UNIT - V Graphs & Trees (1st Part, 2nd Part)

Graph - A graph G = (V, E) consists of V, a nonempty set of vertices (or nodes) & E, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoints.

Infinite Graph - A graph with infinite vertex set.

Finite Graph - A graph with finite vertex set.

Simple graph - A graph in which each edge connects two different vertices & where no two edges connect the same pair of vertices is called a simple graph.

Multigraphs :- Graphs that may have multiple edges connecting the same vertices are called multigraphs.

Loops - edges that connect a vertex to itself.

Pseudographs - Graphs that may include loops, and possibly multiple edges connecting the same pair of vertices are called pseudographs.

Directed Graph (or digraph) (V, E) consists of a non-empty set of vertices V and a set of directed edges (or arcs) E. Each directed edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (u, v) is said to start at u and end at v.

Simple directed graph - Graph that has no loops and no multiple directed edges.

Directed Multigraphs - Directed graphs that may have multiple directed edges from a vertex to a second vertex.

Mixed Graph - A graph with both directed and undirected edges is called a mixed graph.

Graph Models

Type	Edges	Multiple edges allowed?	Loops allowed?
Simple graph	Undirected	NO	NO
Multigraph	Undirected	Yes	NO
Pseudograph	Undirected	Yes	Yes
Simple directed graph	Directed	NO	NO
Directed Multigraph	Directed	Yes	Yes
Mixed Graph	Both	Yes	Yes

Graph Terminology and Special Types of Graphs

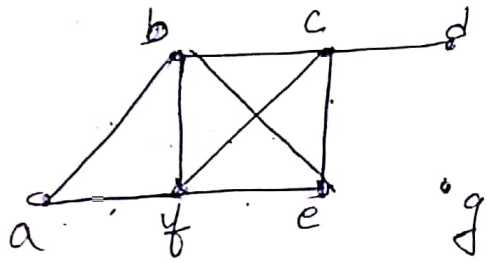
Definition - Two vertices u and v in an undirected graph G are called adjacent (or neighbours) in G if u and v are end points of an edge of G. If e is associated with (u,v), the edge e is called incident with the vertices u and v. The edge e is also said to connect u and v. The vertices u and v are called endpoints of an edge associated with {u,v}.

Definition

The degree of a vertex in an undirected graph is the no: of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. It is denoted by "deg()".

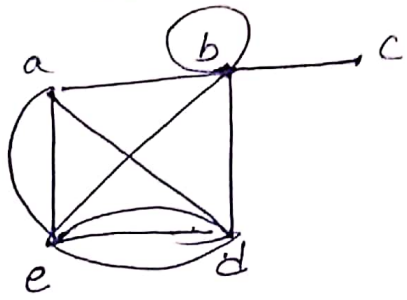
eg Find degrees of vertices.

Graph G



$\deg(a) = 2$ $\deg(e) = 3$
$\deg(b) = 3$ $\deg(f) = 4$
$\deg(c) = 4$ $\deg(g) = 0$
$\deg(d) = 1$

Graph H



$\deg(a) = 4$ $\deg(d) = 5$
$\deg(b) = 6$ $\deg(e) = 6$
$\deg(c) = 1$

Note 1) A vertex of degree zero is called isolated.

2) A vertex is pendent if and only if it has degree one.

Handshaking Theorem : Let $G = (V, E)$ be an undirected graph with e edges. Then

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

eg How many edges are there in a graph with 10 vertices each of degree 6?

$$\sum_{v \in V} \deg(v) = 10 \cdot 6 = 60$$

$$2e = 60$$

$$e = 30.$$

Theorem: An undirected graph has an even no: of vertices of odd degree

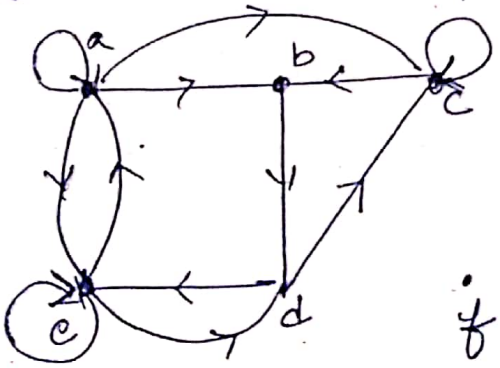
If (u,v) is an edge in a graph G_1. then
u is called initial vertex.
v is called terminal vertex.

→ The initial & terminal vertex of a loop are the same.

Indegree – The indegree of a vertex v, denoted by $\deg^-(v)$, is the no: of edges with v as their terminal vertex.

Outdegree – The ~~de~~ out-degree of v, denoted by $\deg^+(v)$, is the no: of edges with v as their initial vertex.

Eg Find the in-degree & out-degree of each vertex of graph G_1.



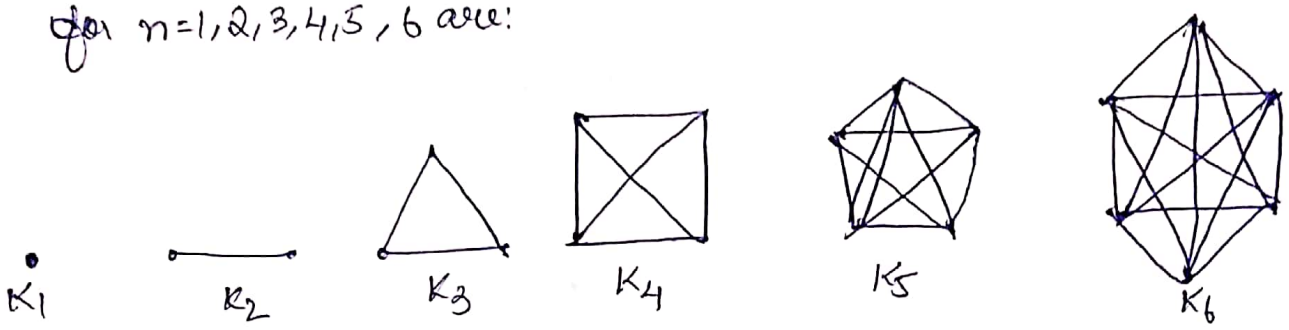
$\deg^-(a) = 2$ $\quad \deg^+(a) = 4$
$\deg^-(b) = 2$ $\quad \deg^+(b) = 1$
$\deg^-(c) = 3$ $\quad \deg^+(c) = 2$
$\deg^-(d) = 2$ $\quad \deg^+(d) = 2$
$\deg^-(e) = 3$ $\quad \deg^+(e) = 3$
$\deg^-(f) = 0$ $\quad \deg^+(f) = 0$

Theorem Let $G=(V,E)$ be a graph with directed edges. Then

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |E|.$$

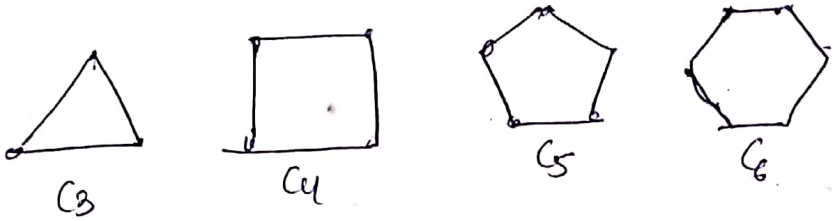
Some Special Simple Graphs

Complete Graphs - The complete graph on n vertices, denoted by K_n, is the simple graph that contains exactly one edge between each pair of distinct vertices. The graphs K_n, for $n = 1, 2, 3, 4, 5, 6$ are:

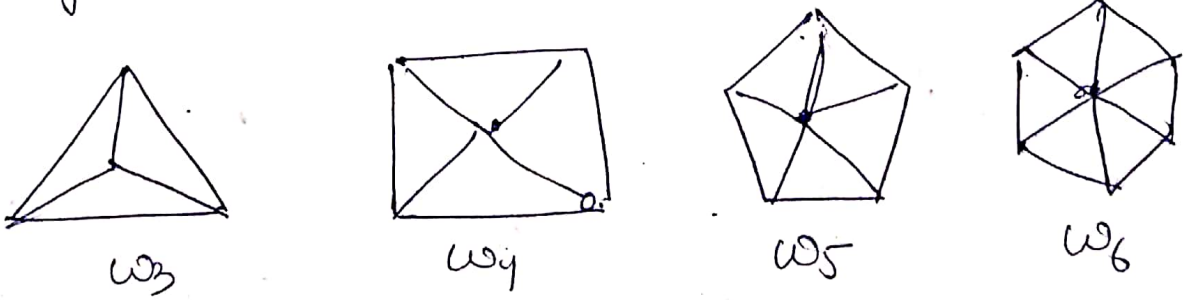


Cycles : The cycle C_n, $n \geq 3$, consists of n vertices ~~denoted~~ by $1, 2, \ldots n$ and edges $(1,2), (2,3), \ldots (n-1, n)$ and $(n, 1)$. The cycles C_3, C_4, C_5 & C_6 are displayed.

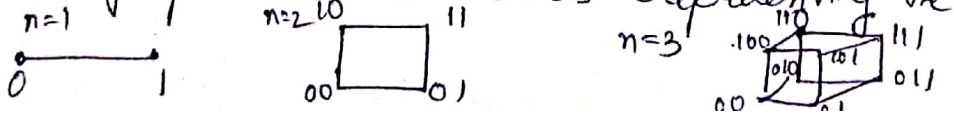
~~Wheels~~:



Wheels: We obtain the Wheel W_n when we add an additional vertex to the cycle C_n, for $n \geq 3$ and connect this new vertex to each of the n vertices in C_n, by new edges. The wheels W_3, W_4, W_5, W_6 are.

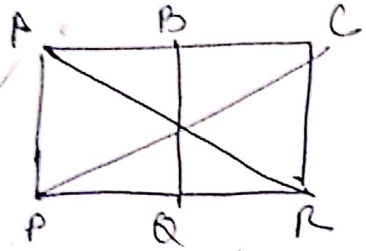


n-Cubes The n-dimensional hypercube or n-cube, denoted by Q_n, is the graph that has vertices representing the 2^n bit strings of length n.

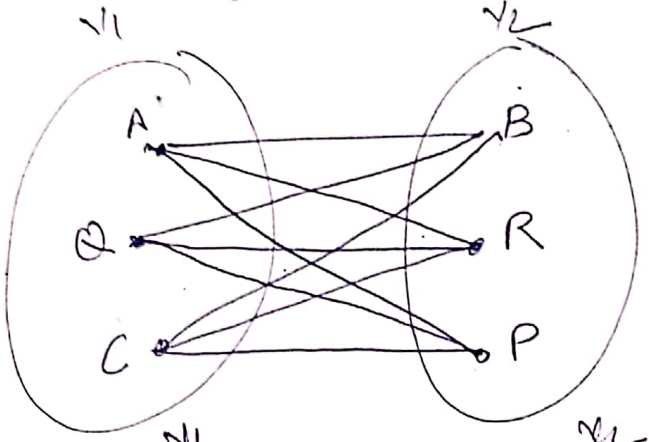


Bipartite Graphs : A simple graph G is called bipartite if its vertex set V can be partitioned into two disjoint sets V_1 and V_2 such that every edge in the graph connects a vertex in V_1 and a vertex in V_2 (so that no edge in G connects either two vertices in V_1 or two vertices in V_2). When this condition holds, we call the pair (V_1, V_2) a bipartition of the vertex set V of G. i.e V_1 & V_2 are called bipartites.

