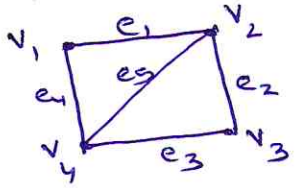
UNIT-V

GRAPHS

A Graph G is a pair of sets (V, E), where V is a set of vertices & E is a set of edges.

$G = (V, E)$



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

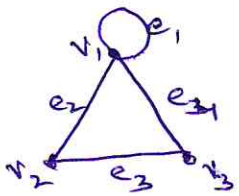
$E = \{e_1, e_2, e_3, e_4, e_5\}$.

The edges are identified as

$e_1 = \{v_1, v_2\}$, $e_2 = \{v_2, v_3\}$, $e_3 = \{v_3, v_4\}$,

$e_4 = \{v_4, v_1\}$, $e_5 = \{v_2, v_4\}$

An edge drawn from a vertex to itself is called a loop.



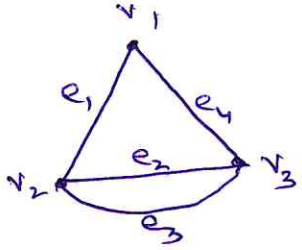
$V = \{v_1, v_2, v_3\}$

$E = \{e_1, e_2, e_3, e_4\}$.

$e_1 = \{v_1, v_1\}$, $e_2 = \{v_1, v_2\}$, $e_3 = \{v_2, v_3\}$.

Here the edge e_1 is a loop.

In a graph if some pair of vertices are joined by more than one edge, such edges are called Parallel edges.



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

$E = \{e_1, e_2, e_3, e_4\}$.

$e_1 = \{v_1, v_2\}$, $e_2 = \{v_2, v_3\}$, $e_3 = \{v_2, v_3\}$

$e_4 = \{v_1, v_3\}$

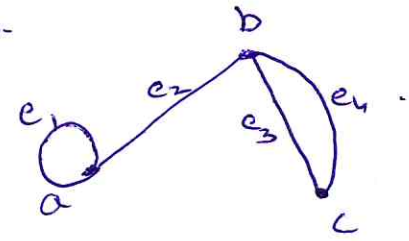
e_3, e_2 are Parallel edges.

A graph without loops & parallel edges is called a simple graph.

If $G=(V,E)$ is a finite graph then the number of vertices in G is called the order of G. & it is denoted by $|V(G)|$ or $|V|$. Number of edges in G is called size of G & it is denoted by $|E(G)|$ or $|E|$.

The degree of a vertex is determined by counting each loop incident on a vertex V as twice & each other edge incident on vertex V as once. It is denoted as deg(V).

Eg:-



order of G i.e $|V| = 3$

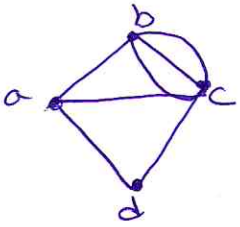
Size of G i.e $|E| = 3$

deg (a) = 3

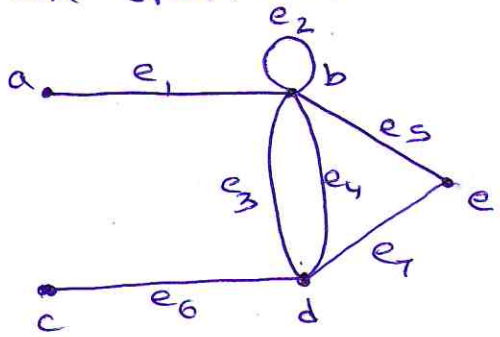
deg (b) = 2

deg (c) = 1

If one allows more than one edge to join a pair of vertices, the result is then called a multigraph.



A walk in a graph is a finite alternating sequence of vertices & edges $v_0 e_1 v_1 e_2 v_2 \cdots e_n v_n$.
such that $e_i = (v_{i-1}, v_i)$, $1 \le i \le n$.
If $v_0 = v_n$ it is called a closed walk, otherwise it is an open walk.



$a e_1 b e_2 b e_4 d$ is an open walk, length = 3

$a e_1 b e_2 b e_1 a$ is a closed walk, length = 3

$a e_1 b e_6 c$ is not a walk

A walk is called a trail if all its edges are distinct.
An open trail is called a path.
A closed trail is called a circuit.

A circuit in which all the vertices are distinct except the first & last vertices is called a cycle.

$be_5 e e_7 d$ is a path

$be_5 e e_7 d e_4 b$ is a circuit

$be_4 d e_3 b e_4 d$ is a walk but not a trial.

$be_5 e e_7 d e_4 b$ is a cycle.

$be_5 e e_7 d e_4 b e_2 b$ is not a cycle but a circuit. $\because$ b is repeated.

A path of length ≥ 1 with no repeated edges & whose end points are equal is called a circuit. A circuit may have repeated vertices other than the end points. A cycle is a circuit with no other repeated vertices except its end points.

A cycle is a simple circuit.

A loop is a cycle of length 1.

ISOMORPHISM

Two graphs G_1 & G_2 are isomorphic if there is a function $f: V(G_1) \to V(G_2)$ such that-

i) f is one-to-one

ii) f is onto.

iii) f-preserves adjacency

i.e for each pair of vertices u & v of G_1, $\{u, v\} \in E(G_1)$ iff $\{f(u), f(v)\} \in E(G_2)$.

Any function f with the above three properties is called an isomorphism from G_1 to G_2.

The condition (iii) says that vertices u & v are adjacent in G_1 iff f(u) & f(v) are adjacent in G_2.

Discovering Isomorphism:

The problem of determining whether or not two graphs are isomorphic is known as the isomorphic problem. To solve it we verify the following in the two graphs:

i) The same number of vertices

ii) The same number of edges.

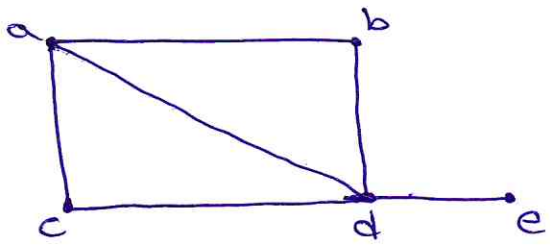
iii) The same number of connected components

iv) The same degree sequences.

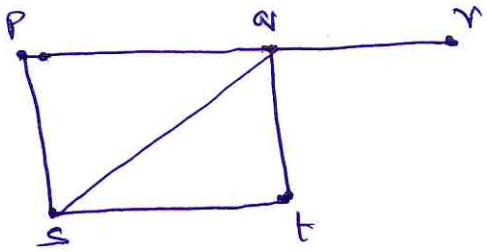
v) The same number of cycles of different lengths.

If any of the above criterion fails the graphs are not isomorphic. The above rules are only necessary but not sufficient for providing two graphs are isomorphic.

Show that the graphs G_1 & G_2 are isomorphic.



G_1



G_2

Graph G_1

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

Graph G_2.

Vertex	p	q	r	s	t
degree	2	4	1	3	2

No of vertices in graph G_1 is equal to the no. of vertices in graph G_2 i.e $|V_1| = 5$, $|V_2| = 5$.

No of edges in graph G_1 is equal to the no of edges in graph G_2 ie $|E_1| = 4$, $|E_2| = 4$.

∴ The two graphs of G_1 & G_2 are of same order & size.

degree sequence is d_1 (4, 3, 2, 2, 1)

d_2 (4, 3, 2, 2, 1).

We observe that both the degree sequence is also same.

Let us define $f: V_1 \to V_2$ as

$f(d)=q$, $f(a)=s$, $f(b)=p$, $f(c)=t$ or p, $f(e)=r$

The adjacency matrix of G_1 for the vertex ordering a, b, c, d, e.

$A_{G_1} =$

	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

The adjacency matrix of G_2 for the vertex ordering s, p, t, q, r is

$A_{G_2} =$

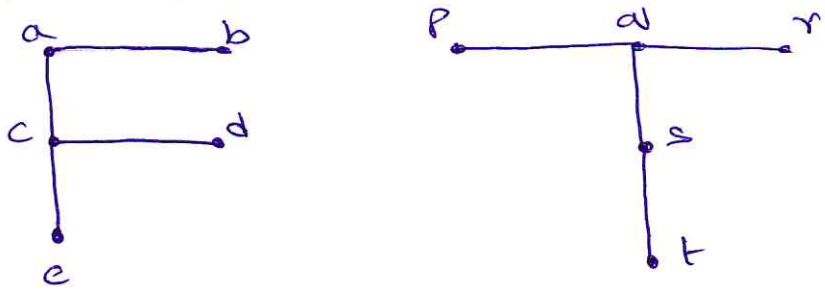
	s	p	t	q	r
s	0	1	1	1	0
p	1	0	0	1	0
t	1	0	0	1	0
q	1	1	1	0	1
r	0	0	0	1	0

$A_{G_1} = A_{G_2}$

Thus f is 1-1, onto & also preserves adjacency.

