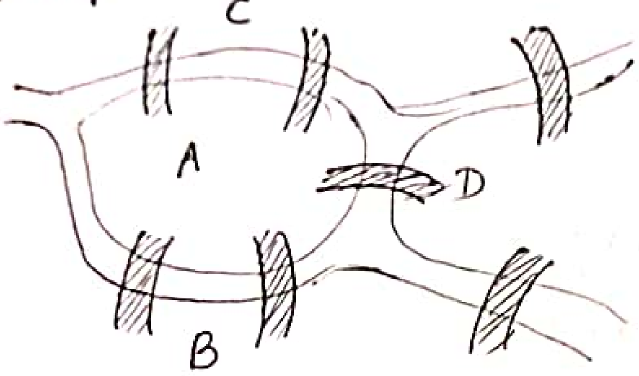



$\therefore \lambda(G) = 2.$

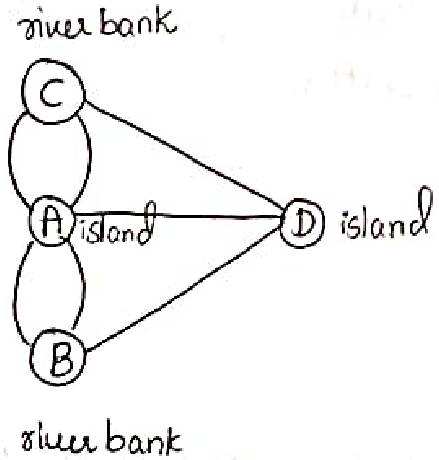
# Euler and Hamiltonian path.

## Konigsberg Bridge Problem :-

The town of Konigsberg, was divided into four sections by the branches of the Pregel river. These four sections included the two regions on the banks of the Pregel, and the region between the two branches of the Pregel.




Seven bridges of Konigsberg.




On this graph, Vertices represent the landmasses and Edges represent the bridges.

Konigsberg Bridge Problem may be stated as -

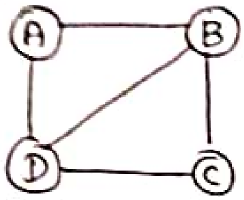
"Starting from any of the four land areas A, B, C, D, is it possible to cross each of the seven bridges exactly once and come back to the starting point without swimming across the river?".

In 1735, A Swiss Mathematician Leonhard Euler solved this problem & provided a solution. He finally concluded that, such a walk is not possible.

## Euler Path

Eulerian Path is a path in graph that visits every edge exactly once with or without repeating the vertices.

ex:-



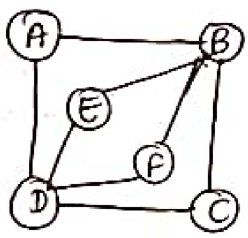

BCDBAD - Euler path

Euler circuit does not exist.

## Euler circuit

Euler circuit is an Eulerian path which starts and ends on the same vertex.

ex:-

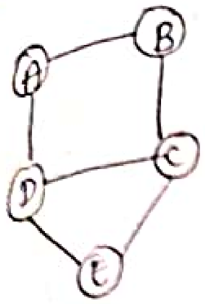



BCDFBEDAB - Euler path & Euler circuit.

## Hamiltonian path

* If there exists a path in the connected graph that visits every vertex of the graph exactly once without repeating the edges, then such a path is called as a Hamiltonian path.

* Hamiltonian circuit is a Hamiltonian path which starts and ends on the same vertex.




ABCDE - Hamiltonian Path

ABCEDA - Hamiltonian circuit.

NOTE :- • For any planar graph with 'v' vertices, 'e' edges and 'r' regions, we have $v - e + r = 2$.

• If a connected planar graph G has 'e' edges & 'v' vertices then, $3v - e \geq 6$.

Q:- What is the number of regions in a connected planar simple graph with 20 vertices each with a degree of 3?

Sol:- Given, No. of vertices = 20.

Degree of each vertex = 3

Sum of degree of vertices = $20 \times 3 = 60$

By, handshaking lemma we have,

Sum of degree of vertices = $2 \times$ No. of edges.

$$60 = 2e$$

$$\boxed{e = 30}$$

$$\therefore r = 2 - v + e$$

$$r = 2 - 20 + 30$$

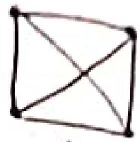
$$r = 2 + 10$$

$r = 12$ regions.

* Show that complete graph $K_4$ is planar.

Sol:- Complete graph $K_4$ contains 4 vertices and 6 edges.

WKT, For a connected planar graph



$K_4$

$$3v - e \geq 6$$

Hence, for $K_4$

$$3(4) - 6 = 6$$

which satisfies the property

$\therefore K_4$ is planar

Hence proved.

* Show that $K_5$ is non-planar.

Sol:- The complete graph $K_5$ contains 5 vertices and 10 edges.

WKT, for a connected planar graph $3v - e \geq 6$.

Hence, for $K_5$

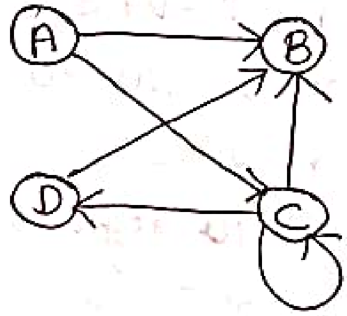
$3(5) - 10 = 5 \ngeq 6$, which does not satisfies the property.

Thus, $K_5$ is non-planar graph.