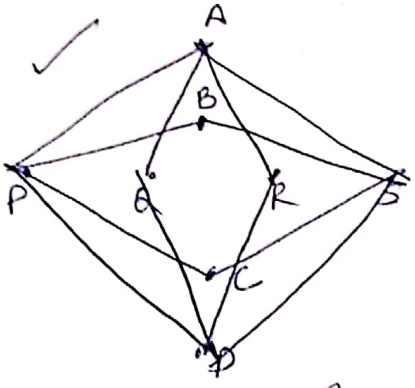
(1)



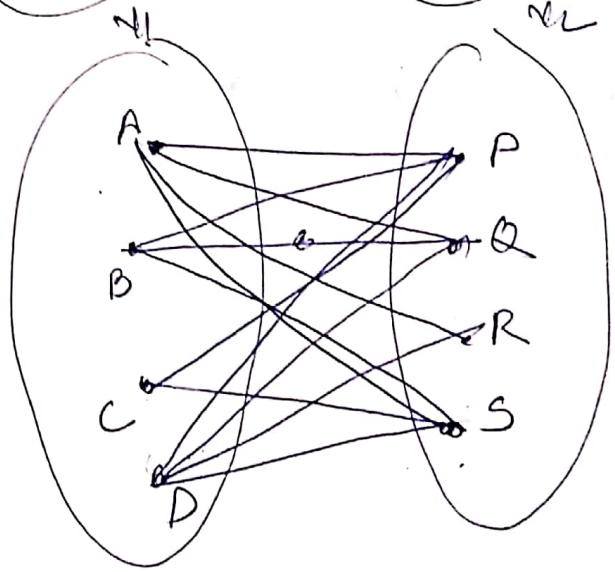
$V_1 = \{A, Q, C\}$
$V_2 = \{B, R, P\}$



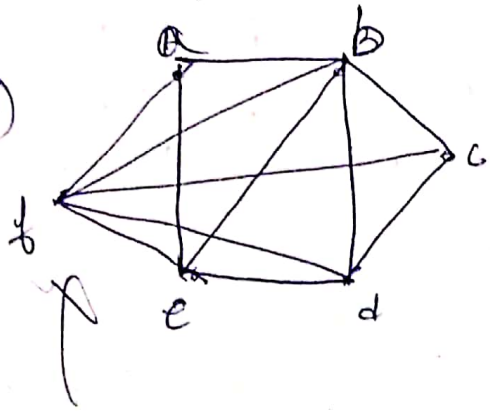
(2)

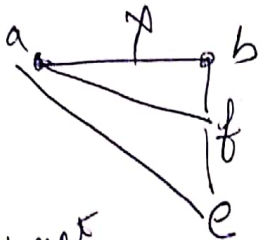


$V_1 = \{A, B, C, D\}$
$V_2 = \{P, Q, R, S\}$



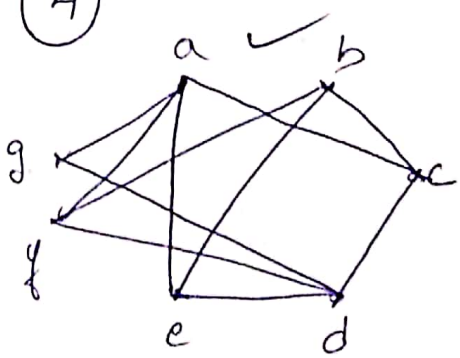
(3)





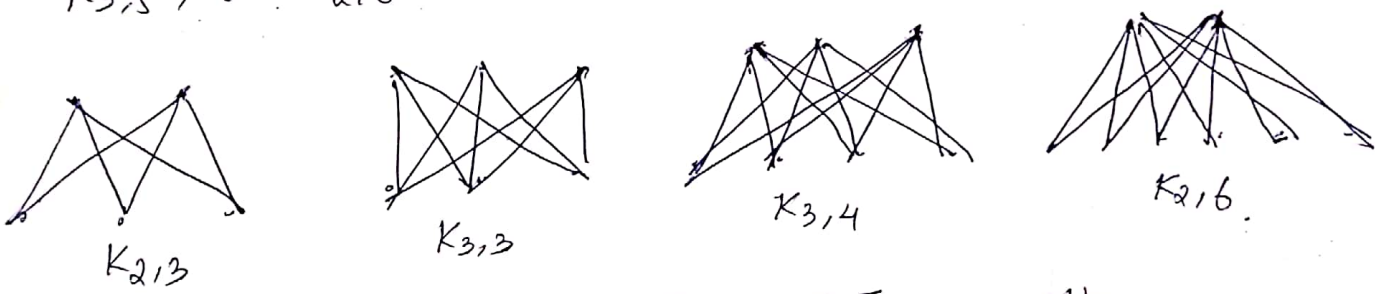
it is not bipartite

(4)



Theorem: A simple graph is bipartite if and only if it is possible to assign one of two different colors to each vertex of the graph so that no two adjacent vertices are assigned the same color.

Complete Bipartite Graphs – The complete bipartite graph $K_{m,n}$ is the graph that has its vertex set partitioned into two subsets of m & n vertices. There is an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset. The complete bipartite graphs $K_{2,3}$, $K_{3,3}$, $K_{3,5}$, & $K_{2,6}$ are



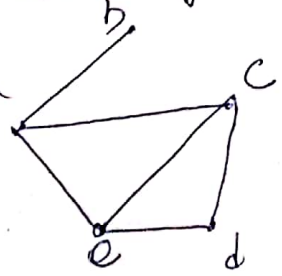
Representing Graphs and Graph Isomorphism

Representing Graphs

→ One way to represent a graph without multiple edges is to list all the edges of this graph.

→ Another way to do represent a graph with no multiple edges is to use adjacency list, which specify the vertices that are adjacent to each vertex of the graph.

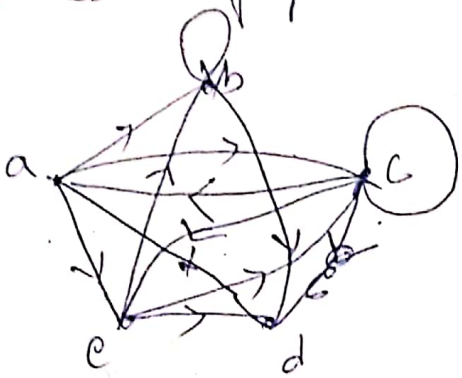
A Simple Graph



Adjacency list for simple graph

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

A Directed graph



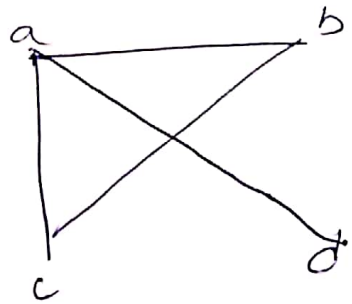
Adjacency List for a Directed graph	
Initial Vertex	Terminal Vertex
a	b, c, d, e
b	b, d
c	c, a, e
d	
e	b, c, d.

Adjacency Matrices

Suppose that $G = (V, E)$ is a simple graph where $|V| = n$. Suppose that the vertices of G are listed arbitrarily as $1, 2, \ldots, n$. The adjacency matrix A (or A_G) of G, with respect to this listing of the vertices, is the $n \times n$ zero-one matrix with 1 as its $(i,j)^{th}$ entry when i and j are adjacent, & 0 as its $(i,j)^{th}$ entry when they are not adjacent. In other words, if its adjacency matrix is $A = [a_{ij}]$, then

$$a_{ij} = \begin{cases} 1 & \text{if } \{i,j\} \text{ is an edge of } G, \\ 0 & \text{otherwise.} \end{cases}$$

Eg

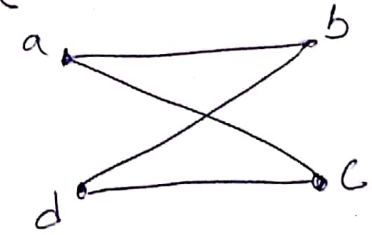


$$\begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

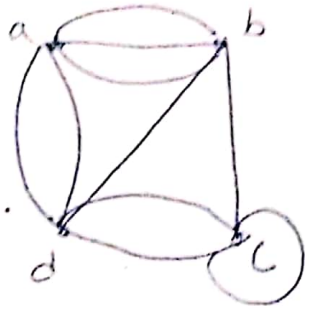
Draw graph with adjacency Matrix

Eg

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



Eg Use an adjacency matrix to represent pseudograph.



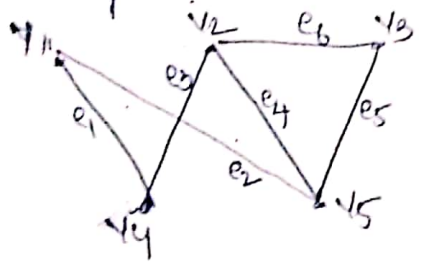
$$\begin{bmatrix} 0 & 3 & 1 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$

Incidence Matrices

Let $G = (V, E)$ be an undirected graph. Suppose that $1, 2, \ldots n$ are the vertices and $e_1, e_2 \ldots e_m$ are the edges of G. Then the incidence matrix with respect to this ordering of V and E is the $n \times m$ matrix $M = [m_{ij}]$, where.

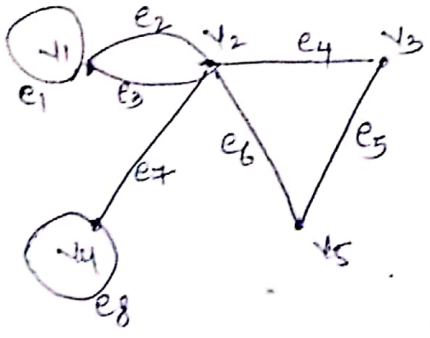
$$m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } i, \\ 0 & \text{otherwise.} \end{cases}$$

Eg) Represent the undirected graph with an incidence matrix



	e_1	e_2	e_3	e_4	e_5	e_6
1	1	1	0	0	0	0
2	0	0	1	1	0	0
3	0	0	0	0	1	1
4	1	0	1	0	0	0
5	0	1	0	1	1	0

Q) Represent the pseudograph using an incidence matrix.



	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
1	1	1	1	0	0	0	0	0
2	0	1	1	1	0	1	1	0
3	0	0	0	1	1	0	0	0
4	0	0	0	0	0	0	1	1
5	0	0	0	0	1	1	0	0

ISOMORPHISM

Two graphs G and G' are isomorphic if there is a function $f: V(G) \to V(G')$ from the vertices of G to the vertices of G' such that.

1) f is one to one
2) f is onto
3) for every pair of vertices u and v of G $\{u, v\} \in E(G)$ iff $\{f(u), f(v)\} \in E(G')$.

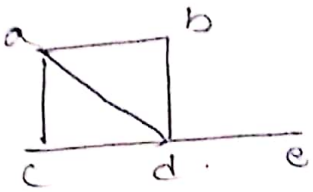
The condition (3) says that vertices u and v are adjacent in G iff f(u) and f(v) are adjacent in G'. In another words we say that the function f preserves adjacency.

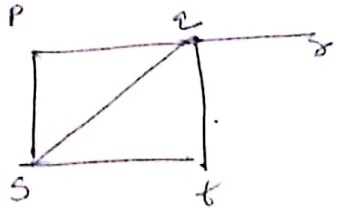
Also when G and G' are isomorphic then

1) $|V(G)| = |V(G')|$
2) $|E(G)| = |E(G')|$
3) if $v \in V(G)$, then $\deg_G(v) = \deg(f(v))$
4) if $\{u, v\}$ is a loop in G then $\{f(u), f(v)\}$ is a loop in G'.
5) if two graphs are isomorphic then their adjacency matrices are same.
6) if $v_0 v_1 \cdots v_{k-1} v_k = v_0$ is a cycle of length k in G then $f(v_0) - f(v_1) - f(v_2) \cdots f(v_{k-1}) \cdots f(v_k) = f(v_0)$ is a cycle of length k in G'

eg

(1)





G_1

G_1	Vertex	a	b	c	d	e
	degree	3	2	2	4	1

	p	q	r	s	t
	2	4	1	3	2

So $a \to s$, $d \to q$, $e \to r$.

No we take $b \to p$ and $c \to t$

AG_1 =

	a	b	c	d	e
a	0	1	1	1	0
b	1	0	0	1	0
c	1	0	0	1	0
d	1	1	1	0	1
e	0	0	0	1	0

AG_2

	s	p	t	q	r
s	0	1	1	1	0
p	1	0	0	1	0
t	1	0	0	1	0
q	1	1	1	0	1
r	0	0	0	1	0

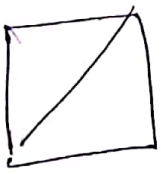
$AG_1 = AG_2$

So ~~AG_1 & AG_2~~ Graphs are isomorphic.

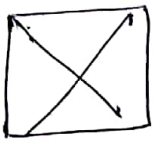
$|V_1| = |V_2|$

$|E_1| \neq |E_2|$ not isomorphic.

(2)

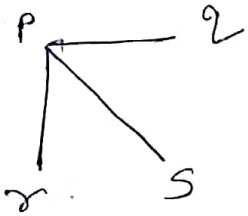


G_1



G_2

(3)



1) $|V_1| = |V_2|$
2) $|E_1| = |E_2|$
3) degree sequence

G_1	V	a	b	c	d	e	f	g	h
	d	3	2	3	2	2	3	4	3

$d_1 = (2, 2, 2, 3, 3, 3, 3, 4)$

G_2	V	p	q	r	s	t	u	v	w
	d	3	3	3	2	2	4	2	

$d_1 = (2, 2, 2, 3, 3, 3, 3, 4)$

$d_1 = d_2$

4) No: of cycles of different length then

G_{11} has 2 cycles of length 4

a-b-h-g-a

c-d-f-e-c.