Hence G_1 & G_2 are isomorphic.

Show that the following two graphs are isomorphic



No of vertices in Graph G_1 is equal to the no of vertices in Graph G_2

$|V_1| = 5 \quad |V_2| = 5$

No of edges in Graph G_1 is equal to the no of edges in Graph G_2 i.e

$|E_1| = 4 \quad |E_2| = 4$

∴ The two graphs of G_1 & G_2 are of same order & size

G_1

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

G_2

vertex	p	q	r	s	t
degree	1	3	1	2	1

degree sequence of d_1 (3, 2, 1, 1, 1)

" " " d_2 (3, 2, 1, 1, 1).

We observe that both the degree sequence is also same.

Let us define $f: V_1 \to V_2$ as

$f(c) = q$, $f(a) = s$, $f(b) = t$, $f(d) = p$ (or) r

$f(e) = r$ (or) p

$A_{G_1} =$

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

$A_{G_2} =$

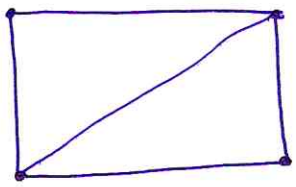
	s	t	q	p	r
s	0	1	1	0	0
t	1	0	0	0	0
p	1	0	0	1	1
q	0	0	1	0	0
r	0	0	1	0	0

$\therefore A_{G_1} = A_{G_2}$.

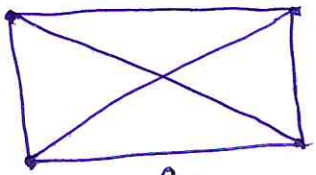
Thus f is one-one, onto & preserves adjacency.

The graph G_1 & G_2 are isomorphic.

Determine whether the following two graphs are isomorphic.



G_1



G_2

No of vertices in G_1 is same as no of vertices in G_2

$\therefore |V_1| = 4$, $|V_2| = 4$

$\Rightarrow |V_1| = |V_2|$.

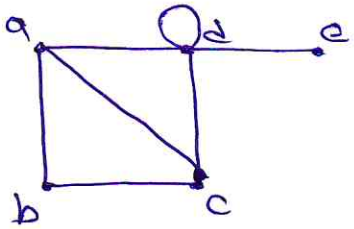
No of edges in G_1 i.e $|E_1| = 5$.

No of edges in G_2 i.e $|E_2| = 6$.

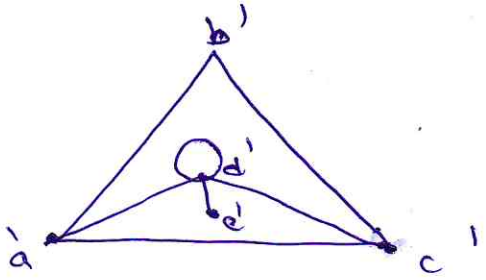
$\Rightarrow |E_1| \neq |E_2|$.

Hence G_1, G_2 are not isomorphic.

Show that two graphs are isomorphic.



G_1



No of vertices in G_1 is same as no of vertices in G_2 i.e $|V_1| = |V_2|$

No of edges in G_1 is same as no of edges in G_2

$|E_1| = |E_2|$.

$\Rightarrow$ The two graphs G_1 & G_2 are of same order & size

Graph G_1

Vertex	a	b	c	d	e
degree	3	2	3	5	1

Graph G_2

Vertex	a'	b'	c'	d'	e'
degree	3	2	3	5	1

degree sequence of $d_1 = \{5, 3, 3, 2, 1\}$

$d_2 = \{5, 3, 3, 2, 1\}$.

Let us define $f : V_1 \to V_2$

$f(a) = a'$ or c' , $f(d) = d'$

$f(e) = e'$, $f(b) = b'$, $f(c) = c'$ or a'

$A_{G_1} =$

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

$A_{G_2} =$

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

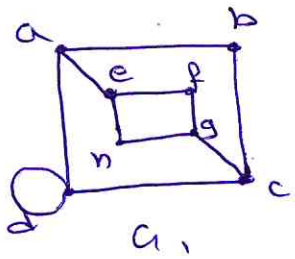
$A_{G_1} = A_{G_2}$

$\Rightarrow$ f preserves adjacency.

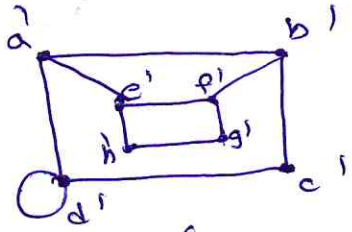
$\therefore$ f is one-one, onto & preserves adjacency.

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

Show that two graphs are not isomorphic.



G_1



G_2.

No of vertices in G_1 is same as no of vertices in G_2 i.e $|V_1| = 8$, $|V_2| = 8$ $\Rightarrow |V_1| = |V_2|$.

No of edges in G_1 is same as no of edges in G_2 i.e $|E_1| = 11$, $|E_2| = 11$, $\Rightarrow |E_1| = |E_2|$

Both graphs have of same order & size.

G_1

vertex	a	b	c	d	e	f	g	h
degree	3	2	3	4	3	2	3	2

G_2

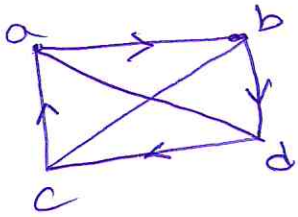
vertex	a'	b'	c'	d'	e'	f'	g'	h'
degree	3	3	2	4	3	3	2	2

i) degree sequence of $d_1 = \{4, 3, 3, 3, 3, 2, 2, 2\}$

$d_2 = \{4, 3, 3, 3, 3, 2, 2, 2\}$

Hamiltonian graphs:

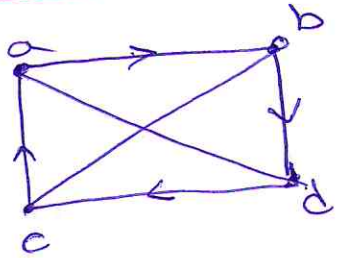
A path in a graph G is called a Hamiltonian path if it contains every vertex of G.



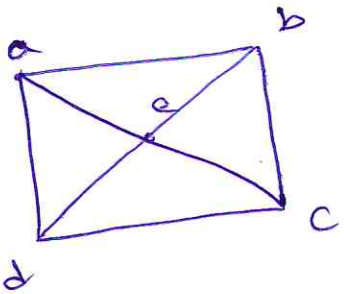
a - b - d - c.

It is a Hamiltonian path if it has all vertices.

A cycle in a graph G is called a Hamiltonian cycle if it contains every vertex of G.



a - b - d - c - a.



a - b - c - d - e - Hamiltonian path

a - b - c - d - e - a - Hamiltonian cycle.

Note :- An Eulerian cycle uses every edge exactly once but may repeat vertices, while a Hamiltonian cycle uses each vertex exactly once (except for the first & last) but may skip edges.

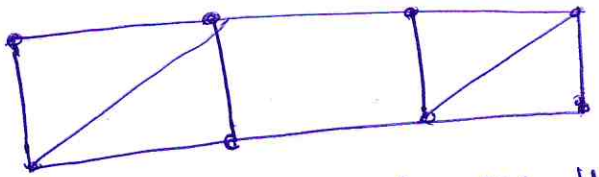
Rules for constructing Hamiltonian Paths & cycles in a graph G.

Rule 1:- If G is a graph with n vertices then a Hamiltonian cycle must contain exactly 'n' edges & a Hamiltonian Path must contain exactly 'n-1' edges.

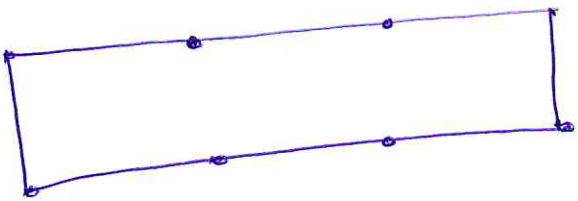
Rule 2:- If 'v' is a vertex of degree 2 then both edges incident on 'v' should contain in every Hamiltonian cycle & every Hamiltonian Path should contain atleast one edge incident on v.

once a Hamiltonian cycle we are constructing has passed through a vertex 'v', then all the unused edges incident on 'v' can be deleted.

Find the Hamiltonian cycle in the following graph.