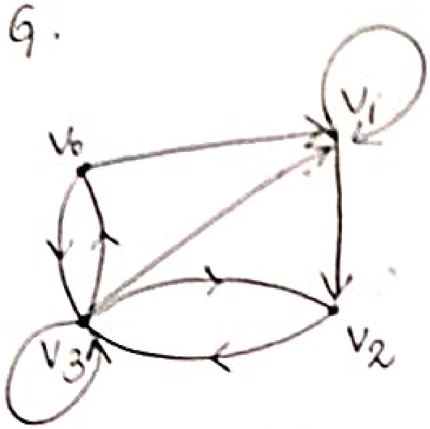
* Draw the digraph G corresponding to the matrix.

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 |
| B | 0 | 0 | 0 | 0 |
| C | 0 | 1 | 1 | 1 |
| D | 0 | 1 | 0 | 0 |

Sol:-



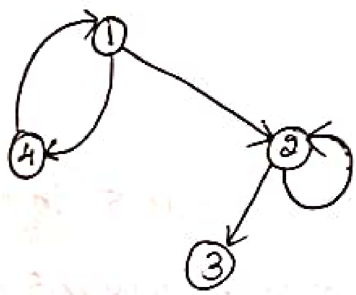
Construct the matrix corresponding to the digraph G.




Sol:-

$$M = \begin{array}{c} \\ V_0 \\ V_1 \\ V_2 \\ V_3 \end{array}\begin{array}{c} \begin{array}{cccc} V_0 & V_1 & V_2 & V_3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

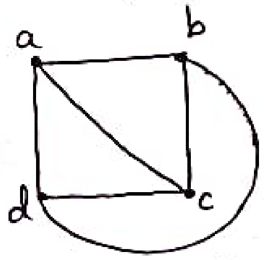
- Find the relation represented by the digraph given below. Write down its matrix.




$$R = \{(1,2),(1,4),(2,2),(2,3),(4,1)\}$$

$$M = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

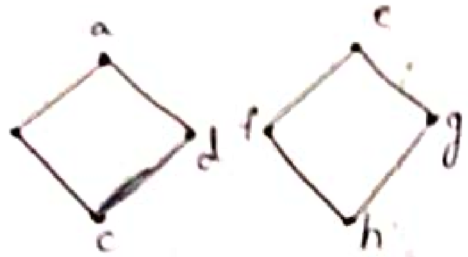
- Write the planar representation of $K_4$




$K_4$

* Draw a disconnected graph with 8 vertices and 2 components.

Sol:-

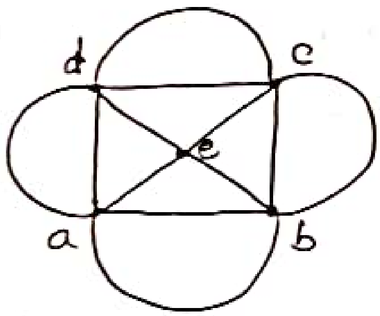



NOTE :-

- A graph will contain an Euler path iff it contains atmost two vertices of odd degree.

* Show that the graph G shown below is not Eulerian.

G:




Sol$^n$:- There are four vertices of degree 5 in the graph, a, b, c, d are the vertices degree 5 in G.

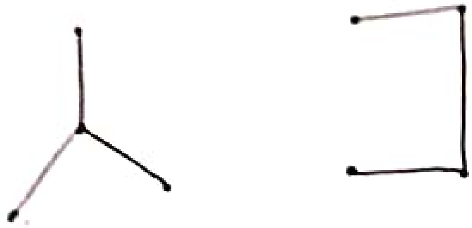
∴ G is not Eulerian.

## Introduction to Trees.

- A tree is a connected acyclic undirected graph
- The graph G is called a tree if G is connected and contains no cycle.

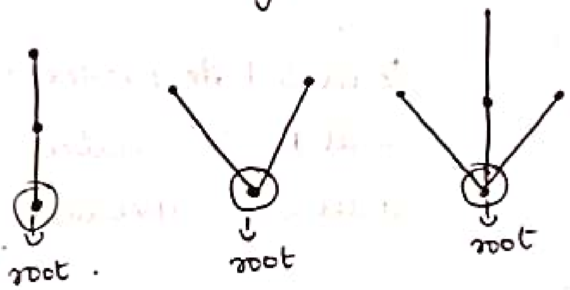
Tree is denoted by symbol T.

ex:-



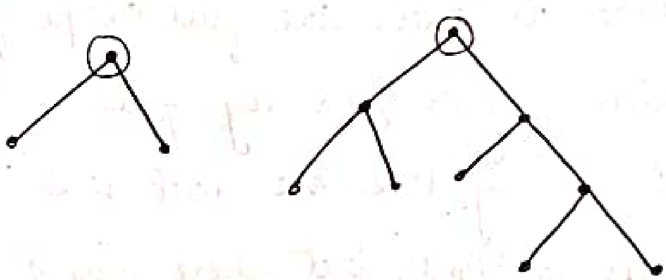
**Rooted tree :-** A rooted tree is a tree in which one vertex is designated as the root

ex:-

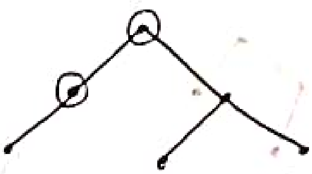



**Binary tree :-** A binary tree is defined as a tree in which there is exactly one vertex of degree two, and each of the remaining vertices is of degree one or three. The vertex of degree two serves as a root.

ex:-



Binary Trees

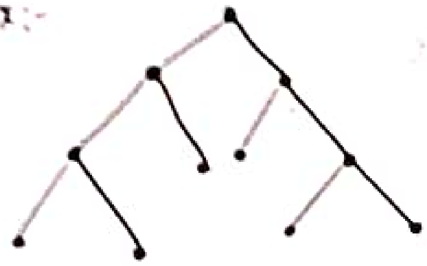


It is not a Binary tree

## Strictly Binary tree :-

A strictly binary tree is a type of binary tree where each vertex has either zero or two children

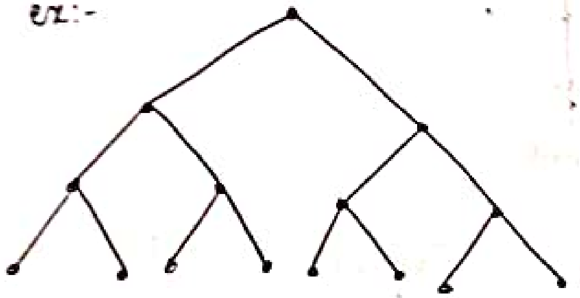
ex:-



## Full or Complete Binary Tree

Full Binary tree is a tree that contains maximum number of possible number of vertices in all levels.