G_{12} has 3 cycles of length 4

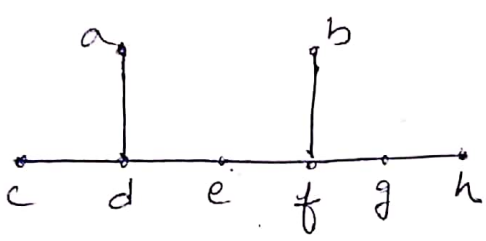
p-q-w-v-p

r-s-u-t-r

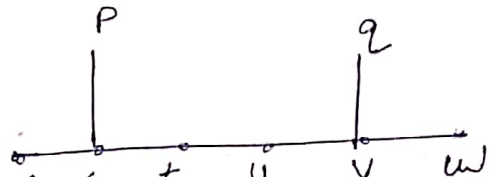
p-q-s-r-p.

So G_{11} & G_{12} are not isomorphic.

(4)



G_{11}



G_{12}

1) $|V_1| = |V_2|$

2) $|E_1| = |E_2|$

3) degree sequence.

G_1	V	a	b	c	d	e	f	g	h
	D	1	1	1	3	2	3	2	1

G_1	V	p	q	r	s	t	u	v	w
	D	1	1	1	3	2	2	3	1

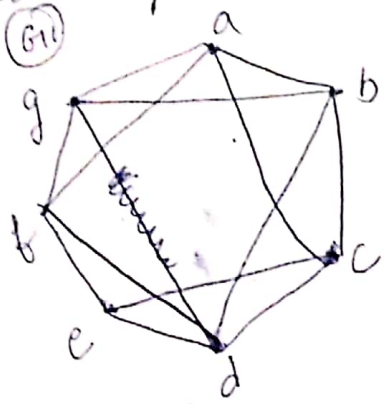
So $d_1 = d_2$

4) G_{11}, G_{12} both has no cycles.

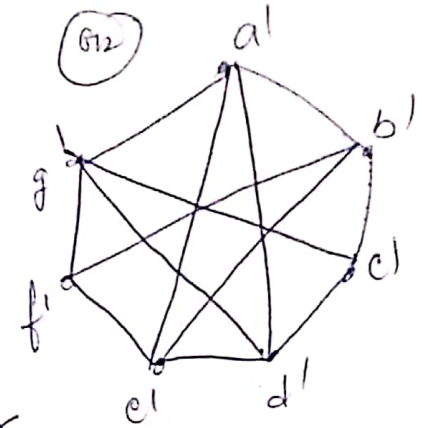
5) But still graphs are not isomorphic since the length of the path between 3 degree vertices in G_{11} is 2(d-e-f) while in G_{12} is 3(s-t-u-v).

(5) isomorphic or not

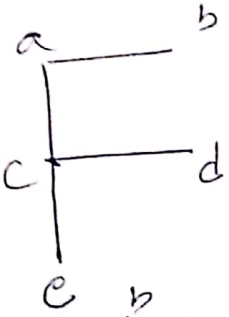
G1

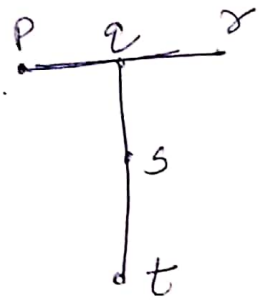


G2

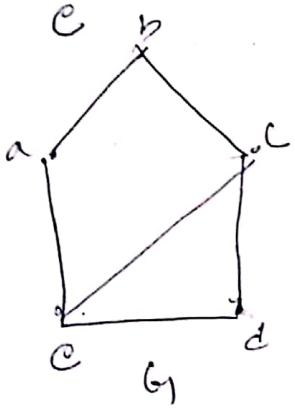


(6)

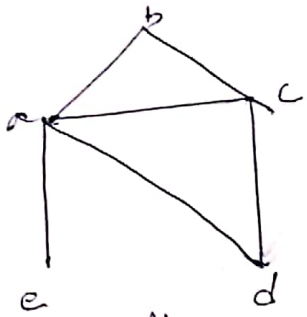




(7)



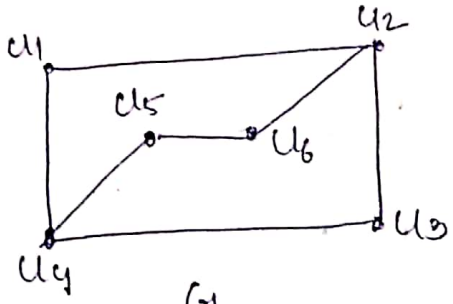
G1



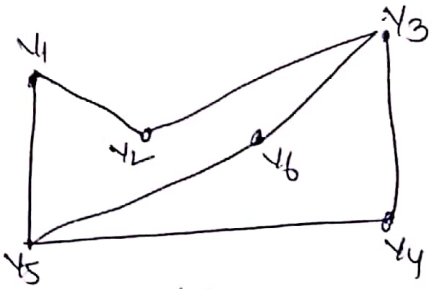
H

✗ Not isomorphic

(8)



G1



H

Isomorphic

Connectivity

Path - A path is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph.

→ The path is a **circuit** if it begins and ends at the same vertex i.e if $u=v$, and has length greater than zero.

→ A path or circuit is simple if it does not contain the same edge more than once.

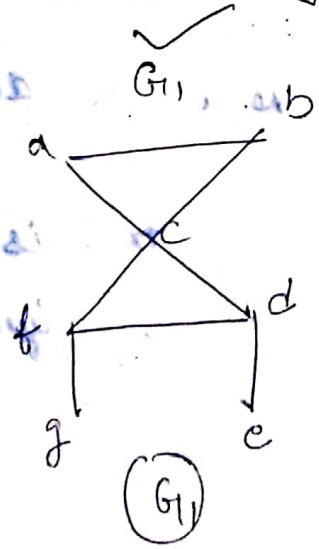
Walk is defined to be an alternating sequence of vertices and edges of a graph $v_0, e_1, v_1, e_2 \ldots v_{n-1}, e_n, v_n$.
where v_{i-1} and v_i are the endpoints of e_i for $i=1,2\ldots,n$.

Closed walk is used instead of circuit to indicate a walk that begins and ends at the same vertex.

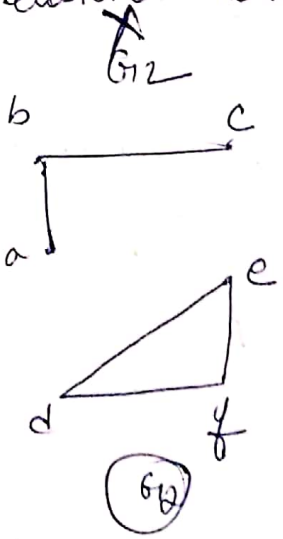
Trail is used to denote a walk that has no repeated edge.

Connectedness In Undirected Graphs

→ An undirected graph is called **connected** if there is a path between every pair of distinct vertices of the graph.



G_1



G_2

Cut Vertices

Removal from a graph of a vertex and all incident edges produces a subgraph with more connected components. Such vertices are called cut vertices (or articulation points). i.e produces a subgraph that is not connected.

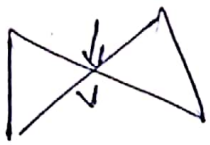
Cut Edge

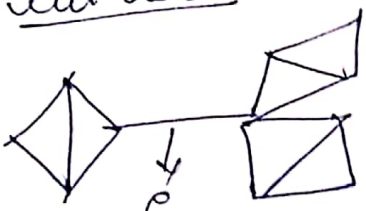
Removal of an edge from a graph produces with more connected components than in the original graph is called a cutedge or bridge.

Cut Edge - If a graph G is a connected and 'e' is an edge such that G-e is not connected, the 'e' is said to be a bridge or a cut Edge.

Cut Vertex (or Articulation Point)

If 'v' is a vertex of G such that 'G-v' is not connected, then 'v' is a cut vertex.





Connectedness in Directed Graphs

→ A directed graph is strongly connected if there is a path from a to b and from b to a whenever a and b are vertices in the graph.

→ A directed graph is weakly connected if there is a path between every two vertices in the underlying undirected graph.

Euler's formula

If G is a connected plane graph then $|V|-|E|+|R|=2$

(or)

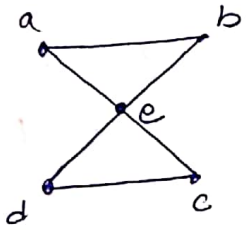
For any plane graph G, no: of vertices − no: of edges + no: of regions = 2.

→ An Euler circuit in a graph G is a simple circuit containing every edge of G.

→ An Euler path in G is a simple path containing every edge of G.

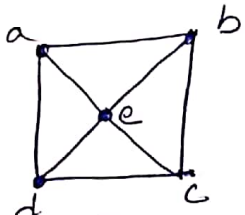
Note: Here vertices may be touched more than once but edges must appear only once.

Eg Which of the below undirected graph have an Euler circuit? Of those that do not, which have an Euler path?



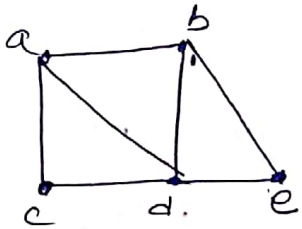
G_1

G_1 has Euler circuit
Eg a, e, c, d, e, b, a



G_2

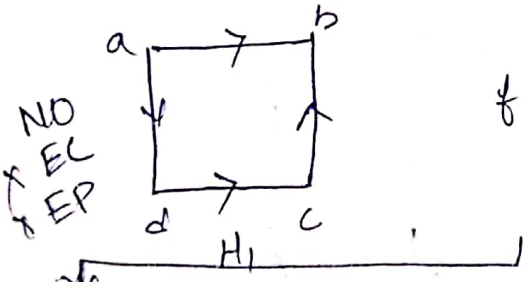
NO EC
NO EP



G_3

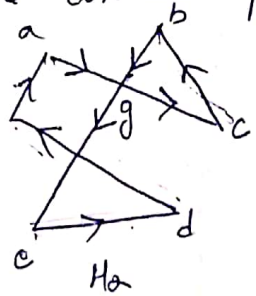
NO EC
EP → a, c, d, e, b, d, a, b.

Eg Which of the directed graphs have an Euler circuit? Of those that do not, which have an Euler path?

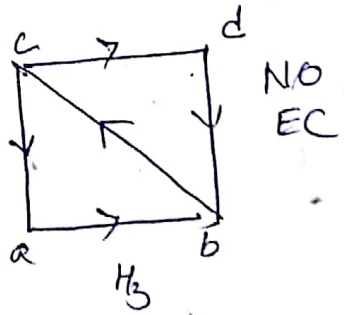


H_1

EC − a, g, c, b, g, e, d, f, a



H_2



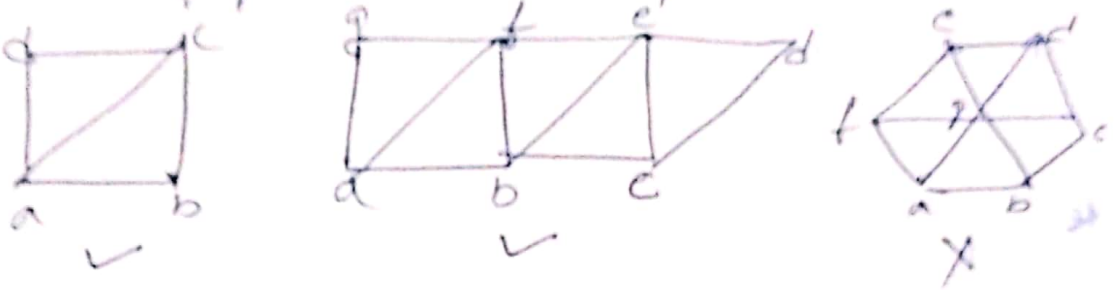
H_3

EP − c, a, b, c, d, b

Theorem: A connected multigraph with atleast two vertices has an Euler circuit if and only if each of its vertices has even degree

Theorem: A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.

e) Which graphs have an Euler path



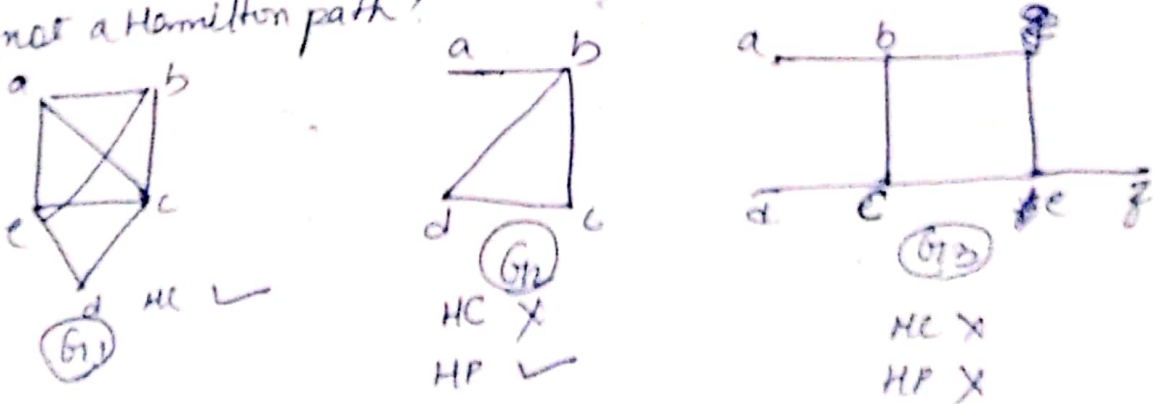
Hamilton Path & Circuit

Hamilton Path – A simple path in a graph G that passes through every vertex exactly once is called Hamilton path.