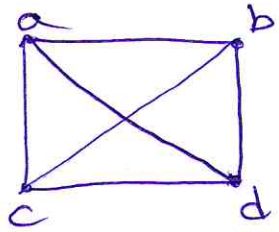


The Hamiltonian cycle in the graph is



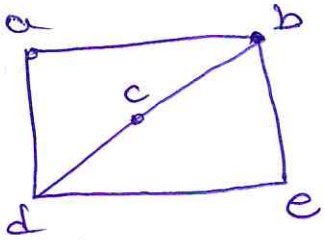
Determine which of the following are Hamiltonian graphs:

i)



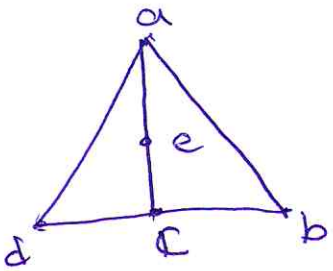
It is a Hamiltonian graph.
∴ it has an Hamiltonian cycle.
a-b-d-c-a (or) a-b-c-d-a

ii)



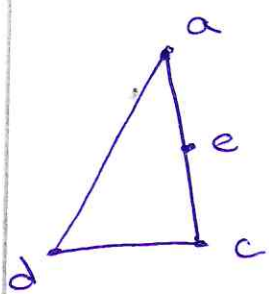
It is not Hamiltonian graph
∴ it has no Hamiltonian cycle, but it has a Hamiltonian path.
a-d-e-b-c.

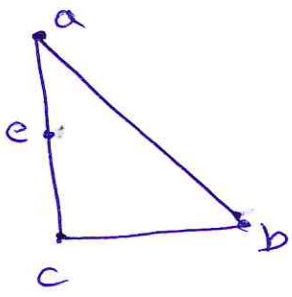
1) Show that following graph is not Hamiltonian cycle.

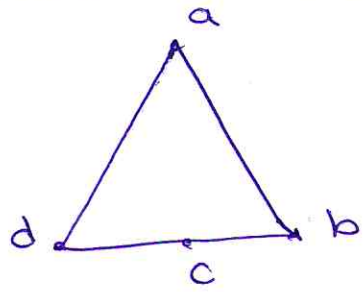


Deg(e) = 2, Deg(b) = 2
Deg(d) = 2, Deg(a) = 3
Deg(c) = 3

Here, there exists no Hamiltonian cycle.







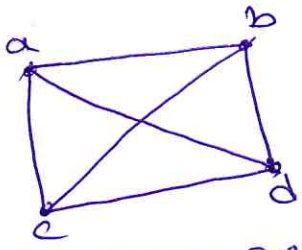
There are 3 possible cycles, but in all the cases we miss a vertex.

a-e-c-d-a is a cycle without a vertex b.

⇒ It is not Hamiltonian cycle.

III'y a-e-c-b-a & a-d-c-b-a are cycles but not Hamiltonian cycle.

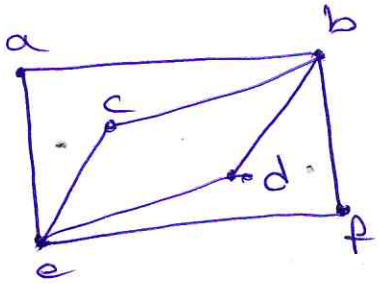
1) Give an example of a graph which is Hamiltonian but not eulerian.



G1 is Hamiltonian, since it has an Hamiltonian cycle a-b-d-c-a.

But G1 is not eulerian, since it has more than 2 vertices of odd degree (each vertex is of degree 3).

2) Give an example of a graph which is Eulerian but not Hamiltonian.



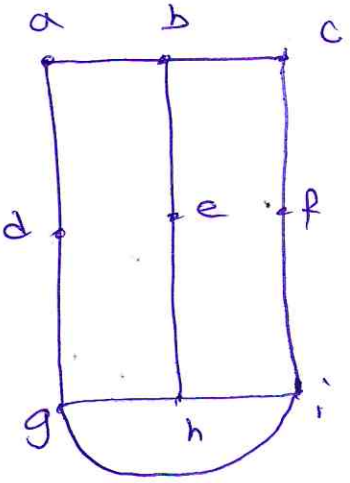
G1 is Eulerian, since each vertex is of even degree.

An eulerian cycle in G1 is

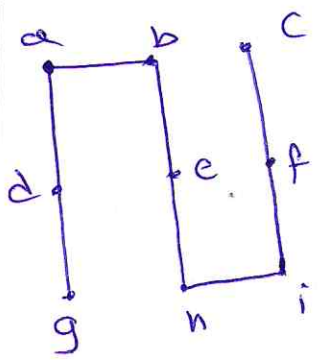
a-b-c-e-d-b-f-e-a

But G is not Hamiltonian. since we have 6 vertices in which 4 are two degree vertices. Hence the 8 edges must be included in every Hamiltonian cycle but a Hamiltonian cycle of G should contain only 6 edges.

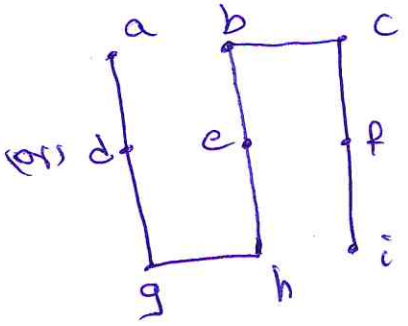
Show that the following graph has no Hamiltonian cycle. But the graph has a Hamiltonian path.



The degree of vertices d, e, f is 2



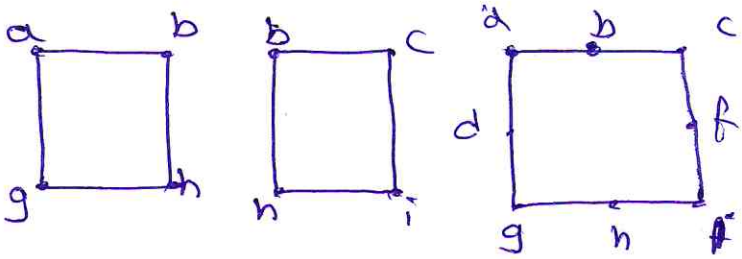
(or)



No Hamiltonian cycle exists in G

But a Hamiltonian path a-d-g-h-e-b-c-f-i exists in G.

Possible cycles are

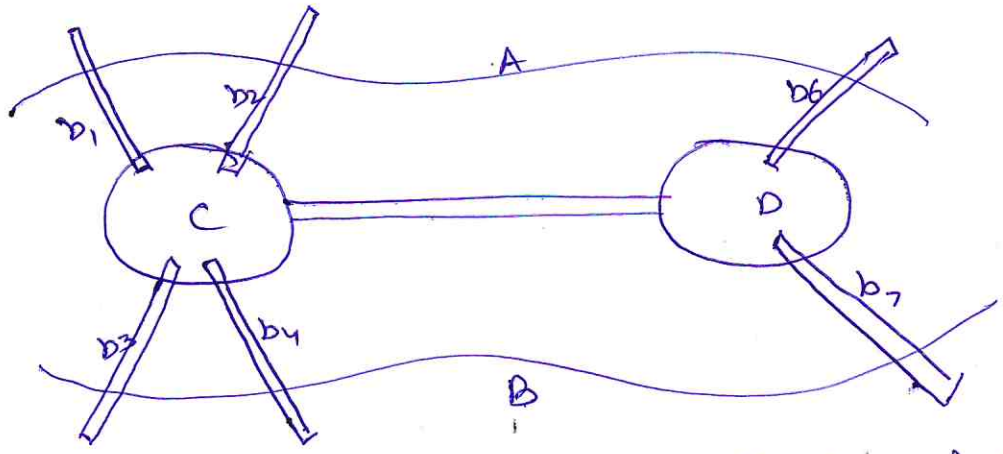


But these are not Hamiltonian cycle.

Explain the Konigsberg bridge problem?

The Konigsberg bridge problem is the best known example in graph theory.

Two islands C & D formed by Pregal river in the city Konigsberg were connected to each other & to the banks A & B with seven bridges as shown in the figure.



C & D are islands. A & B are Banks of Pregal river. $b_1, b_2, b_3, \ldots b_7$ are bridges.

The problem was to start at any of the 4 land areas of the city A, B, C or D walk over each of the 7 bridges exactly once & return to the starting point.