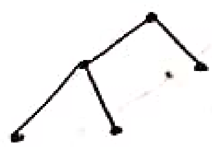
ex:-

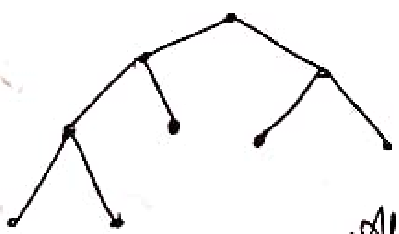


level 0 have 1 vertex
level 1 2 vertices
level 2 4 vertices
⋮
level n $2^{n-1}$ vertices.

## Almost full Binary Tree.

An almost complete binary tree is a type of binary tree where all levels are full except for the last level, which may be partially filled. In an almost complete binary tree, the last level is filled from left to right, but there may be empty spaces on the right.





Almost Binary Tree.

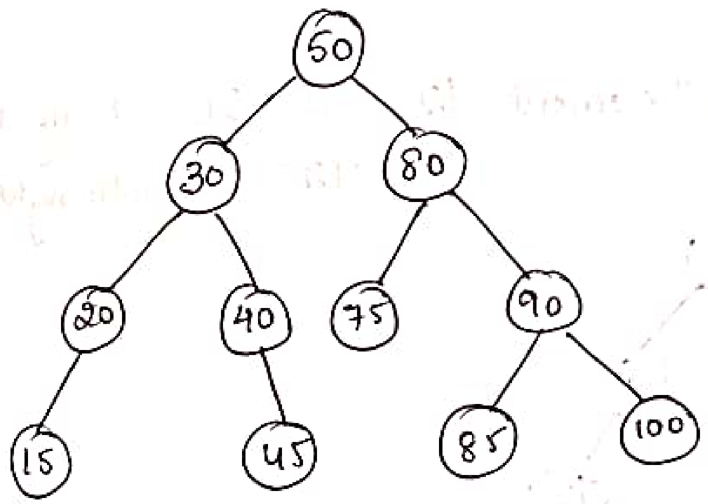
## Important properties

1. A simple non-directed graph G is a tree if and only if G is connected and has no cycles.
2. A tree T with n vertices has exactly (n-1) edges.
3. A graph G is a tree if and only if G has no cycles and $|E| = |V| - 1$
4. Every non-trivial tree has atleast 2 vertices of degree 1.

## Binary Search tree

The binary search tree is a binary tree in which each vertex has value greater than every vertex of left subtree and less than every vertex of right subtree.

ex:-

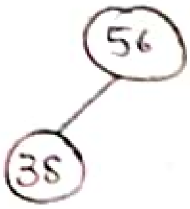



## Constructing a Binary search tree (or) Inserting

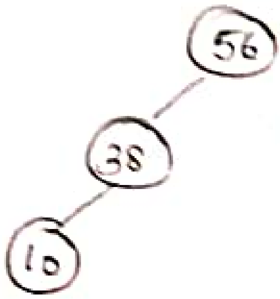
ex:- 56, 38, 10, 65, 72, 44, 50.

Step1:- First element is the root

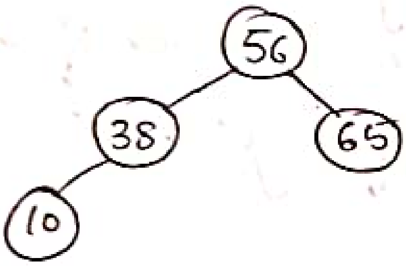

Step2:- Take second element $38 < 56$, go to left.



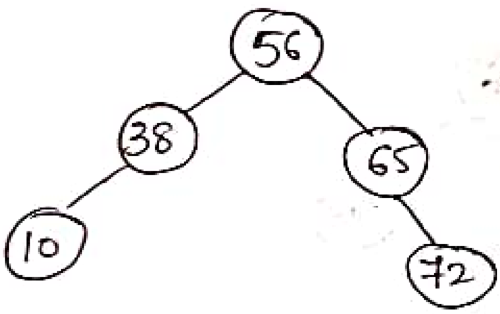

Step3:- Take 3rd element 10 . $10 < 56$ , go to left of 56
$10 < 38$ , go to left of 38



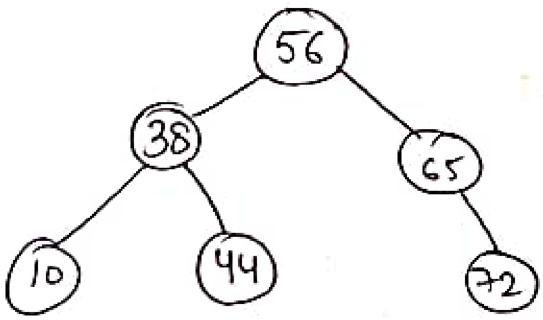

Step4:- Take 4th element 65 , $65 > 56$ , goto right 56




Step5:- Take 5th element 72, $72 > 56$ goto right of 56
$72 > 65$ goto right of 56.