Hamilton Circuit – A simple circuit in a graph G that passes through every vertex exactly once is called a Hamilton circuit.

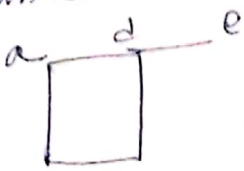
Note – Hamilton circuit uses each vertex exactly once (except for the first & last) but may skip edges.

Eg Which of the simple graphs have a Hamilton circuit or if not a Hamilton path?

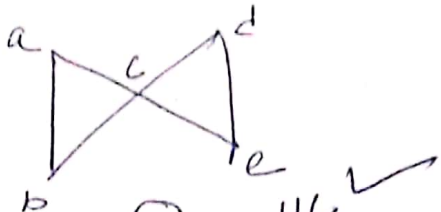


Condition for the existence of Hamilton Circuits

→ A graph with a vertex of degree one cannot have a Hamilton circuit.



NO HC (6)



(11) HC

K_n has a Hamilton circuit whenever $n \geq 3$.

DIRAC's Theorem – If G_1 is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G_1 is atleast $n/2$, then G_1 has a Hamilton circuit.

ORE's Theorem : If G_1 is a simple graph with n vertices with $n \geq 3$ such that $\deg(u) + \deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G_1, then G_1 has a Hamilton circuit.

Shortest-Path Problems

Many problems can be modeled using graphs with weights assigned to their edges.

→ Graphs that have a number assigned to each edge are called weighted graphs.

→ Weighted graphs are used to model computer networks.

→ The length of a path in a weighted graph be the sum of the weights of the edges of this path.

i,e finding the least length is the target.

Travelling salesperson problem can be solved using DP

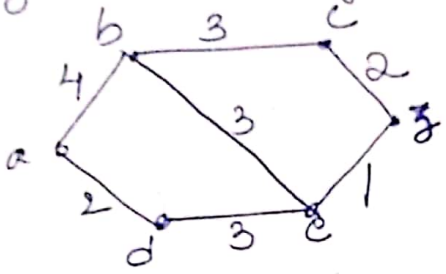
A Shortest-Path Algorithm

There are several different algorithms that find a shortest path between two vertices in a weighted graph.

→ A greedy algorithm discovered by the Dutch mathematician Edsger Dijkstra in 1959.

→ It solve shortest-path problem to solve shortest-path in undirected weighted graphs where all the weights are positive.

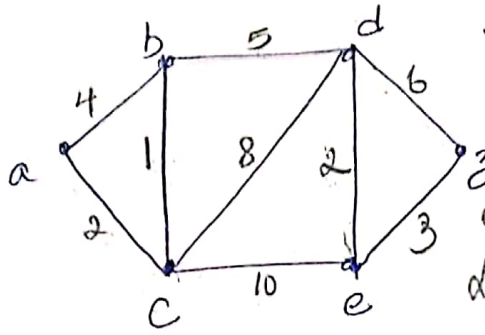
eg what is the length of a shortest path between a and z in the weighted graph.



	a	b	c	d	e	z
	∞	∞	∞	∞	∞	∞
a	0	4	∞	(2)	∞	∞
{a, d}	0	(4)	∞	(2)	5	∞
{a, d, b}	0	(4)	7	(2)	(5)	∞
{a, d, b, e}	0	(4)	7	(2)	(5)	(6)
{a, d, b, e, z}	0	(4)	(7)	(2)	(5)	(6)
{a, d, b, e, z, c}	0	(4)	(7)	(2)	(5)	6

∴ The length of shortest path from a to z is 6

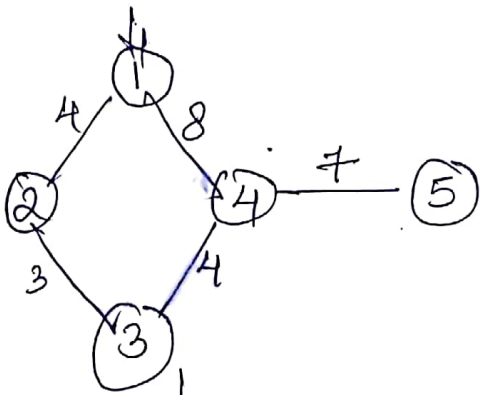
eg Find the shortest path



Visited	a	b	c	d	e	z
	∞	∞	∞	∞	∞	∞
a	0	4	2	∞	∞	∞
{a,c}	0	3	2	10	12	∞
{a,c,b}	0	3	2	9	12	∞
{a,c,b,d}	0	3	2	8	10	13
{a,c,b,d,e}	0	3	2	8	10	13
{a,c,b,d,e,z}	0	3	2	8	10	13

The shortest path from a to z is a, c, b, d, e, z with length 13.

eg



Visited	1	2	3	4	5
	∞	∞	∞	∞	∞
{1}	0	4	∞	8	∞
{1,2}	0	4	7	8	18
{1,2,3}	0	4	7	8	18
{1,2,3,4}	0	4	7	8	15
{1,2,3,4,5}	0	4	7	8	15

Dijkstra's Algorithm

procedure Dijkstra (G: weighted connected simple graph, with all weights positive)

{G has vertices $a = v_0, v_1, \ldots, v_n = z$ and lengths $w(v_i, v_j)$ where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G}

for $i := 1$ to n

$L(v_i) := \infty$

$L(a) := 0$

$S := \emptyset$

{the labels are now initialized so that the label of a is 0 and all other labels are ∞, and S is the empty set}

while $z \notin S$

$u :=$ a vertex not in S with $L(u)$ minimal

$S := S \cup \{u\}$

for all vertices v not in S

if $L(u) + w(u, v) < L(v)$ then $L(v) := L(u) + w(u, v)$

{this adds a vertex to S with minimal label and updates the labels of vertices not in S}

return $L(z)$ {$L(z)$ = length of a shortest path from a to z}

Theorem statement: Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) to find the length of a shortest path between two vertices in a connected simple undirected weighted graph with n vertices.

Planar Graphs

A graph is planar if it has a drawing without crossings.

Following are the criteria to detect the planarity of a connected graph.

1) If G is connected planar graph then
$$|V| - |E| + |R| = 2.$$

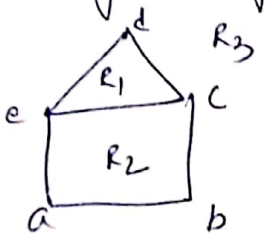
2) A complete graph K_n is planar iff $n \leq 4$; where n is the no: of vertices.

3) A complete bipartite graph $K_{m,n}$ is planar iff $m \leq 2$ or $n \leq 2$.

4) A connected simple graph is planar with ~~if~~ $|E| > 1$
   (a) $|E| \leq 3|V| - 6$ &
   (b) There is a vertex v of G such that degree(v) ≤ 5.

5) A graph is non-planar iff it contains a subgraph that is homomorphic to either K_5 or $K_{3,3}$.

Eg Verify the given graph is planar or not



Euler's formula
$|V| - E| + |R| = 2$
$5 - 6 + 3 = 2$
$2 = 2$

∴ Above graph is planar

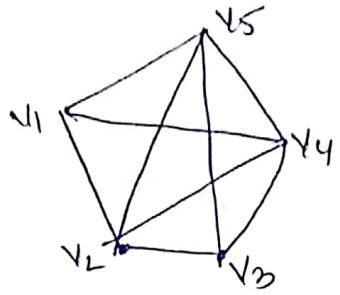
Corollary: If a connected planar simple graph has e edges and v vertices with $v \geq 3$ and no circuits of length three, then $e \leq 2v - 4$

Q) Which of the following non-planar graph have the property that the removal of any vertex and all edges incident with that vertex produces a planar graph?

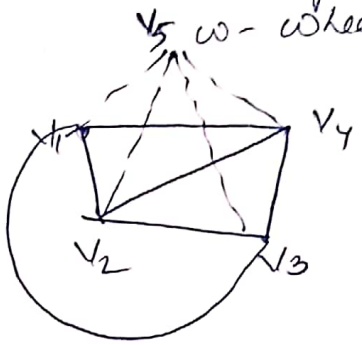
(i) K_5 (ii) K_6 (iii) $K_{3,3}$ (iv) $K_{3,4}$

K - Complete graph
C - Cycle
W - Wheel

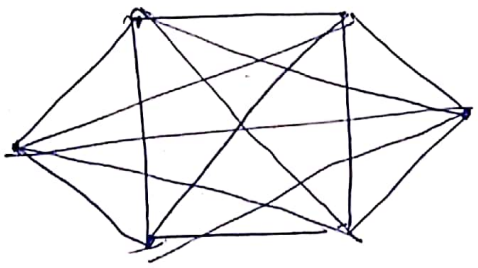
(i)



After removing V_5

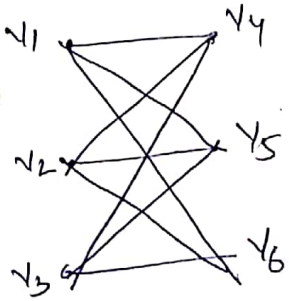


(ii) K_6

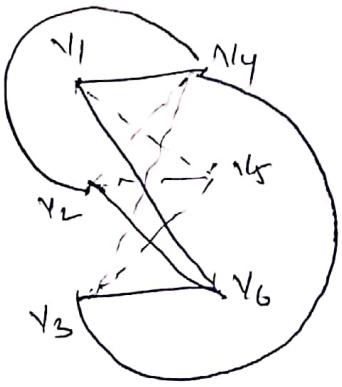


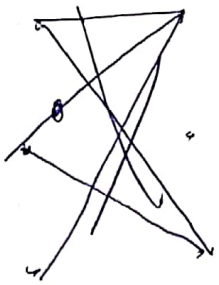
The graph resulting after the removal of any vertex of K_6, & all edge incident with that vertex cannot be a planar graph.

(iii) $K_{3,3}$

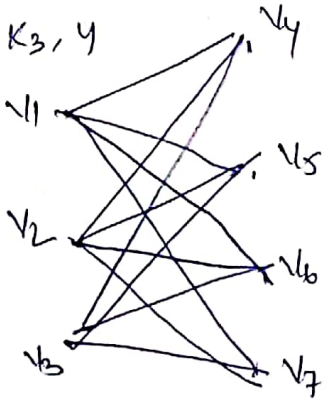


Remove V_5





(iv) $K_{3,4}$



⟶ Not possible

Dual of a planar graph

Definition - Given a plane graph G_1, one can define another graph G_1^* as follows:

Corresponding to each region of G_1 there is a vertex f^* and G_1^* and corresponding to each edge of G_1 there is an edge e^* of G_1^*, two vertices f^* and g^* are joined by e^* if G_1^* iff their corresponding regions are separated by a common edge 'e' in G_1. (i.e 2 regions are adjacent). (geometric dual)

The graph G_1^* is called dual of G_1.

Note that if e is a loop of G_1, then e^* is a pendent edge in G_1^* & vice versa.