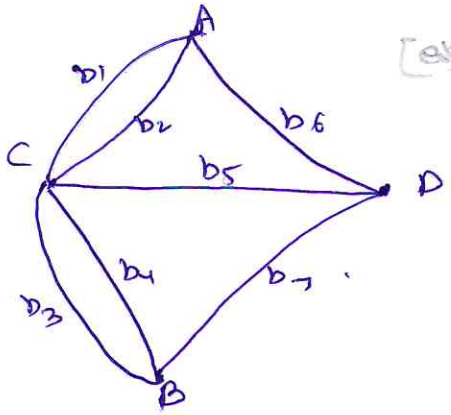
Euler represented this by means of a graph where the vertices represents the land areas & the edges represent the bridges as shown

in the figure.

Euler proved that a solution for this problem doesn't exist.

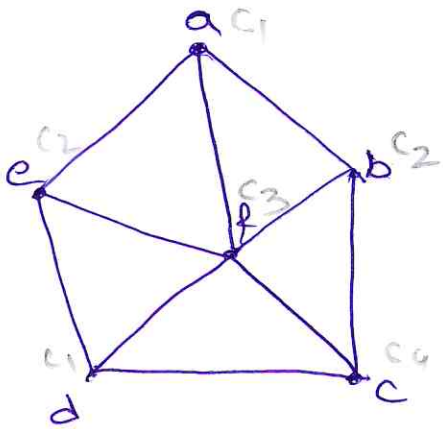
∴ not all vertices are of even degree, the graph is not an euler graph. Thus it is not possible to walk over each of the seven bridges exactly once & return to the starting point.

Euler's graphical representation.



[edge exactly once but may repeat vertices].

Find the chromatic number of the following graph.



vertex	a	b	c	d	e	f
degree	3	3	3	3	3	5

∴ deg(f) = 5 & all other vertices have degree 3.

$\Delta(G) = 5$.

$\Delta(G)$ = largest degree of any vertex of G.

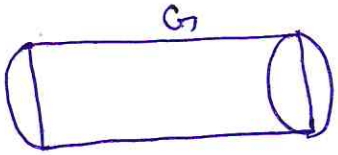
$\chi(G) \leq 1 + \Delta(G) = 6$.

$\therefore \chi(G) = 4$

MULTIGRAPHS & EULER CIRCUITS

A graph containing parallel edges is called a multigraph.

Eg:-



G is a multigraph.

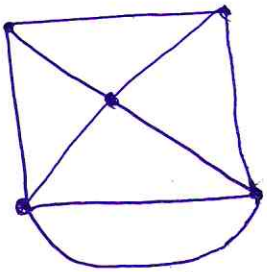
An Euler Path in a multigraph is a path that includes each edge of the multigraph exactly once & intersects each vertex of the multigraph at least once.

A multigraph is said to be traversable if it has an Euler Path

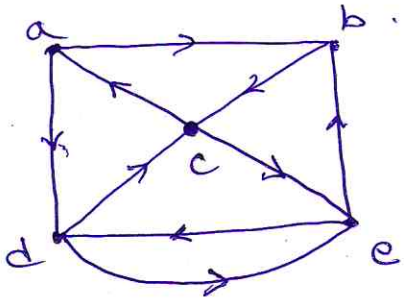
An Euler circuit is an euler path whose end points are identical.

A multigraph is said to be an Eulerian multigraph if it has an euler circuit.

Give an example of a traversable multigraph.



G_1

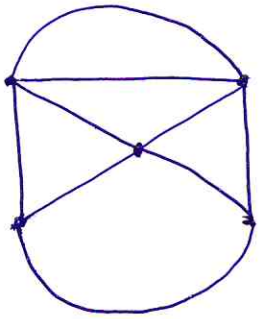


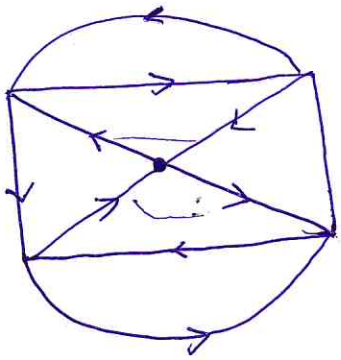
Euler Path in G_1

Euler Path - a - b - c - a - d - c - e - d - e - b.

Euler circuit - a - b - c - a - d - c - e - d - e - b - a

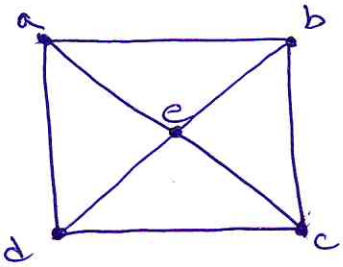
Give an example of an Eulerian graph.

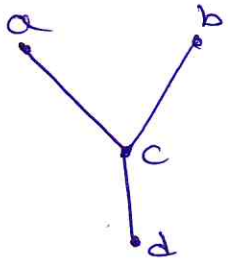




Both G_1 & G_2 of above are traversable graphs but G_1 is not an eulerian graph.

Give an example which is not traversable.



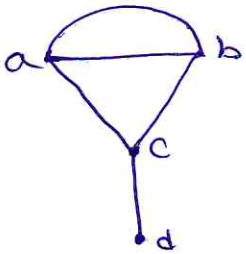


are not traversable

A graph with more than two odd degree vertices is not traversable.

$d(a) = 3$, $d(b) = 3$, $d(c) = 3$,

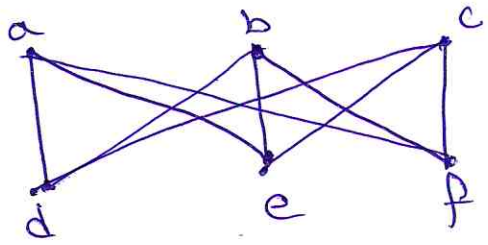
$d(d) = 1$

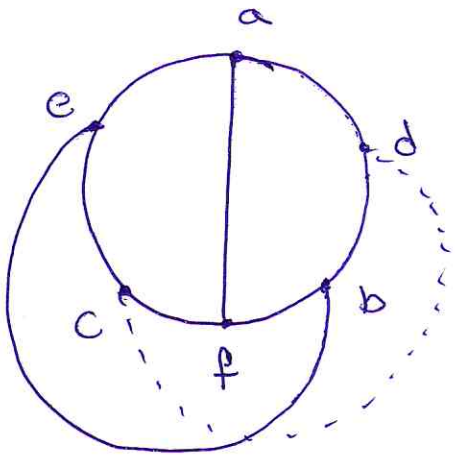


More than 2 odd degree vertices. so it is not traversable.

A connected graph G is an Eulerian iff each vertex of G has even degree.

Show that $K_{3,3}$ is non-planar





It is not possible to draw the edge [c,d] without cross over.

$\therefore K_{3,3}$ is non planar

Euler's formula :-

If G is a connected plane graph, then

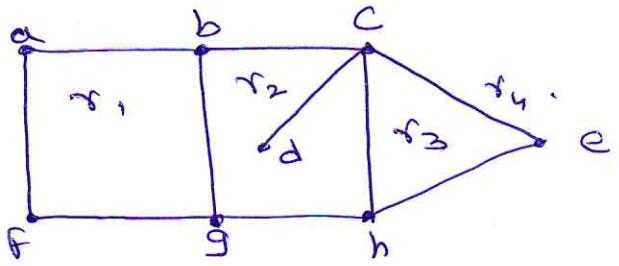
$|V| - |E| + |R| = 2$

where $|V|, |E|, |R|$ are respectively the number of vertices, edges & regions of G

A plane graph divides the plane into connected areas called regions or faces. Each plane graph G determined a region of infinite area called the exterior region of G.

The degree of a region is the length of the closed path bordering the region.

Find the degree of the regions of the graph.



$R(G) = \{r_1, r_2, r_3, r_4\}$.

$\deg(r_1) = 4$, $\deg(r_2) = 6$, $\deg(r_3) = 3$,

$\deg(r_4) = 7$.

Region	border	degree
r_1	a-b-g-f-a	4
r_2	b-c-d-c-h-g-b	6
r_3	c-e-h-c	3
r_4	a-b-c-e-h-g-f-a	7.

Note:- $V(G)$ = The set of all vertices of G

$E(G)$ = The set of all edges of G.

$R(G)$ = The set of all regions of plane graph G.

Theorem :. If G is a plane graph, then the sum of the degrees of the regions determined by G is twice the number of edges of G.

i.e $\sum_{r \in R(G)} \deg(r) = 2|E|$.

where $|E|$ is the number of edges of G.

From above figure

$|E| = 10$

$$\sum_{r \in R(G)} \deg(r) = \deg(r_1) + \deg(r_2) + \deg(r_3) + \deg(r_4)$$

$$= 4 + 6 + 3 + 7$$

$$= 20 = 2 \times 10 = 2|E|.$$

Find the number of edges in a plane graph G containing 4 regions each of degree 3.