Step6:- Take 6th element 44, $44 < 56$ goto left of 56
$44 > 38$ goto right of 38

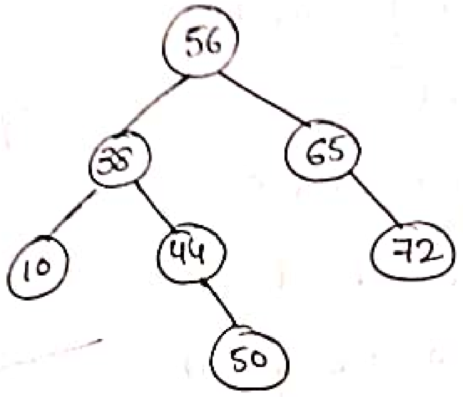


Step 7:- Take 7th element 50,

$50 < 56$, goto left of 56

$50 > 38$, goto right of 38.

$44 > 38$, goto right of 44.



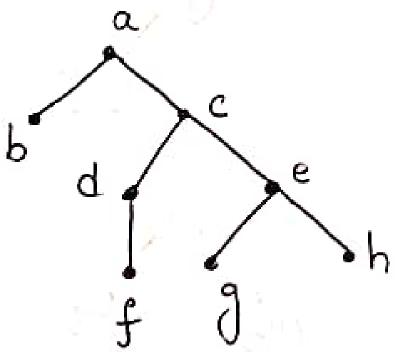

Binary Search Tree

Forest:- A forest is a graph that is acyclic and is made up of one or more trees.

A forest is a collection of disjoint trees. Each connected component of a forest is a tree.

Tree Forest

Q. Consider the rooted tree,




(a) What is the root of T?

A:- Vertex 'a'

(b) Find the leaves & the internal vertices of T.

A:- Leaves are those vertices that have no children

→ b, f, g, h

The internal vertices are c, d and e.

(c) What are the levels of c and e.

A:- The levels of c and e are 1 and 2 respectively.

(d) Find the children of c and e.

A: The children of c are d & e

The children of e are g & h.

(e). Find the descendants of the vertices a and c.

A:- Descendants are those vertices that have a and c as an ancestor.

The descendant of a are :- b, c, d, e, f, g, h.

The descendant of c are :- d, e, f, g, h.

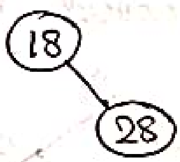
Q. Construct a binary search tree for below elements.

18, 28, 10, 55, 82, 37, 40, 92, 66.

Sol:- Step 1:-

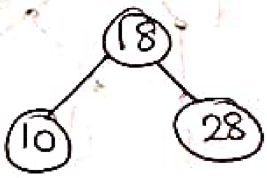

Step 2:-

28 > 18 → right.




Step 3:-

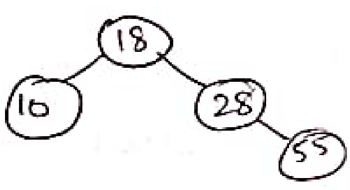
10 < 18 → left.




Step 4:-

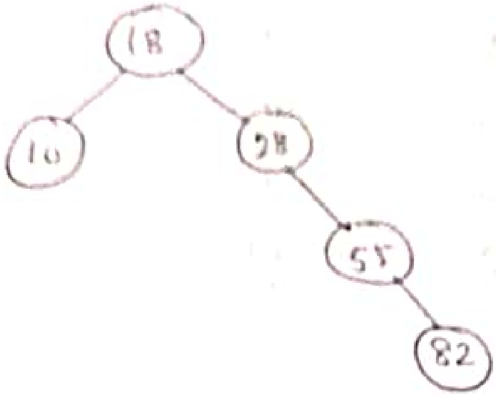
55 > 18 → right

55 > 28 → right.



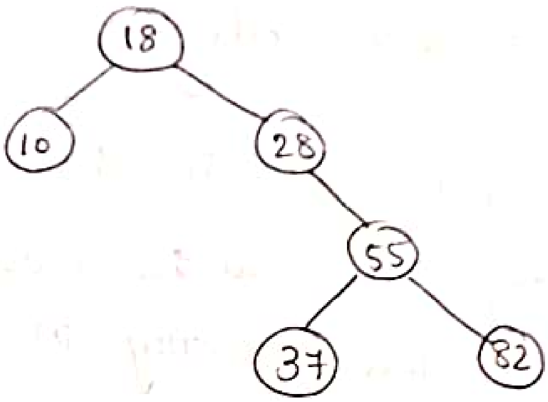
Step 5:-

82 > 18 → right
82 > 28 → right
82 > 55 → right



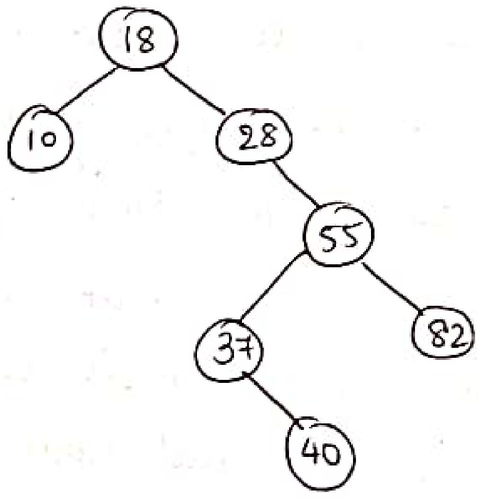

Step 6:-

37 > 18 → right
37 > 28 → right
37 < 55 → left.