Pendent edge = An edge incident on a pendant vertex.

Rules to Construct a dual of G_1

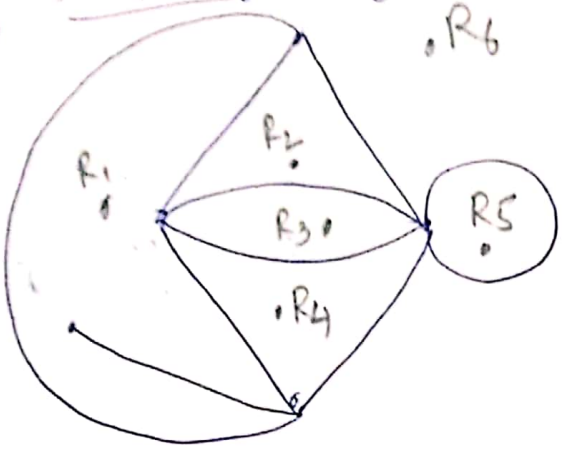
1) Place the vertices $P_1, P_2 - - P_n$ of the graph to be constructed in one in each region of a plane representation of a given graph.
2) Suppose if 2 regions say R_i & R_j have an edge in common and contain new vertices P_i and P_j, join these vertices or points P_i & P_j that intersects the common edge exactly one.
3) If there is more than one edge common between R_i & R_j draw one line between points P_i and P_j for each of the common edges.

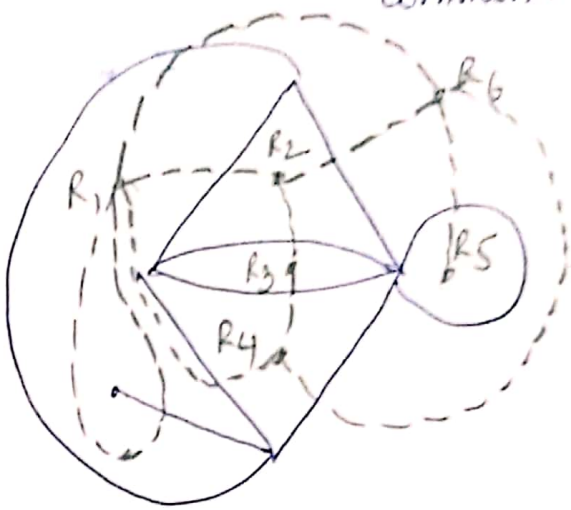
4) For an edge e entirely lying in one region say R_i i.e pendant edge, draw a self-loop at point P_i intersecting e exactly once.

5) For each loop in G_1, draw an edge between the vertices in two regions which have loop in common.

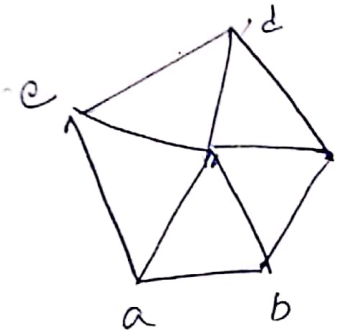
Find dual of the following planar graph G

eg



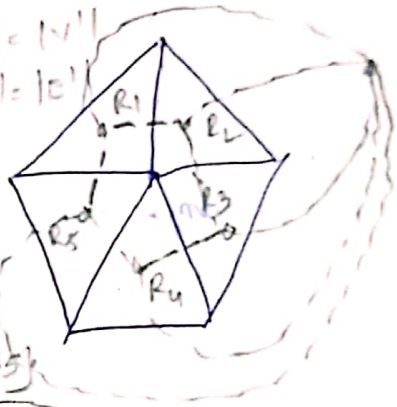


eg

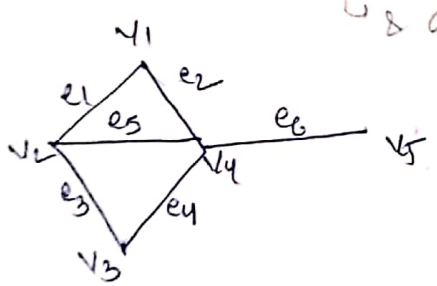


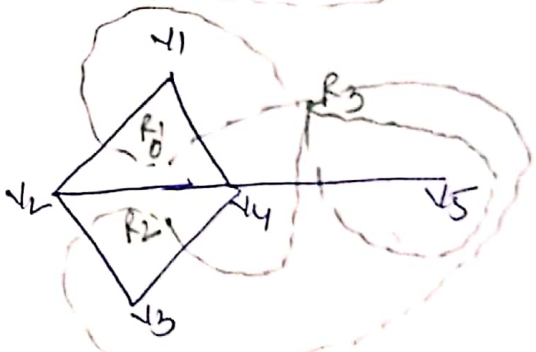
isomorphic
one-one & onto. $|V| = |V'|$
$|E| = |E'|$

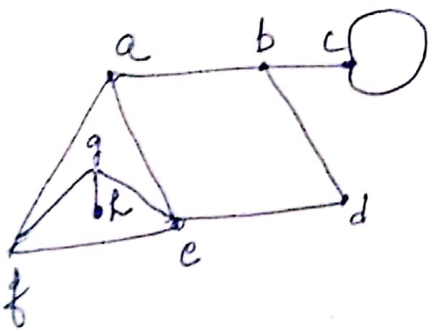
$f(a) = a'$, $f(b) = b'$
$f(c) = c'$, $f(d) = d'$
$f(e) = e'$, $f(f) = f'$
& degree sequence $\{3,3,3,3,3,5\}$

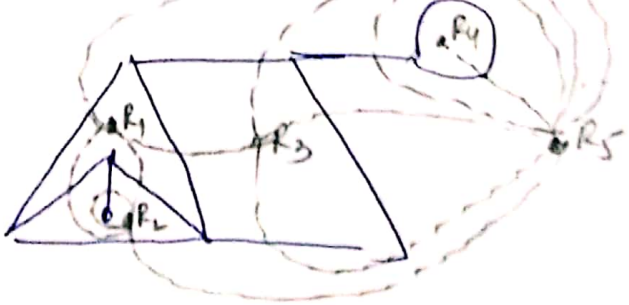


eg



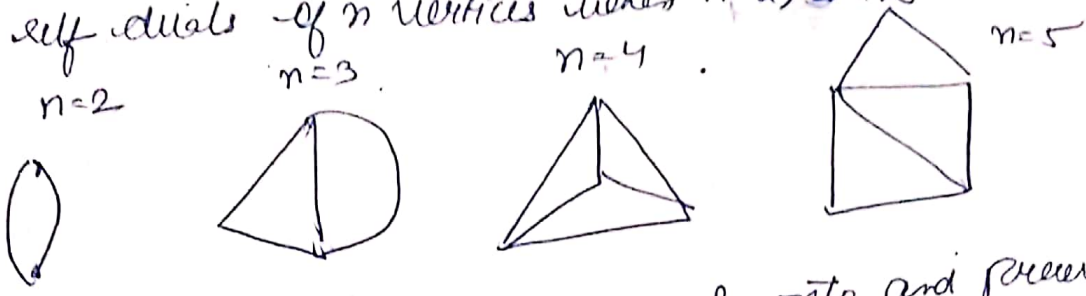






Self dual: A graph is called a self dual graph, if it is isomorphic to its dual.

1) Find self duals of n vertices when n=2, 3, 4, 5



Isomorphic:- Function is one-one & onto and preserves adjacency.

$\chi(G)$ for complete bipartite $K_{m,n}$ is 2.

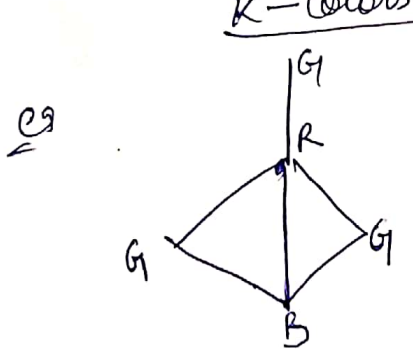
Chromatic Number

→ A coloring of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.

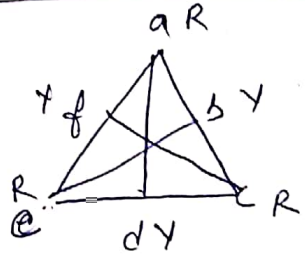
→ The chromatic number of a graph is the least number of colors needed for a coloring of this graph. The chromatic number of a graph G is denoted by $\chi(G)$.

→ Theorem- The chromatic number of a planar graph is no greater than four.

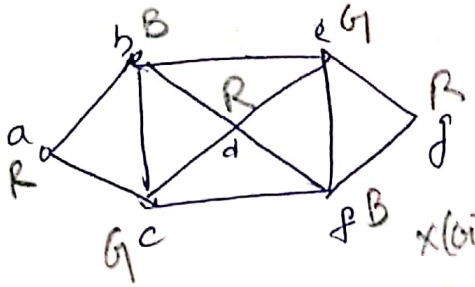
K-colors mean K chromatic graph.



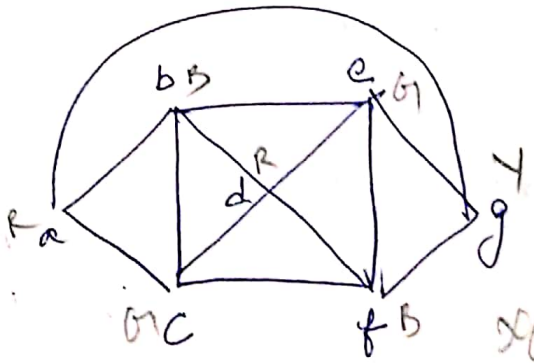
3-Chromatic graph



$\chi(G) = 2$



$\chi(G) = 3$



$\chi(G) = 4$

Trees

→ A tree is a connected undirected graph with no simple circuit.

→ An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices

→ A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

→ If there is a directed edge from u to v, then u is the parent of v and v is called a child of v.

→ Vertices with the same parent are called siblings.

→ The ancestors of a vertex other than the root are the vertices in the path from the root to this vertex.

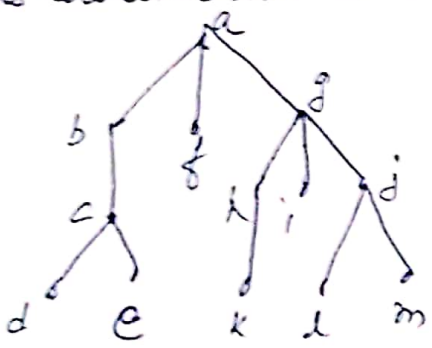
→ The descendants of a vertex v are those vertices that have an ancestor.

→ A vertex of a rooted tree is called a leaf if it has no children.

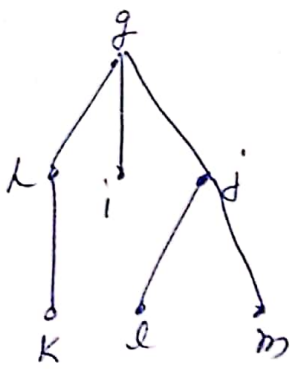
→ Vertices that have children are called internal vertices.

→ If a is a vertex in a tree, the subtree with a as its root is the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

Eg =



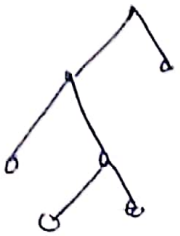
A Rooted Tree T



Subtree Rooted at g

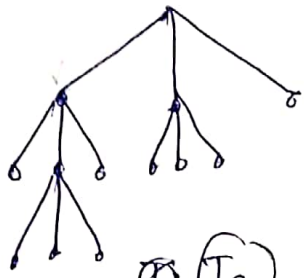
Definition: A rooted tree is called a **m-ary tree**, if every internal vertex has no more than m children. The tree is called a **full m-ary tree**, if every internal vertex has exactly m children. An m-ary tree with m=2 is called a **binary tree**.

Eg Are the given rooted trees, full m-ary trees for some positive integer m?