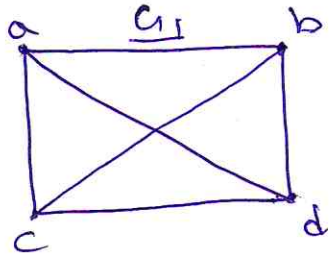
By sum of degrees theorem

$$\sum \deg(r) = 2|E|$$

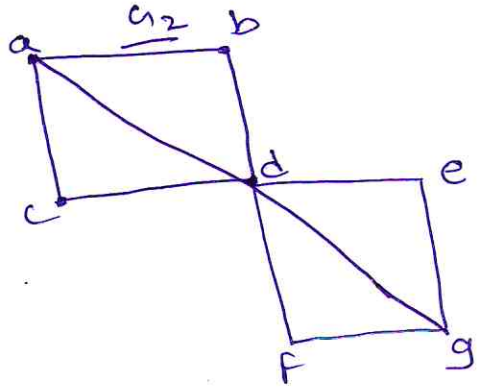
$$4 \times 3 = 2|E|$$

$$\Rightarrow |E| = 6$$

Which of the following graphs are traversable, Eulerian or neither.

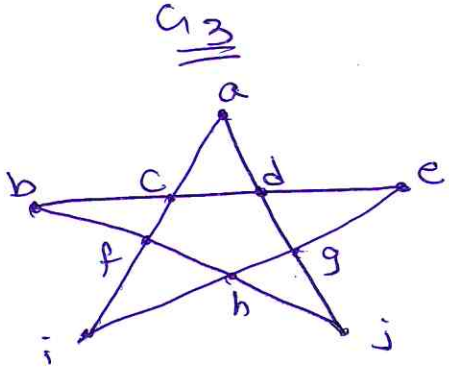


Each vertex is of degree 3 i.e it has more than 2 vertices of odd degree. Hence it is neither traversable nor Eulerian.



It has exactly two vertices 'a' & 'g' of odd degree hence it is traversable but not eulerian.

A traversable trial in G_2 is a-b-d-c-a-d-e-g-f-d-g.

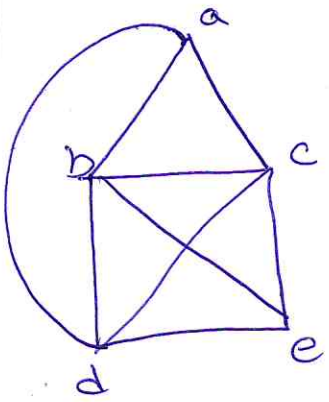


Each vertex is of degree even. Hence it is both ~~traversable &~~ Eulerian.

An Euler circuit in G_3 is

a - c - f - i - h - g - e - d - c - b - f - h - j - g - d - a.

Find which of the following multigraphs have Euler paths, circuits or neither



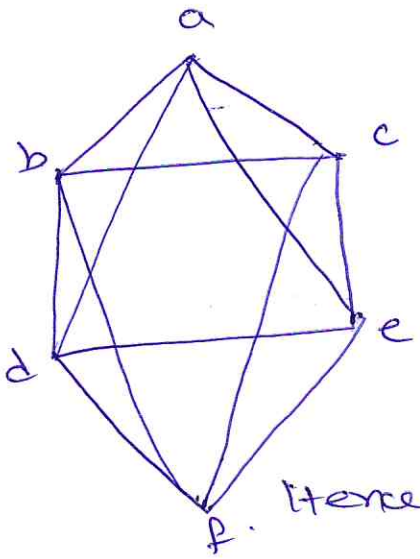
vertex	a	b	c	d	e
degree	3	4	4	4	3

It has two vertices are of odd degree. Hence it is traversable but not eulerian i.e we can find euler path (between a & e) but there is no euler circuit.

Euler path is a-b-c-a-d-b-e-c-d-e.

(or)

a-d-b-a-c-b-e-d-c-e.



vertex	a	b	c	d	e	f
degree	4	4	4	4	4	4

Each vertex is of even degree. Hence it is both traversable & eulerian.

An euler path or an euler circuit is

a-b-d-f-e-c-b-f-c-a-d-e-a.

(or) a-b-c-a-d-b-f-e-c-f-d-e-a.

a-b-c-e-d-f-e-a-d-b-f-c-a.

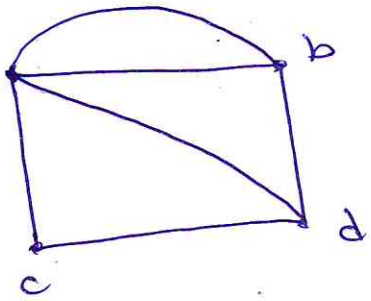
Subgraph:-

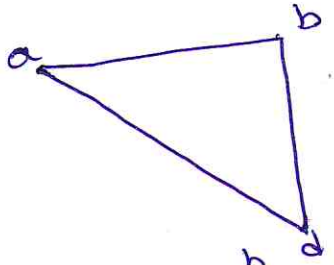
A graph $H = (V_1, E_1)$ is called a subgraph of $G = (V, E)$ if and only if $V_1 \subseteq V$ & $E_1 \subseteq E$

Spanning subgraph:

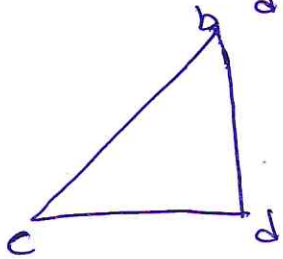
A graph $H = (V_1, E_1)$ is called a spanning subgraph of $G = (V, E)$ if and only if $V_1 = V$ & $E_1 \subseteq E$.

Eg):-

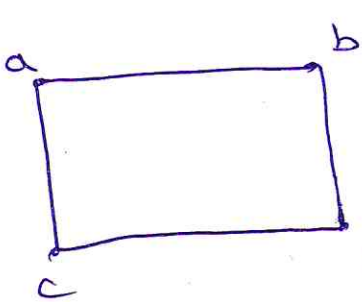


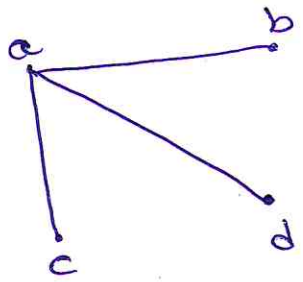


is a subgraph of G



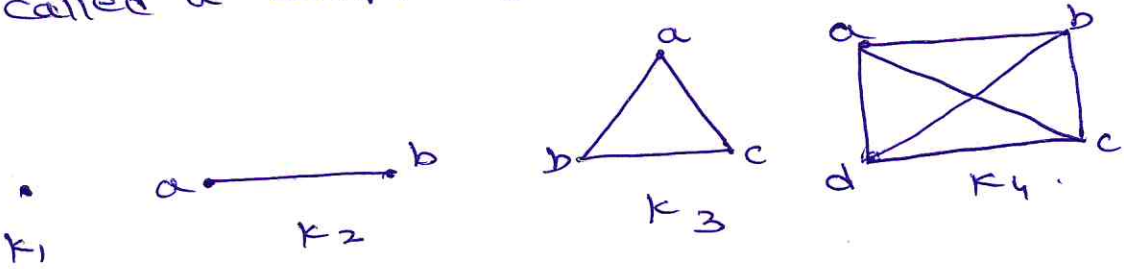
is not a subgraph of G





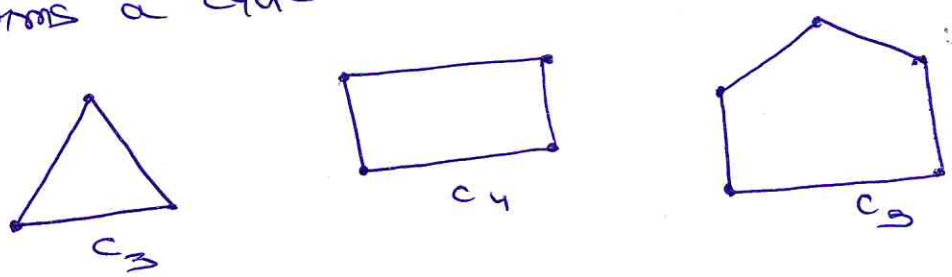
are spanning subgraph of G

Complete graph :- A graph in which every vertex is connected to every other vertex is called a complete graph.

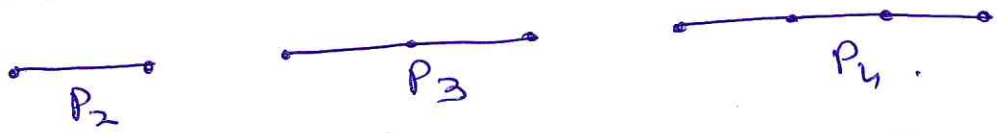


The number of edges in K_n is $\frac{n(n-1)}{2}$.