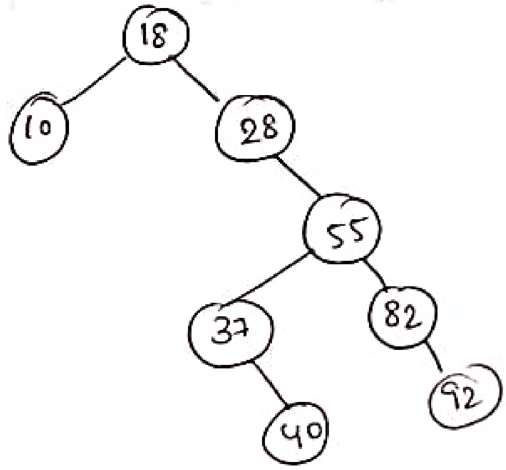

Step 7:-

40 > 18 → right
40 > 28 → right
40 < 55 → left
40 > 37 → right.



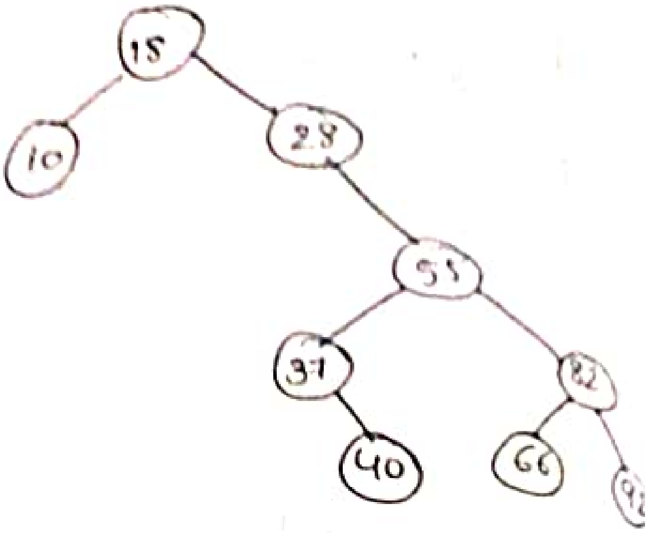

Step 8:-

92 > 18 → right
92 > 28 → right
92 > 55 → right
92 > 82 → right.



Step 9:-

66>18 → right
66>28 → right
66>55 → right
66<82 → left




## Huffman coding.

Step-1 :- Calculate the frequency of each string.

Step 2 :- Sort all the characters on the basis of their frequency in ascending order.

Step-3 :- Mark each unique character as a leaf node.

Step 4 :- Create a new internal node

Step 5 :- The frequency of the new code as the Sum of the single leaf node.

Step 6 :- Mark the first node as this left child and another node as the right child of the recently created node.

Step 7 :- Repeat all the steps from Step 2 to step 6.

String

BCAADDDCCACACAC

Each character occupies 8 bits.

There are a total of 15 characters.

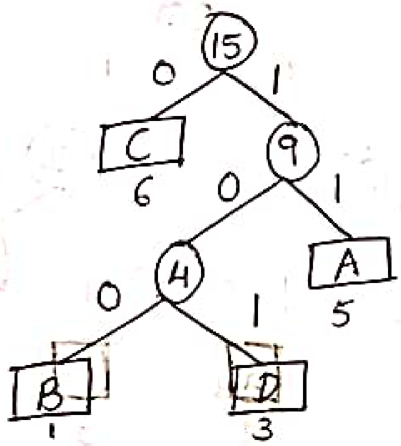
Thus, a total 8 x 15 = 120 bits.

(1) frequencies

| 1 | 6 | 5 | 3 |
|---|---|---|---|
| B | C | A | D |

(2) Arranging in increasing order

| 1 | 3 | 5 | 6 |
|---|---|---|---|
| B | D | A | C |




| character | frequency | Code | Size |
|---|---|---|---|
| A | 5 | 11 | 5 x 2 = 10 |
| B | 1 | 100 | 1 x 3 = 3 |
| C | 6 | 0 | 6 x 1 = 6 |
| D | 4 | 101 | 3 x 3 = 9 |
| 4 x 8 = 32 bits | 15 bits | | 28 bits. |

Without encoding, the total size of the string was 120 bits. After encoding the size is reduced to 32 + 15 + 28 = 75

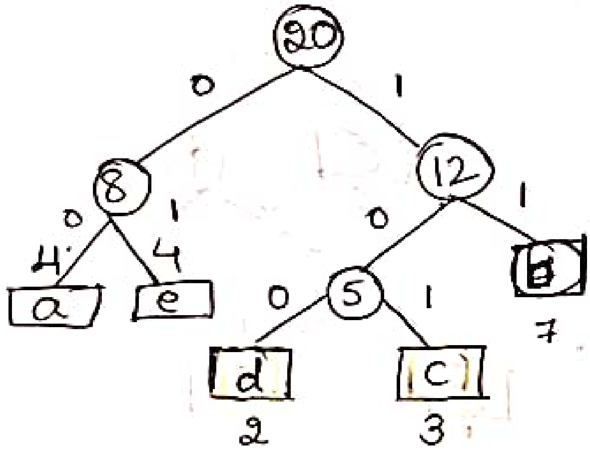
Q.

| Character | Frequency |
|---|---|
| a | 4 |
| b | 7 |
| c | 3 |
| d | 2 |
| e | 4 |

20×8 = 160 bits

Arrange in increasing order

| | | | | |
|---|---|---|---|---|
| 2 | 3 | 4 | 4 | 7 |
| d | c | a | e | b |

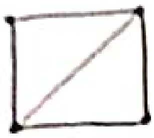



| | | |
|---|---|---|
| a - 4 | 00 | 2×4 = 8 |
| b - 7 | 11 | 2×7 = 14 |
| c - 3 | 101 | 3×3 = 9. |
| d - 2 | 100 | 3×2 = 6 |
| e - 4 | 01 | 2×4 = 8 |
| | | 45 |

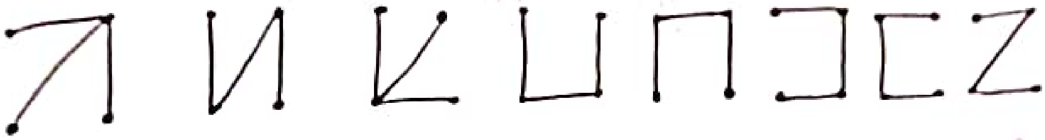
5×8 + 20 bits
= 40 bits

Size is reduced to 40+20+45 = 105 bits

Spanning Tree :- If G is any connected graph, a spanning tree in G is a subgraph T of G, which is a tree.

ex:- 

G



Spanning Trees

**Important properties**

1. Every connected graph has a spanning tree.
2. Any two spanning trees for a graph have the same number of edges.
3. A non-directed graph G is connected if and only if G contains a spanning tree.