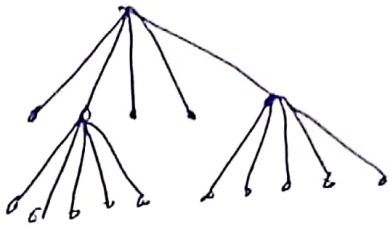
T1

full binary tree



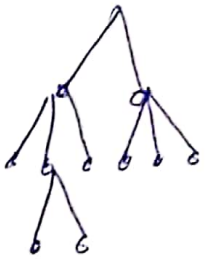
T2

full 3-ary tree



T3

full 5-ary tree

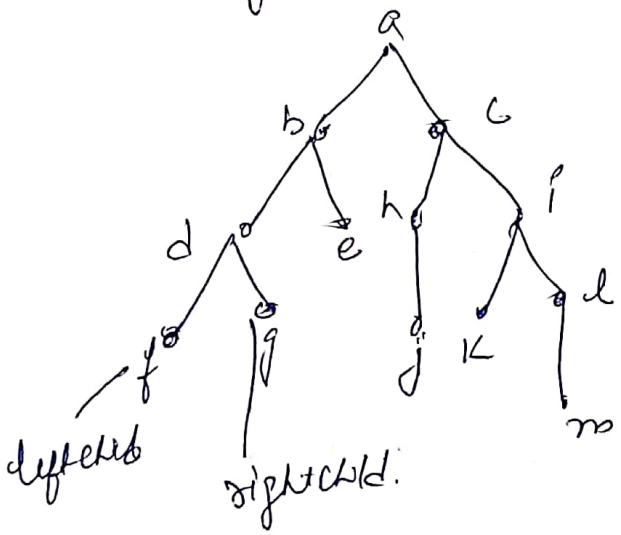


T4

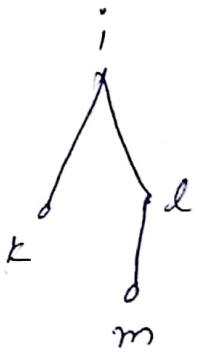
not full ary tree

Ordered Rooted Trees

→ An ordered rooted tree is a rooted tree where the children of each internal vertex are ordered.



left subtree of c



right subtree of c

Theorems Properties of Trees
proofs from T.B

1) A tree with n vertices has $n-1$ edges.

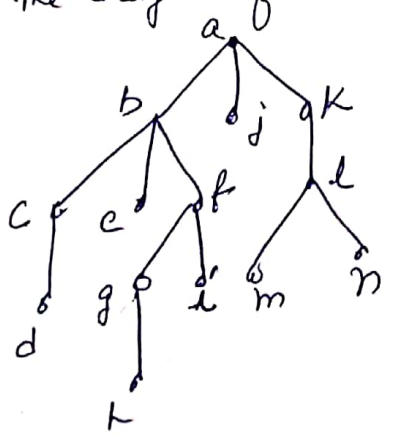
2) A full m-ary tree with i internal vertices contains $n = mi+1$ vertices

3) A full m-ary tree with
   (i) n vertices has $i = (n-1)/m$ internal vertices and $l = [(m-1)n+1]/m$ leaves.
   (ii) i internal vertices has $n = mi+1$ vertices and $l = (m-1)i+1$ leaves.
   (iii) l leaves has $n = (ml-1)/(m-1)$ vertices and $i = (l-1)/(m-1)$ internal vertices

→ The level of a vertex v in a rooted tree is the length of the unique path from the root to this vertex.
- The level of the root is defined to be zero.

→ The height of a rooted tree is the length of the longest path from the root to any vertex.

Eg Find the level of each vertex in the rooted tree. What is the height of this tree?



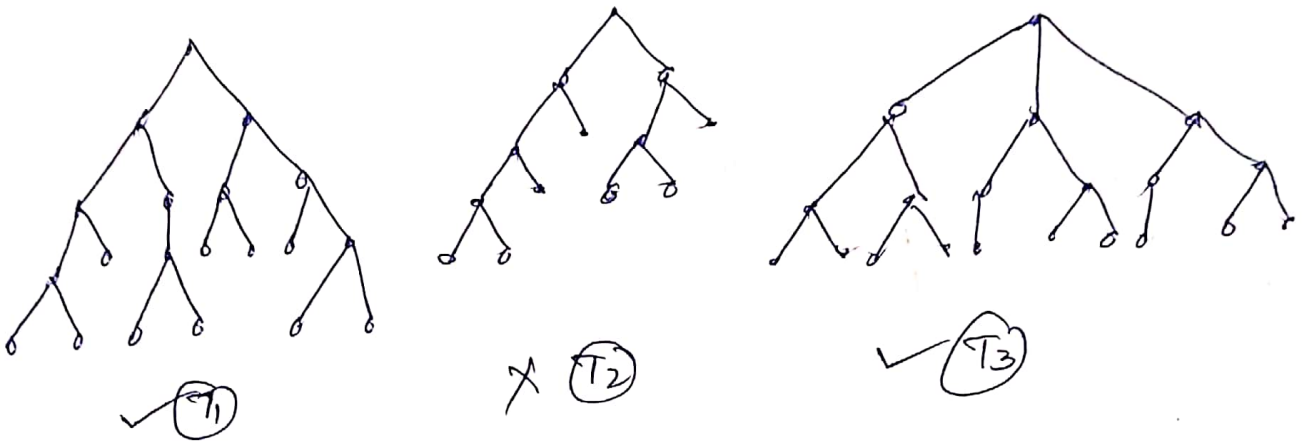
levels
a — 0
b, j, K — 1
c, e, f, l — 2
d, g, i, m, n — 3
h — (4) → maximum

height is 4

Property

4) A rooted m-ary tree of height h is balanced if all leaves are at levels h or h-1.

eg Which of the rooted trees are balanced



✓ T1 ✗ T2 ✓ T3

5) There are almost m^h leaves in an m-ary tree of height h.

6) If an m-ary tree of height 'h' has l leaves, then $h \geq \lceil \log_m l \rceil$. If the m-ary tree is full and balanced, then $h = \lceil \log_m l \rceil$.

Tree Traversal

→ Ordered rooted tree are often used to store information.

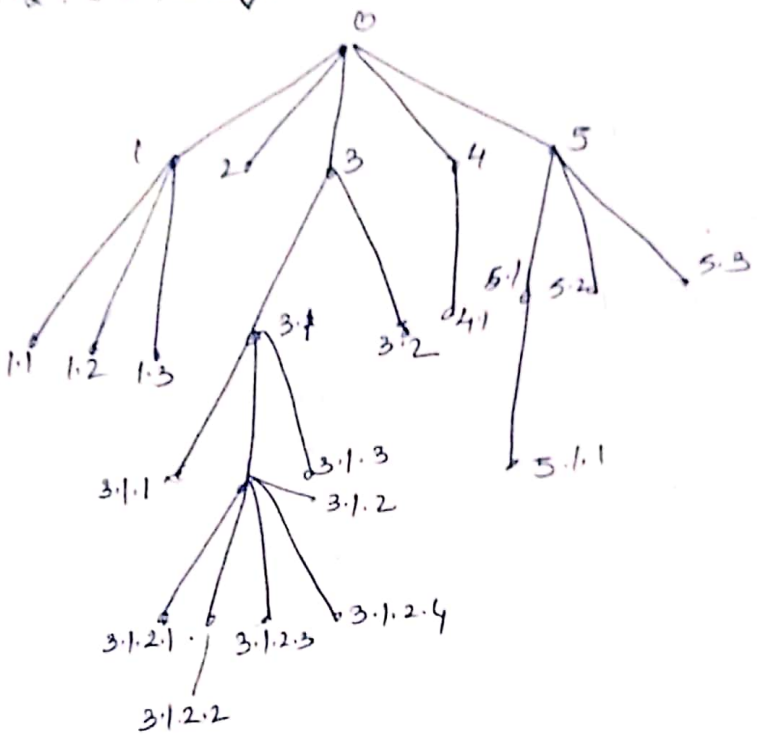
→ In order to visit each vertex of an ordered rooted tree to access data, we need procedures.

– The different listings of the vertices of ordered rooted trees used to represent expressions are useful in the evaluation of them expressions.

Universal Address Systems

1) Label the root with the integer 0. Then label its k children (at level 1) from left to right with 1, 2, ... k.

2) For each vertex N at level n with label A, label its k_v children, as they are drawn from left to right, with A.1, A.2 ... A.k_v.



Traversal Algorithms

Procedures for systematically visiting every vertex of an ordered rooted tree are called Traversal algorithms.

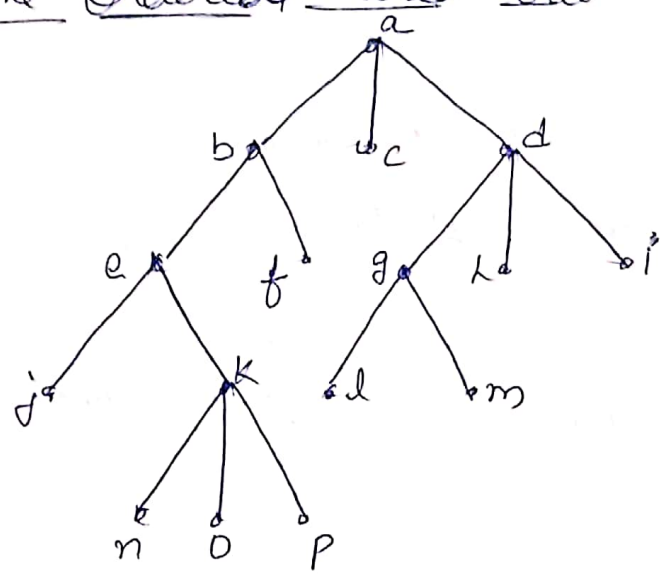
3 algorithms are

preorder traversal → root, left, right

inorder " → left, root, right

Postorder " → left, right, root

The Ordered rooted tree

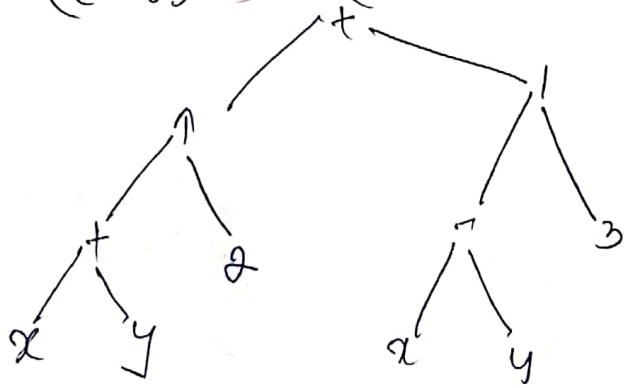


Pre-order → a b e j k n o p f c d g l m h i

~~Post-order~~ Inorder j e n k o p b f a c l g m d h i

Postorder → j n o p k e f b c l m g h i d a

Eg What is the ordered rooted tree that represent the expression $((x+y)\uparrow 2)+((x-4)/3)$?



Prefix form + ↑ + x y 2 / − x 4 3.

Evaluating Prefix exp

+ − * 2 3 5 / ↑ 2 3 4

8 4

+ − * 2 3 5 2

+ − 6 5 2

+ 1 2

3

Evaluating Postfix exp

7 2 3 * − 4 ↑ 9 3 / +

7 6 − 4 ↑ 9 3 / +

1 4 ↑ 9 3 / +

1 9 3 / +

1 3 +

4

Q) What is the value of prefix exp $+-*235/\uparrow234$

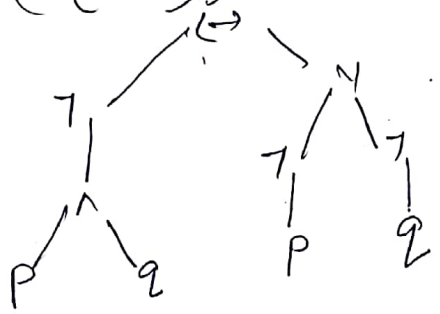
Ans ~~$+-*235/\uparrow234$~~

3

Q) $((x+y)\uparrow 2)+((x-4)/3)$

Postfix $xy+2\uparrow x4-3/+$

Q) Find the ordered rooted tree representing the compound proposition $(\neg(P\wedge Q))\leftrightarrow(\neg P\vee\neg Q)$



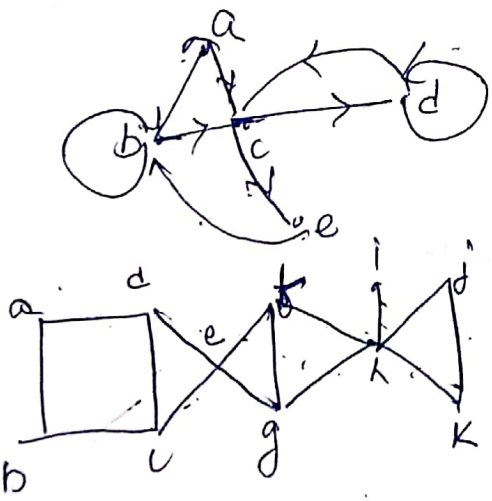
Spanning Trees

Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.