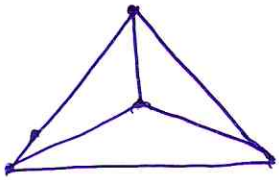
cycle graph:- A graph in which all the edges forms a cycle is called a cycle graph.



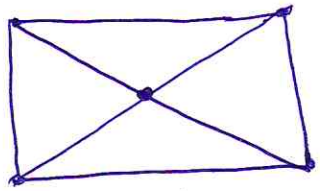
Path graph :- A graph whose edges forms a path is called a path graph.



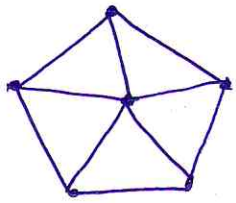
wheel graph :- A graph obtained from a cycle graph by joining a single new vertex (the hub) to each vertex of the cycle is called a wheel graph.



W_4

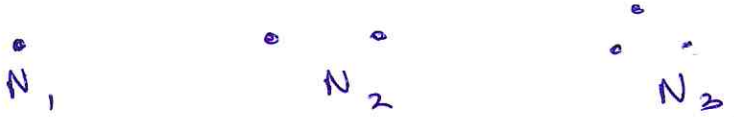


W_5



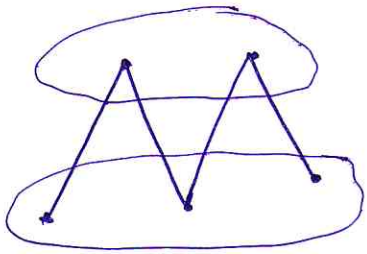
W_6.

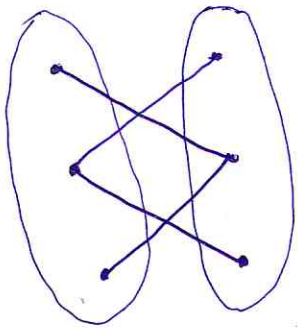
Null graph :- A graph with no edges is called a null graph.



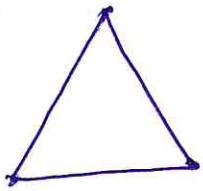
N_1 N_2 N_3.

Bipartite graph :- A graph in which the set of vertices can be partitioned into two sets M & N in such a way that each edge joins a vertex in M to a vertex in N is called a bipartite graph.





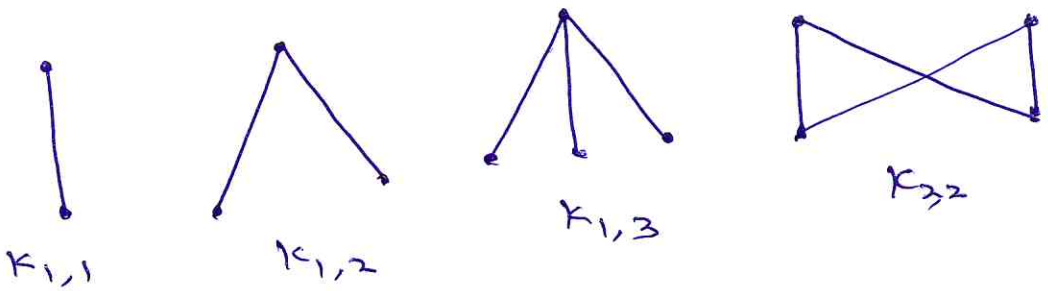
are bipartite graphs.

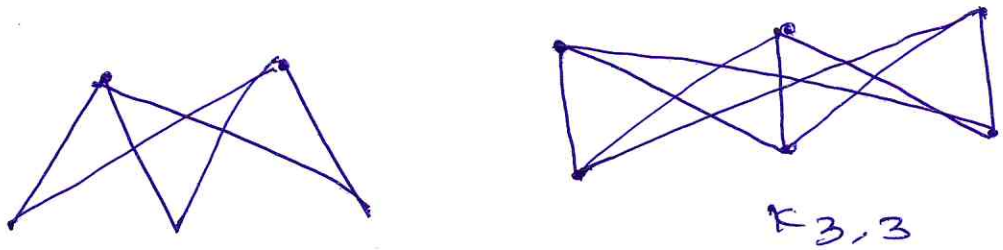


is not a bipartite graph.

Complete bipartite graph:-

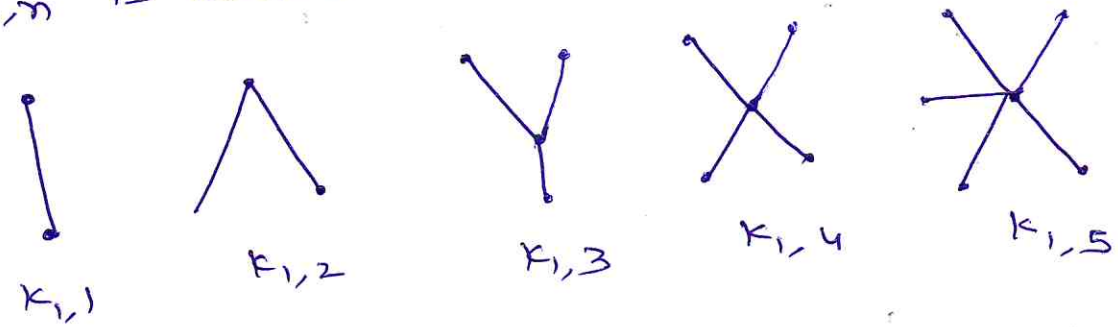
A bipartite graph in which every vertex of M is adjacent to every vertex of N is called a complete bipartite graph.





$K_{m,n}$ has $m+n$ vertices & mn edges.

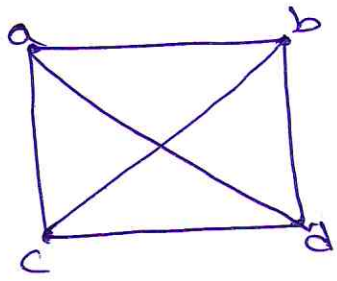
Star graph:- A complete bipartite graph $K_{1,n}$ is called a star graph.



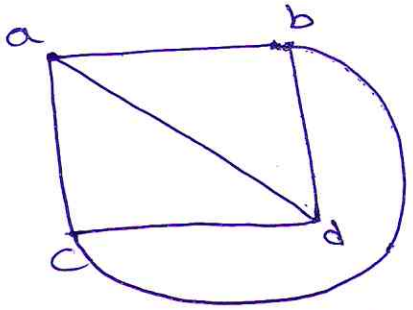
Planar Graphs:

A Graph G is said to be planar if it can be drawn in a plane so that its edges do not cross over.

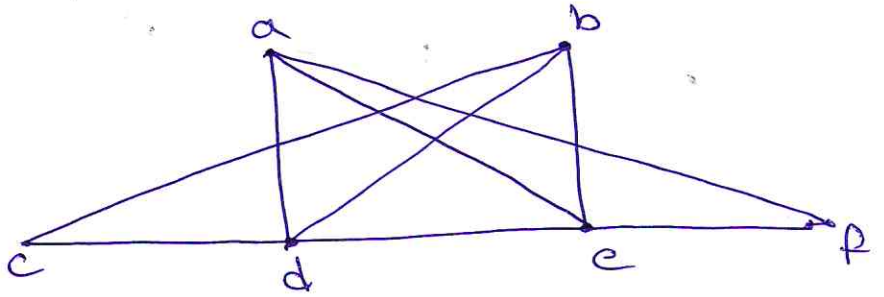
Eg:-



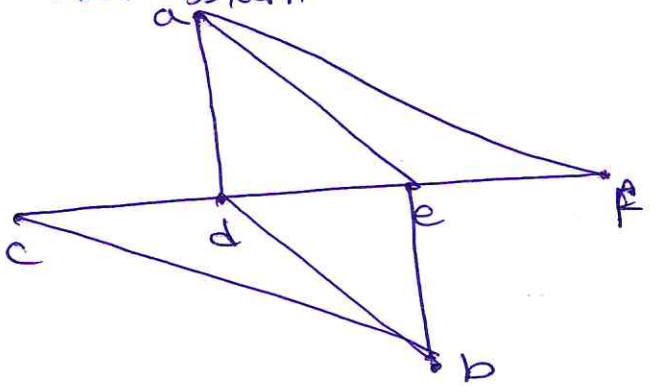
It can be written as.



Show that the following graph G is planar & draw plane graph of it.



It is planar graph.