**Minimum Spanning Tree.**

A Minimum spanning tree for a weighted graph is a spanning tree with minimum weight.

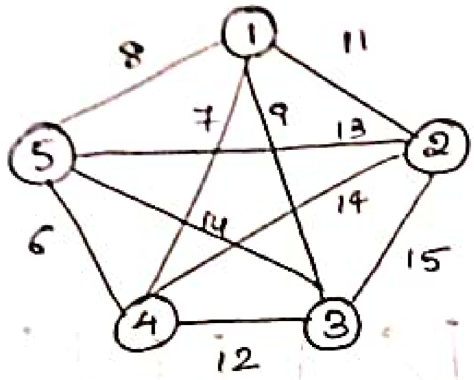
To find minimum spanning tree two algorithms are used :-

(1) Kruskal's Algorithm

(2) Prim's Algorithm.

PRIM'S ALGORITHM.

1. Find the minimum weight spanning tree by PRIM'S Algorithm.



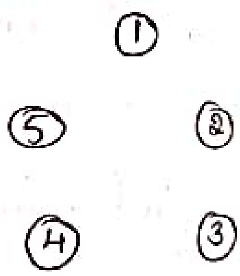

Sol:- Cost Adjacency matrix.

| C | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | - | 11 | 9 | 7 | 8 |
| 2 | 11 | - | 15 | 14 | 13 |
| 3 | 9 | 15 | - | 12 | 14 |
| 4 | 7 | 14 | 12 | - | 6 |
| 5 | 8 | 13 | 14 | 6 | - |

Denote the unvisited vertices by G and visited vertices by S.

Step 1:- S = 0 (no vertices are visited)

G = {1, 2, 3, 4, 5}




Step 2:- S = 1

G = {2, 3, 4, 5}

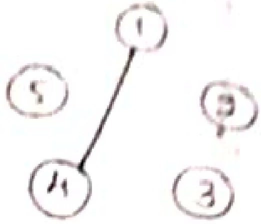
Looking at the adjacency matrix all the edges from S i.e. (1), the minimum edge is to be determined.

= min { <1,2>, <1,3>, <1,4>, <1,5> }

= min { 11, 9, 7, 8 }

= 7

So, draw an edge from 1 → 4




Step 3:- S = {1, 4}

G = {2, 3, 5}

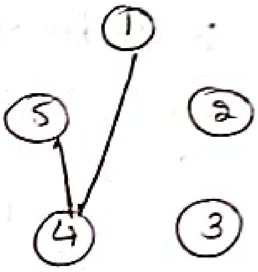
Looking at the adjacency matrix all the edges from S i.e., (1) and (4) to the unvisited vertices i.e., G and determining the minimum edge vertex

min { <1,2>, <1,3>, <1,5>, <4,2>, <4,3>, <4,5> }

= min { 11, 9, 8, 14, 12, 6 }

= 6

So, draw an edge from 4 → 5




Step 4:- S = {1, 4, 5}

G = {2, 3}

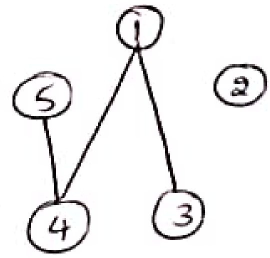
Looking at the adjacency matrix all the edges from S i.e., (1), (4) and (5) to the unvisited vertices i.e. G & determining the minimum edge vertex.

min { <1,2>, <1,3>, <4,2>, <4,3>, <5,2>, <5,3> }

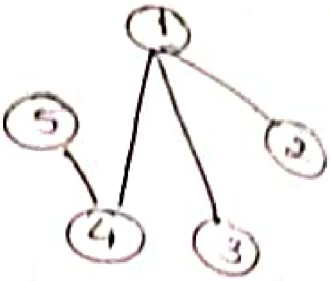
= min { 11, 9, 14, 12, 13, 14 }

= 9

So, draw an edge from 1 → 3.



Step 5:- Now $S = \{1,3,4,5\}$
$Q = \{2\}$



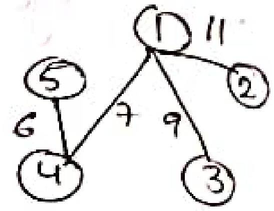

Looking at the adjacency matrix all the edges from S to Q

$\min\{<1,2>, <3,2>, <4,2>, <5,2>\}$

$= \min\{11, 15, 14, 13\}$

$= 11$

So draw an edge from $1 \rightarrow 2$

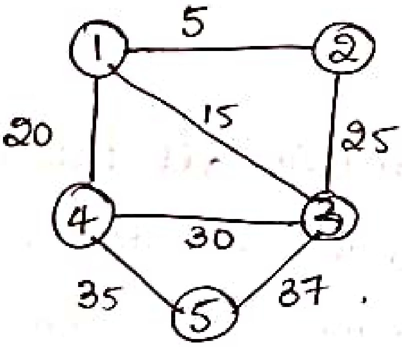



Step 6:- Now $S = \{1,2,3,4,5\}$
$Q = \text{Null}$

As there are no unvisited vertices, the algorithm is to be stopped and the total cost is determined by adding all the costs of final illustration