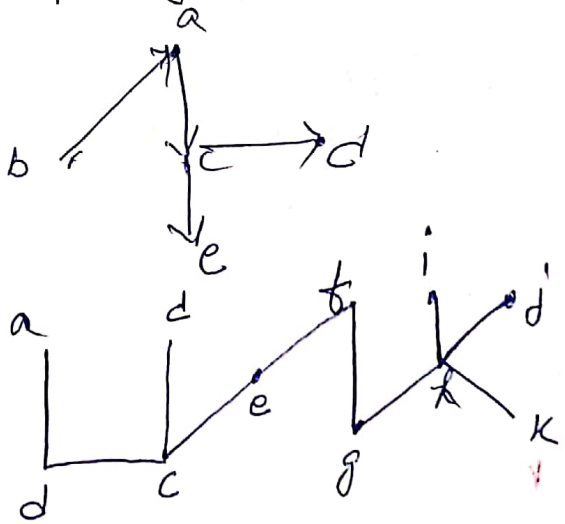
Theorem: A simple graph is connected if and only if it has a spanning tree.

Eg

Graph



Spanning tree



→ In general, if G is a Connected graph with n vertices and m edges, a spanning tree of G must have n-1 edges. Hence, the no: of edges that must be removed before a spanning tree is obtained must be $m-(n-1)=m-n+1$. This number is frequently called the circuit rank of G.

→ We define algorithms for finding a spanning tree of connected graph. These algorithms are known as breadth first search and depth-first search algorithms.

Algorithm Breadth-First Search

```
procedure BFS (G: connected graph with vertices V1, V2 ..., Vn)
T := tree consisting only of vertex V1
L := empty list
put V1 in the list L of unprocessed vertices
while L is not empty
begin
    remove the first vertex, V, from L
    for each neighbour w of V
        if w is not in L and not in T then
        begin
            add w to the end of the list L
            add w and edge {V,w} to T
        end
end.
```

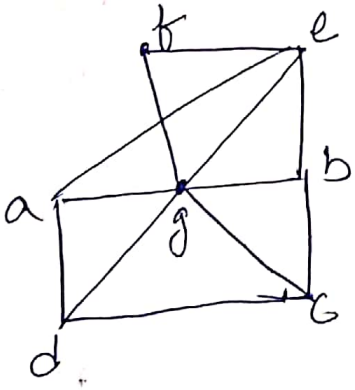
BFS:

Procedure: We start at any vertex, taken as a root, and find edges incident on it. The vertices added at this stage become the vertices at level 1 in the spanning tree, arbitarily order them.

Next for each vertex at level 1, taken inorder, we add edges incident in it so long as it does not produce a cycle.

- This leads to the vertices at level 2. We repeat the procedure till no further edge can be added to any of the vertex at the last level reached.

Eg Use BFS to find a spanning tree for the graph given below with the vertex ordering abcdefg and gfedcba.

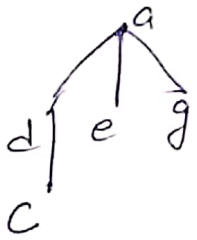


Let the vertex ordering be abcdefg

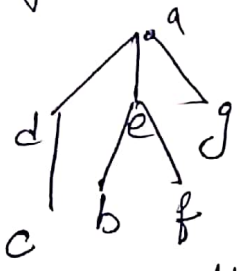
1) Start with vertex 'a' as a root.

2) Add all the edges incident on 'a'. They are. {a,d}, {a,e}, {a,g}. The vertices d, e, g are at level 1 in the tree

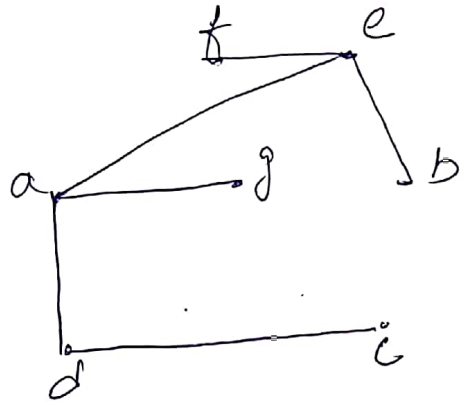
3) Add d to {d, c} but not {d, g} as it forms a cycle.



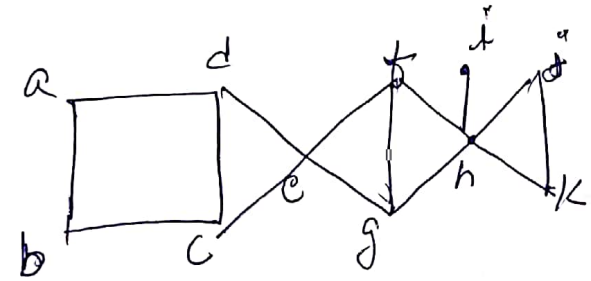
4) Add e add {e, b} & {e, f} but not {e, g} as it form a cycle

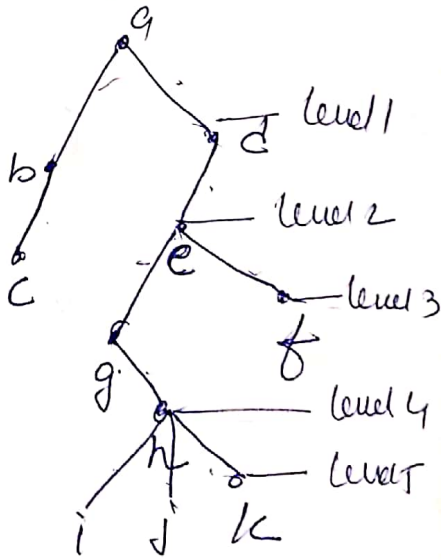


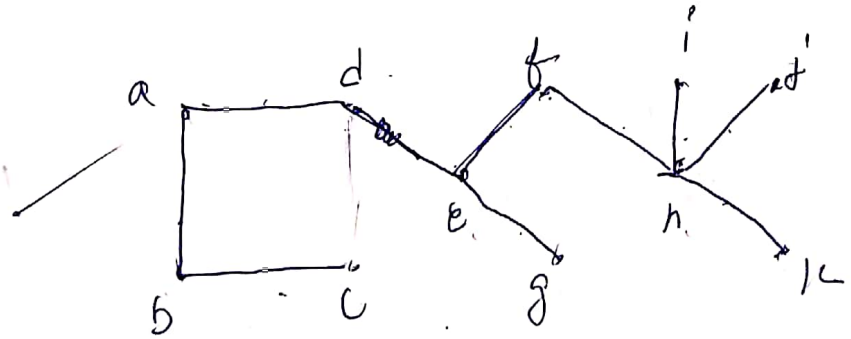
It contains all the vertices. Hence it is a spanning tree. We can also write this spanning tree as follows.



(2)







DFS (Also called Backtracking)

Algorithm Depth-First Search.

```
procedure DFS (G: Connected graph with vertices 1, 2, ... n)
T := tree consisting only of the vertex V1
  visit(1)
  procedure visit (: vertex of G)
  for each vertex w adjacent to and not yet in T
  begin
     add vertex w and edge {, w} to T
     visit(w)
  end
```

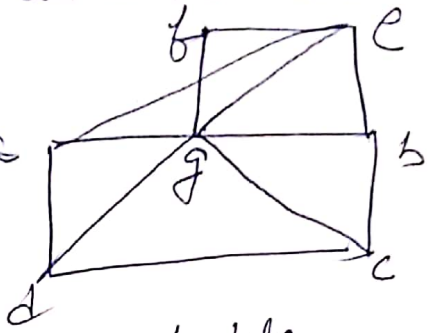
Procedure:

Arbitarily choose a vertex as the root of the spanning tree T. Form a path starting at this vertex by successively adding edges where each new edge is incident with the last vertex in the path and the vertex not already in the path. Continue this process by adding edges to this path as long as possible without producing any cycles.

- If the path goes through all the vertices in the graph, then the tree becomes a spanning tree. Otherwise more edges must be added, move back to next vertex to last vertex in the graph and if possible, form a new path starting at this vertex passing through that are not already visited.

If still all the vertices are not visited, move back to another vertex and try again. Repeat this procedure until no more edges can be added.

Q) Apply DFS for

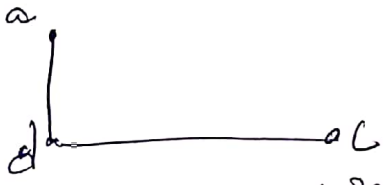


Let the vertex ordering be abcdefg.

(i) Start with the vertex ~~Ordering be abcdefg~~ 'a' as a root.

(ii) We can not add b and c to a. Add d to a

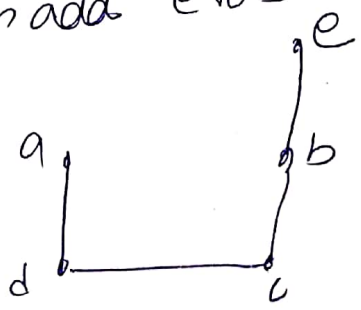
(iii) Now start with d. We cannot add b. Add c to d



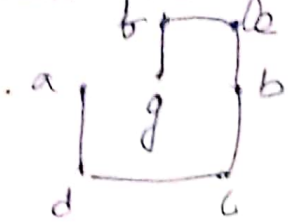
(iv) We start with c. We can~~not~~ add b to c.

(v) Start at b. We remain with vertices e, f, g.

We can add e to b

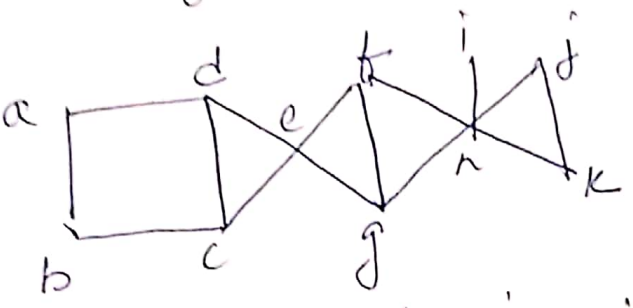


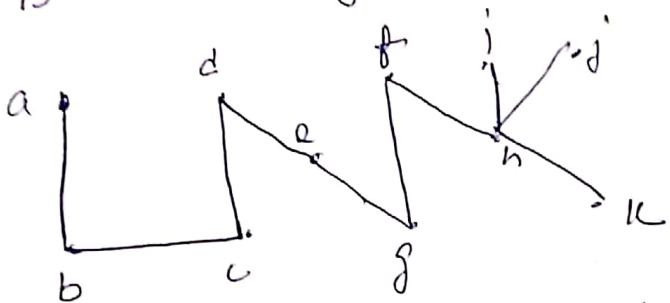
(vi) we can add f to e and then g to f



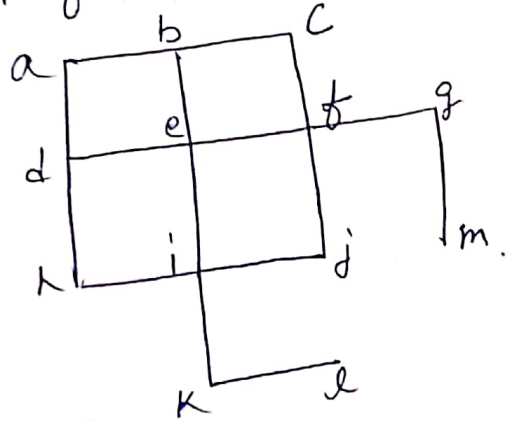
It contains all the vertices, hence it is the spanning tree as ~~follows~~ above.