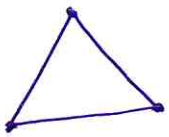
Complete graph :- A graph with 'n' vertices so that each of the 'n' vertices are adjacent to each of the other (n-1) vertices is called a complete graph it is denoted by K_n.

Show that K_n is planar for $n = 1, 2, 3, 4$.

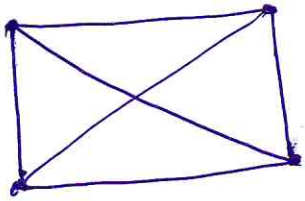
K_1 is planar

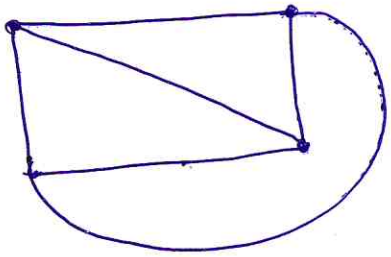
K_2 is planar

K_3 is planar



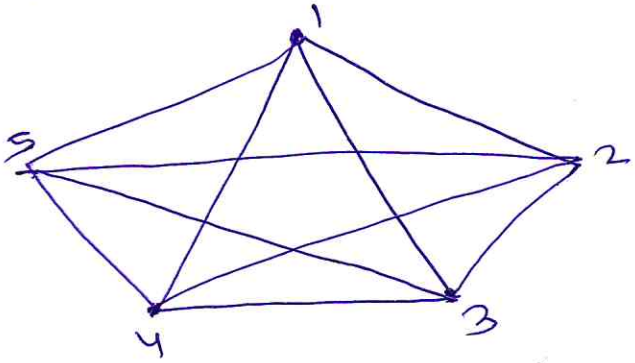
K_4 is planar



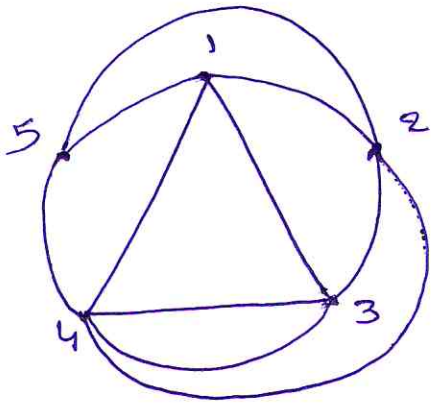


Show that K_5 is not planar.

The graph K_5 is



The number of edges in K_n is $\frac{n(n-1)}{2}$.



It is not possible to draw {3, 5}. since. it crosses {1, 4}. Thus, it is impossible to draw the edge {3, 5} without cross over. Thus K_5 is non-Planar.

TREES

Introduction :-

The concept of a tree was initially introduced to study the chemical composition by cayley in the year 1857. Trees form one of the most widely used & extensively studied subclass of graphs. Many of the applications in graph theory directly or indirectly involve trees. Trees are useful in describing any structure which involves a hierarchy. In computer science, trees are useful in organizing & storing data in a database. Hence, trees play a vital role in graph theory.

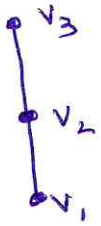
Definition :- A connected graph with no cycle is known as a tree. Its edges are called branches.

Any graph without cycles may also be called a forest. Hence, trees are components of a forest.

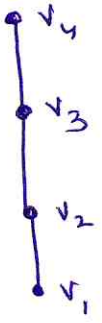
Examples of trees with atmost five vertices.

one vertex

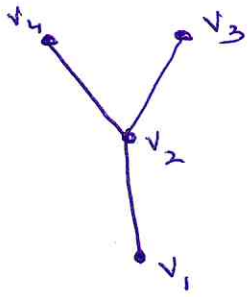
Two vertices



Three vertices

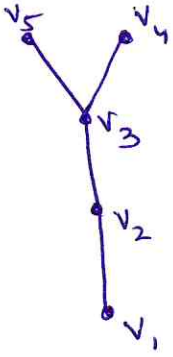


Four vertices

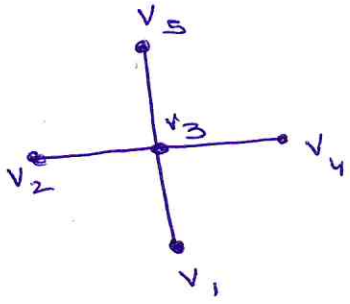


Four vertices.

Five vertices



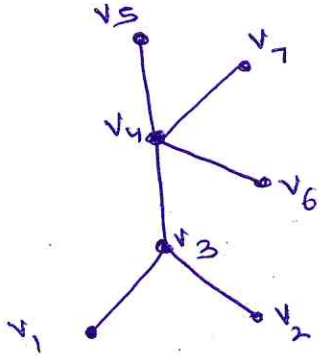
Five vertices

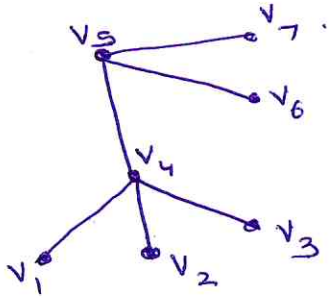


Five vertices.

A tree with only one vertex is called a trivial tree, otherwise T is called a non-trivial tree.

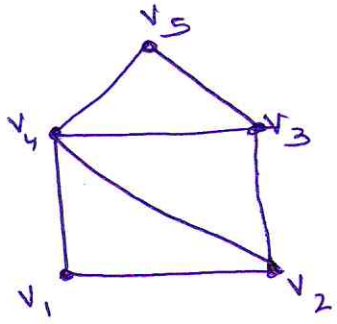
Consider the graphs shown below & determine whether they are trees.

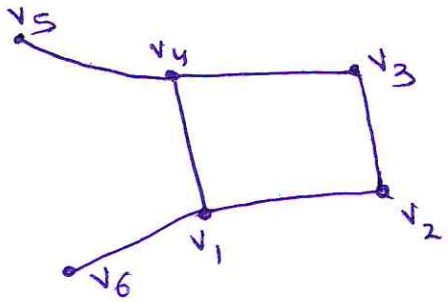




The given graphs are connected as they have no cycles. Hence, both the graphs are trees.

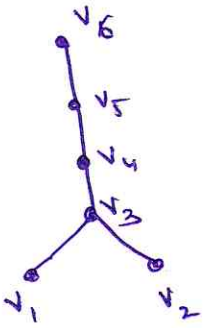
Consider the graphs shown below & determine whether they are trees.

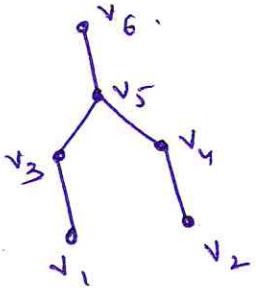


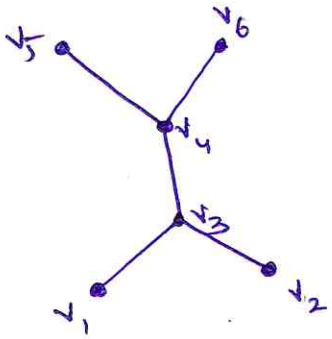


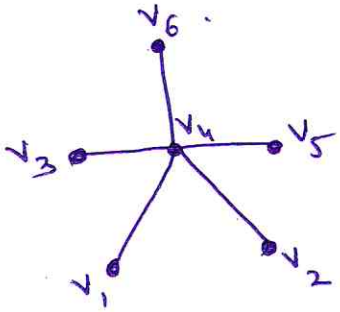
Solution :- ∵ each of the graphs are connected. However, ~~but~~ Both the graphs have a cycle. ∴ none of the graphs is a tree.

The trees on 6 vertices





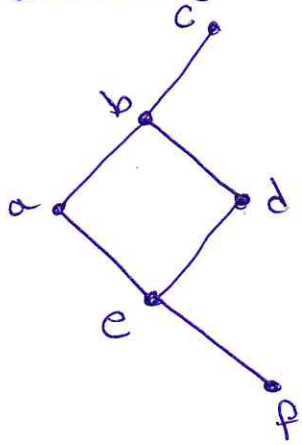




connected graph- If there is a path from any point to any other point in the graph.

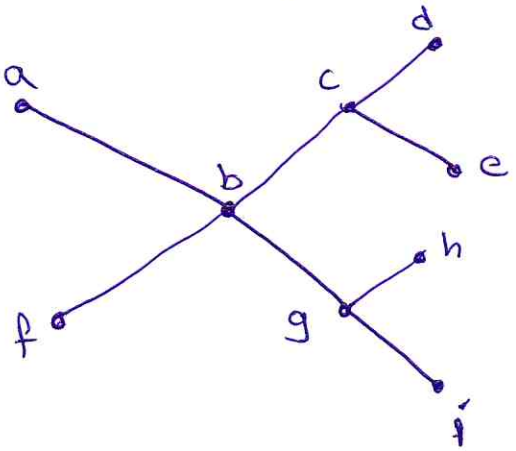
Identify which of the following are trees:

i)



It is a connected graph & it has a cycle a-b-d-e-a. Hence it is not a tree.