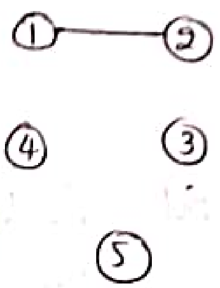
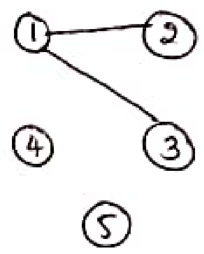
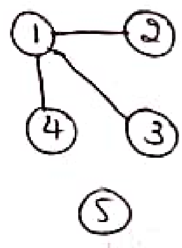
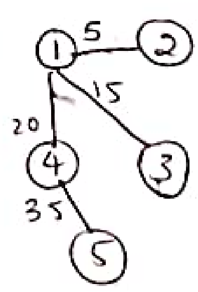
$\therefore$ Total cost $= 6+7+9+11 = 33$.

2.




Sol:- Cost Adjacency matrix

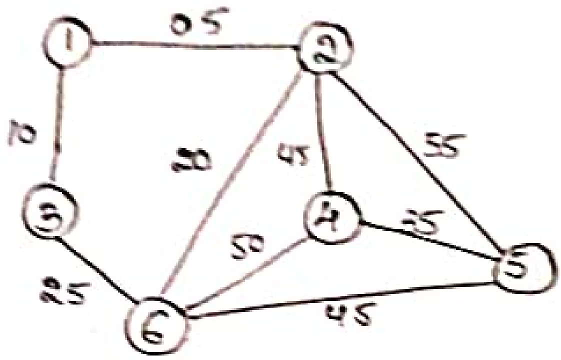
| C | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | - | 5 | 15 | 20 | ∞ |
| 2 | 5 | - | 25 | ∞ | ∞ |
| 3 | 15 | 25 | - | 30 | 37 |
| 4 | 20 | ∞ | 30 | - | 35 |
| 5 | ∞ | ∞ | 37 | 35 | - |

| | Visited vertices | Unvisited vertices | Minimum edges from S to G | Illustration |
|---|---|---|---|---|
| 1 | S = 0 | G = {1,2,3,4,5} | – |  |
| 2 | S = {1} | G = {2,3,4,5} | $\min\{\langle 1,2\rangle, \langle 1,3\rangle, \langle 1,4\rangle, \langle 1,5\rangle\}$ $= \min\{5, 15, 20, \infty\}$ $= 5$ |  |
| 3 | S = {1,2} | G = {3,4,5} | $\min\{\langle 1,3\rangle, \langle 1,4\rangle, \langle 1,5\rangle, \langle 2,3\rangle, \langle 2,4\rangle, \langle 2,5\rangle\}$ $= \min\{15, 20, \infty, 25, \infty, \infty\}$ $= 15$ |  |
| 4 | S = {1,2,3} | G = {4,5} | $\min\{\langle 1,4\rangle, \langle 1,5\rangle, \langle 2,4\rangle, \langle 2,5\rangle, \langle 3,4\rangle, \langle 3,5\rangle\}$ $= \min\{20, \infty, \infty, \infty, 30, 37\}$ $= 20$ |  |
| 5 | S = {1,2,3,4} | G = {5} | $\min\{\langle 1,5\rangle, \langle 2,5\rangle, \langle 3,5\rangle, \langle 4,5\rangle\}$ $= \min\{\infty, \infty, 37, 35\}$ $= 35$ |  |
| 6 | S = {1,2,3,4,5} | G = null | | |

Total cost
= 5+15+20+35
= 75.

# Kruskal's Algorithm:

1. Find minimum weight spanning tree by Kruskal's algorithm. ( )

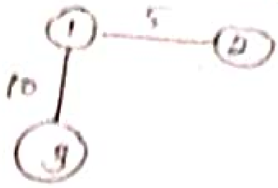



Sol:- Firstly, the edges are to be arranged in the increased order of their weights.

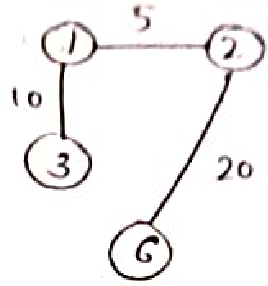
| Edge | Weight |
|---|---|
| 1-2 | 05 |
| 1-3 | 10 |
| 2-6 | 20 |
| 3-6 | 25 |
| 4-5 | 35 |
| 2-4 | 45 |
| 5-6 | 45 |
| 4-6 | 50 |
| 2-5 | 55 |

| Step | Edge Connected | Connected components | Graph |
|---|---|---|---|
| Initial | - | [1], [2], [3], [4], [5] [6] | - |
| 1 | 1-2 | [1,2] [3] [4] [5] [6]<br>Accepted because no cycle formation |  |

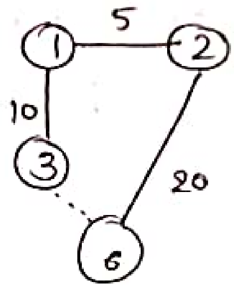
2\. 1-3 [1,2,3] [4] [5] [6]
Accepted because no cycle formation