(2)

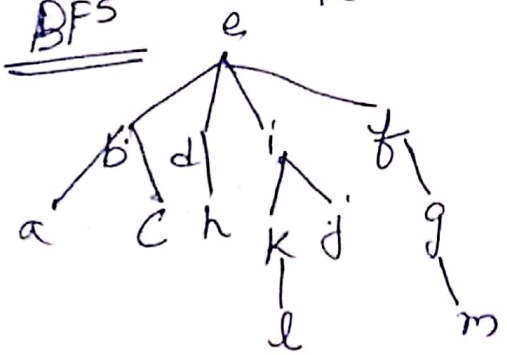




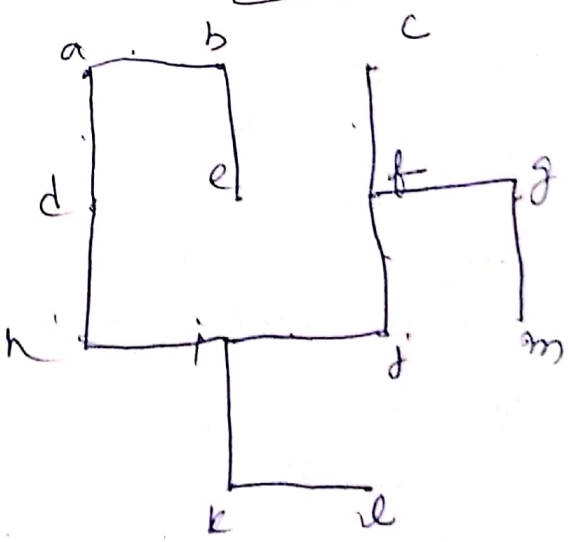
Eg Using the BFS and DFS method, determine the spanning tree T for the graph G with 'e' as the root of T



BFS



DFS



Minimal Spanning Tree

Definition: A minimum spanning tree in a connected weight graph is a spanning tree that has the smallest possible sum of weights of its edges.

There are two methods for constructing minimal spanning trees (i.e 2 algorithms)

1. Prim's alg.
2. Kruskal's alg.

Algorithm 1 Prim's Algorithm

procedure Prim (G: weighted connected undirected graph with n vertices)

T := a minimum-weight edge

for i = 1 to n-2

e := an edge of minimum weight incident to a vertex in T and not forming a simple circuit in T if added to T.

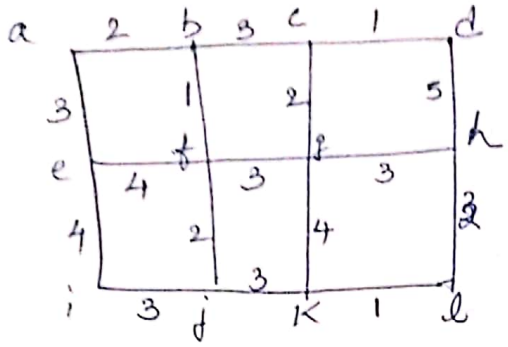
T := T with e added.

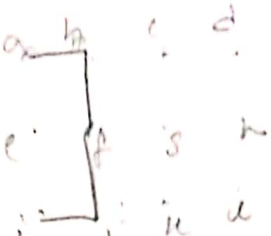
return T {T is a minimum spanning tree of G}

Procedure

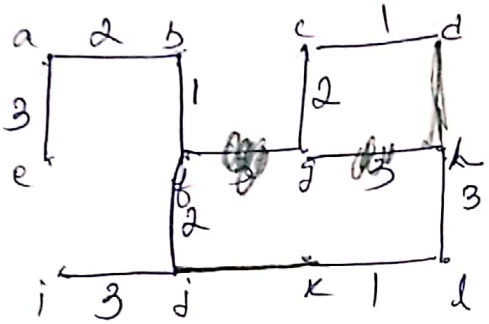
1) Begin by choosing an edge with smallest weight, putting it into the spanning tree.
2) Successively add to the tree edges of minimum weight that are incident to a vertex already in the tree, never forming a simple circuit with those edges already in the tree.
3) Stop when n-1 edges have been added.

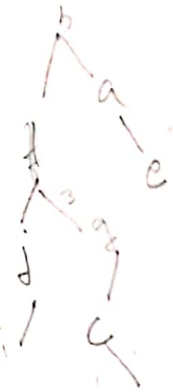
eg Produce Minimum Spanning Tree using Prim's Algorithm.





b-f	1
ab	2
f-j	2
a-e	3
j-i	3
fg	3
c-g	2
c-d	1
gh	3
n-l	3
k-l	1
	24





(2) Kruskal's Alg

Procedure:

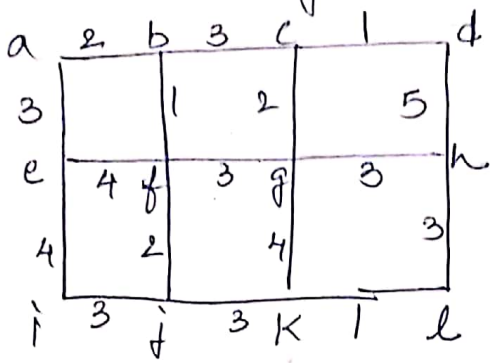
1) Choose an edge in the graph with minimum weight.
2) Successively add edges with minimum weight that do not form a simple circuit with those edges already chosen.
3) Stop after n-1 edges have been selected.

Algorithm 2 Kruskal's Alg

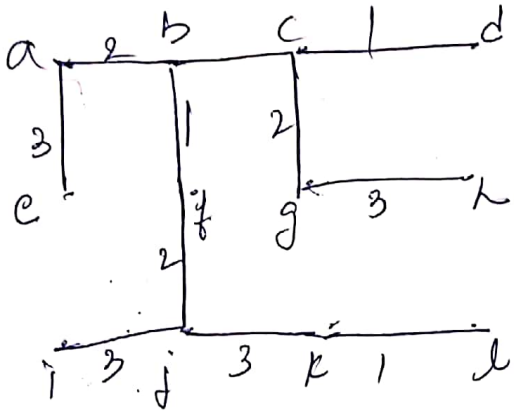
procedure Kruskal (G: weighted connected undirected graph with n vertices)

T := empty graph
for i := 1 to n-1
  e := any edge in G with smallest weight that does not form a simple circuit when added to T.
  T := T with e added
return T {T is a minimum spanning tree of G}.

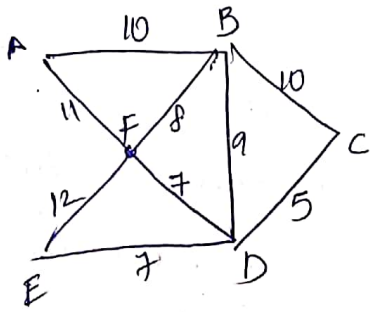
eg Produce a minimum spanning tree using Kruskal's Algorithm.

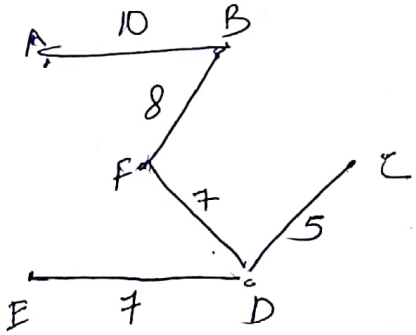


edge	weight
c-d	1
k-l	1
b-f	1
c-g	2
a-b	2
f-j	2
b-c	3
j-k	3
g-h	3
i-j	3
a-e	3
	24

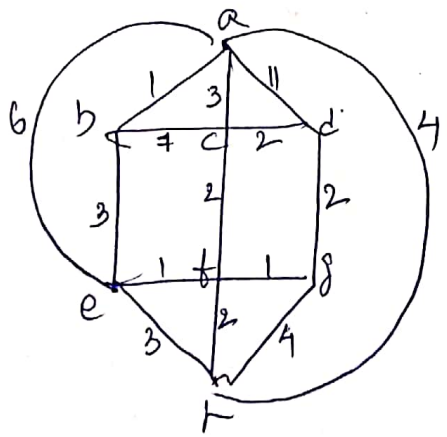


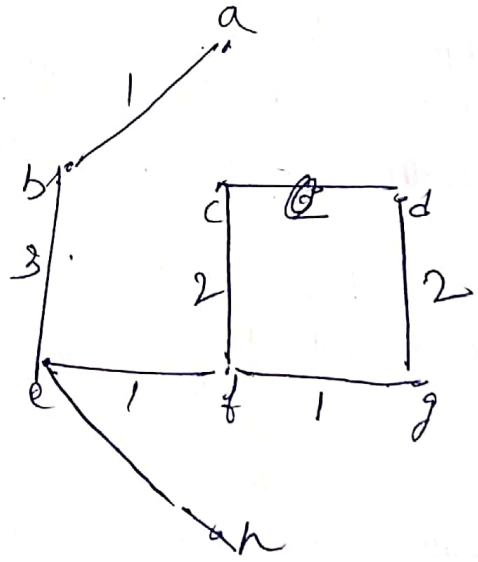
eg

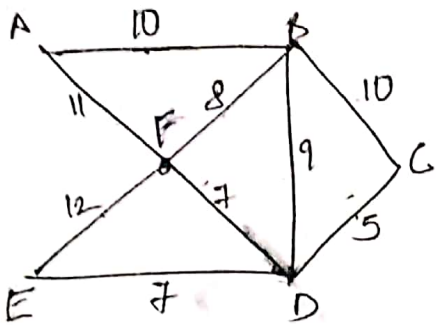




eg

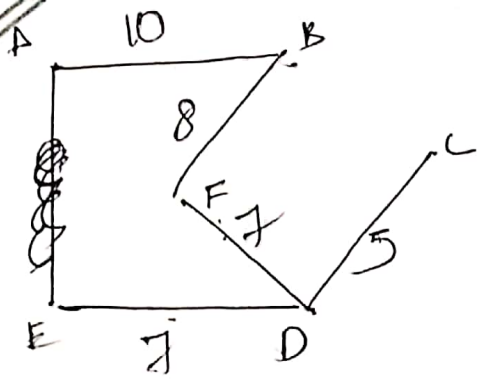






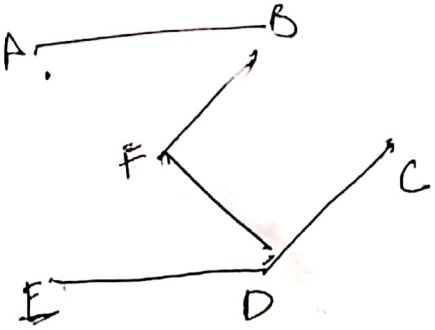
	A	B	C	D	E	F
A	-	10			11	
B	10	-	10	9	8	8
C		10	-	5		
D		9	5	-	7	7
E				7	-	12
F	11	8		7	12	-

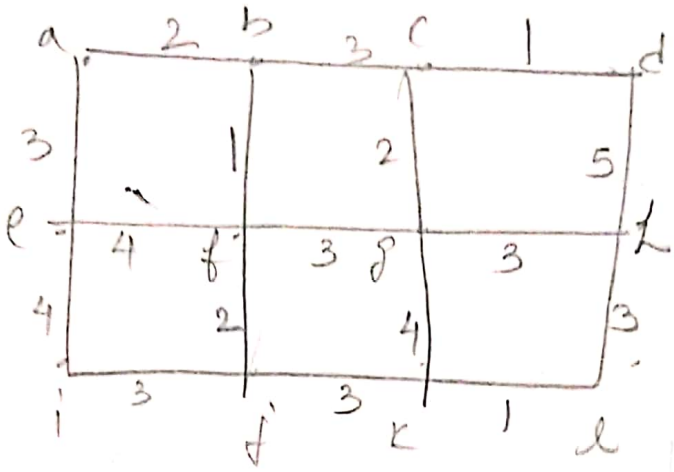
Prim's

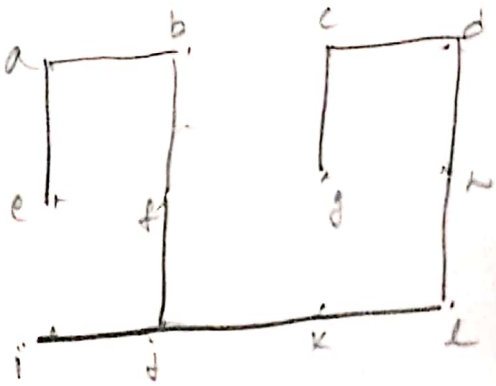


Kruskal

Edge	CD	ED	FD	FB	BD	AB	BC	AF	EF
Weight	5	7	7	8	9	10	10	11	12







	a	b	c	d	e	f	g	h	i	j	k	l
a	-	2	∞	∞	3	∞	∞	∞	∞	∞	∞	∞
b	∞	-	3	∞	∞	1	∞	∞	∞	∞	∞	∞
c		3	-	1			2					
d			1	-				5				
e	3				-	4			4			
f		1			4	-	3			2		
g			2			3	-	3			4	
h				5				-				3
i					4				-	3		
j						2			3	-	3	
k							4			3	-	1
l								3			1	-