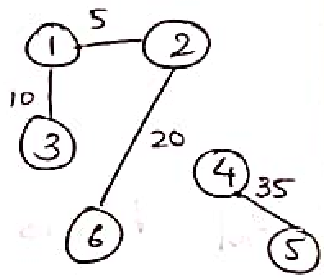

3\. 2-6 [1,2,3,6] [4] [5]
Accepted because no cycle formation



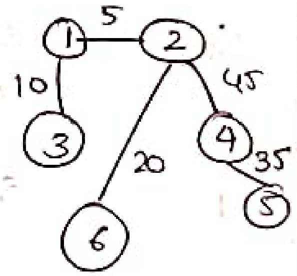

4\. 3-6 Rejected because there is a cycle formation




5\. 4-5 [1,2,3,6] [4,5]
Accepted because no cycle formation




6\. 2-4 [1,2,3,4,5,6]
Accepted because no cycle formation



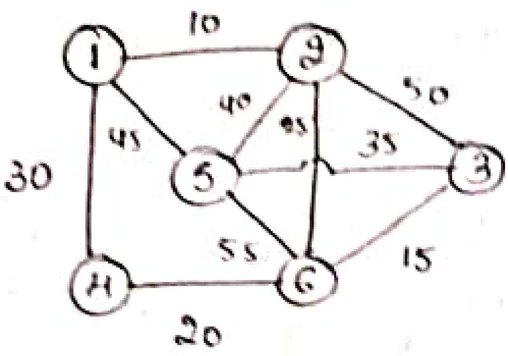

7\. 5-6 Rejected because cycle formation —

8\. 4-6 —"— —

9\. 2-5 —"— —

∴ Total weight = 5+10+20+35+45
= 115

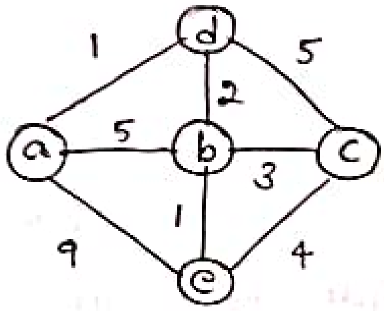
Ex 2:- Consider the following graph.




Ans :-

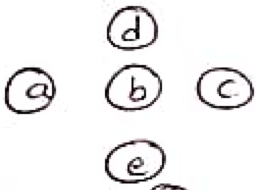
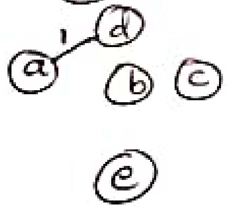
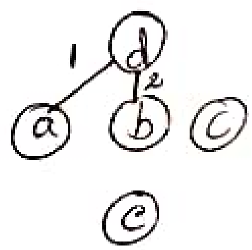
Total weight = 10+15+20+25+3
= 105.

Ex 3:- Find the minimum cost spanning tree for the below graph using Prim's and Kruskal's Algorithm.




| c | a | b | c | d | e |
|---|---|---|---|---|---|
| a | - | 5 | ∞ | 1 | 9 |
| b | 5 | - | 3 | 2 | 1 |
| c | ∞ | 3 | - | 5 | 4 |
| d | 1 | 2 | 5 | - | ∞ |
| e | 9 | 1 | 4 | ∞ | - |

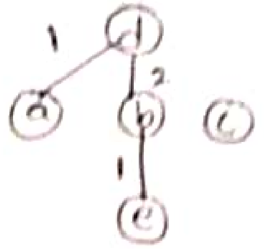
Sol:- Prim's Algorithm

| Step | Visited Vertices | Unvisited vertices | Minimum edge from S to Q | Graph |
|---|---|---|---|---|
| 1 | S=0 | Q={a,b,c,d,e} | - |  |
| 2 | S={a} | Q={b,c,d,e} | min{<a,b>,<a,c> <a,d>, <a,e>} = min{5,∞,1,9} = 1 |  |
| 3: | S={a,d} | Q={b,c,e} | min{<a,b>,<a,c> ,<a,e>,<d,b>, <d,c>, <d,e>} = min {5,∞,9,2, 5, ∞} = 2 |  |

S={a,d,b}  Q={c,e}  min{<a,c>,<a,e>, <d,c>,<d,e>, <b,c>,<b,e>}

= min{∞, 9, 5, ∞, 3, 1}

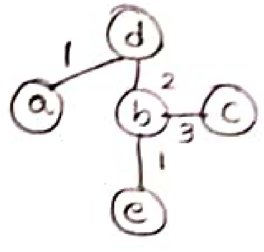
= 1




S={a,b,d,e}  Q={c}  min{<a,c>,<b,c>, <d,c>,<e,c>}

= min{∞, 3, 5, 4}

= 3

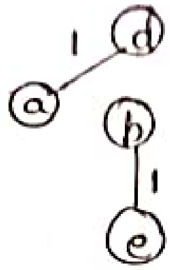
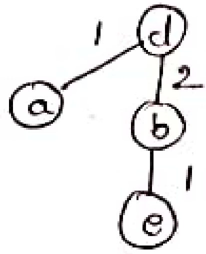
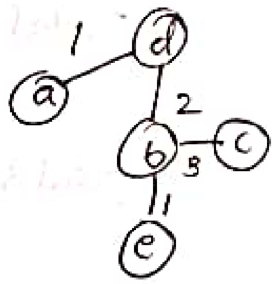



S= {a,b,c,d,e}  Q=null

Total weight = 1+2+1+3
= 7.

Kruskal's Algorithm

| Edges | Weights. |
|---|---|
| a-d | 1 |
| b-e | 1 |
| b-d | 2 |
| b-c | 3 |
| c-e | 4 |
| a-b | 5 |
| c-d | 5 |
| a-e | 9 |

| Steps | Edges | Components | Graphs |
|---|---|---|---|
| Initial | | [a] [b] [c] [d] [e] | — |
| 1 | a-d | [a,d] [b] [c] [e] Accepted because no cycle formation |  |
| 2. | b-e | [a,d] [b,e] [c] Accepted because no cycle formation |  |
| 3. | b-d | [a,b,d,e] [c] accepted because no cycle formation |  |
| 4. | b-c | [a,b,c,d,e] accepted because no cycle formation |  |
| 5 | c-e | rejected due to cycle formation | |
| 6. | a-b | rejected | |
| 7. | c-d | rejected | |
| 8. | a-e | rejected. | |

∴ Total weight = 1+2+3+1 = 7