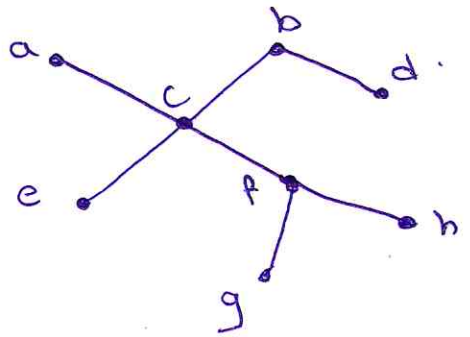
ii)



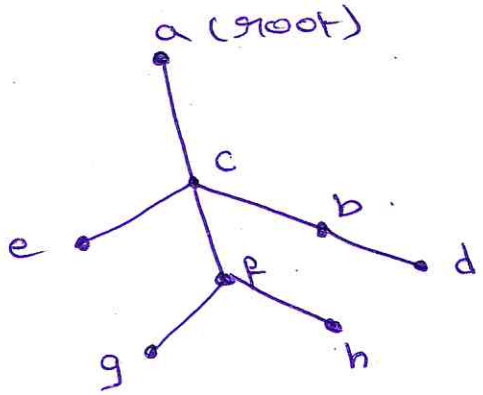
It is a tree, since it is a connected graph & has no cycles.

A rooted tree is a tree in which a particular vertex is disignated as the root.

Eg:- Let us consider the following tree.

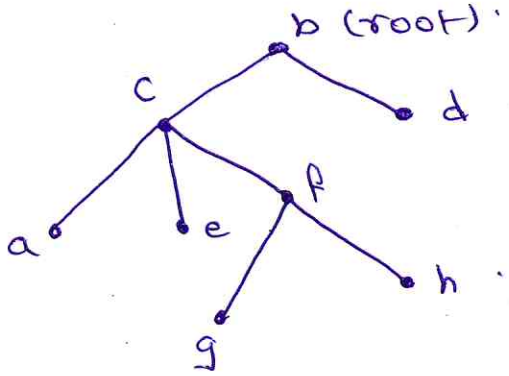


Let 'a' be designated as root. Then the rooted tree can be written as.



Parent	children
a	c
c	e, b, f
f	g, h.
b	d.

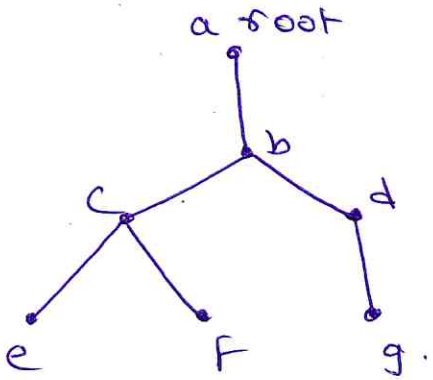
Let 'b' be designated as root. Then the rooted tree can be written as



Parent	children
b	c, d
c	a, e, f
f	g, h

A rooted tree is a directed tree if there is a root from which there is a directed path to each vertex.

Eg:- The following graph is a directed tree.



The level of a vertex is defined as the number of edges along the unique path between it & the root. The level of the root is defined as 0. The vertices immediately under the root are said to in level 1, & so on.

The height of a rooted tree is defined as the maximum level to any vertex of the tree. The depth of a vertex 'v' in a tree is the length of the path from the root to v.

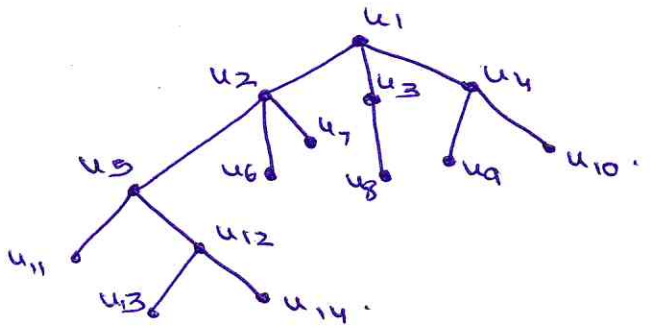
Let v be any internal vertex of a rooted tree T. The children of v are all those vertices that are adjacent to v & are one level farther away from the root than v.

If w is a child of v, then v is called the parent of w. Two vertices that are both children of the same parent then they are called ~~sibb~~ siblings.

If a vertex v of T has no children, then v is called a leaf or a terminal vertex. If v is not a terminal vertex then it is called an internal vertex.

Let T be a rooted tree. Let u & v be distinct vertices in T. Then, v is said to be a descendant of u if u & v are on the unique path from the root of T to v & u appears before v on this path.

Consider the tree shown below. & describe its different components.



Solution:- In the rooted tree T,

the level of vertices u_2, u_3, u_4 is 1

the level of vertices u_5, u_6, u_7, u_8, u_9, & u_{10} is 2.

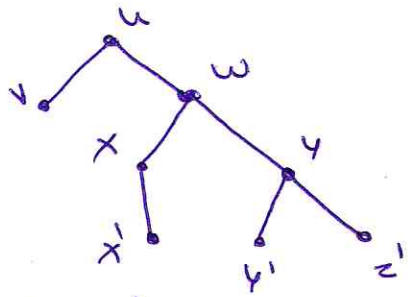
the level of vertices u_{11}, u_{12} is 3.

the level of vertices u_{13} & u_{14} is 4.

The vertices $u_6, u_7, u_8, u_9, u_{10}, u_{11}, u_{13}$ & u_{14} are leaves. All other vertices are internal vertices.

The vertices $u_5, u_6, u_7, u_{11}, u_{12}, u_{13}$ & u_{14} are descendants of u_2.

consider the rooted tree T given below.



i) what is the root of T?

Vertex 'u' is the only vertex distinguished from the others & is located at the top of the tree. Therefore, u is the root.

ii) Find the leaves & the internal vertices of T.

Leaves are those vertices that have no children. Hence, the leaves are v, x', y' & z'. The internal vertices are w, x & y.

iii) what are the level of w & z.

The levels of w & z are 1 & 2 respectively.

iv) Find the children of w & y

The children of w are x & y.

The children of y are y' & z'

v) Find the descendants of the vertices u & w

The descendants of u are v, w, x, y, y', z'

The descendants of w are x, y, x', y'.

Spanning tree :-

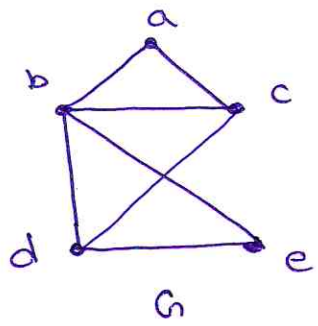
A subgraph H of a graph G is called a spanning tree of G if

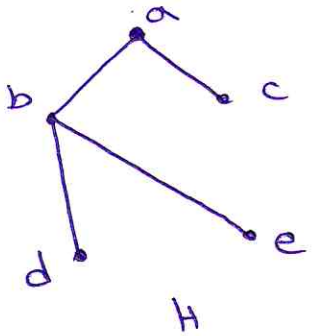
i) H is a tree, &

ii) H contains all the vertices of G.

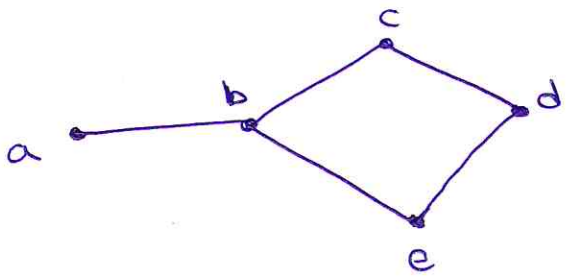
A spanning tree that is a directed tree is called a directed spanning tree of G.

Eg :- A graph G & its spanning tree H is shown below.





1) Find all spanning trees of the following graph G.

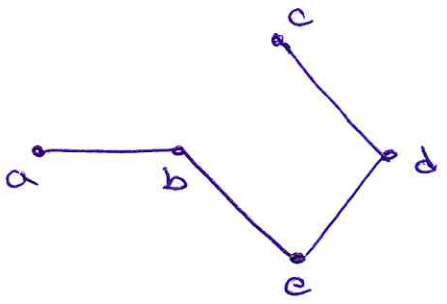


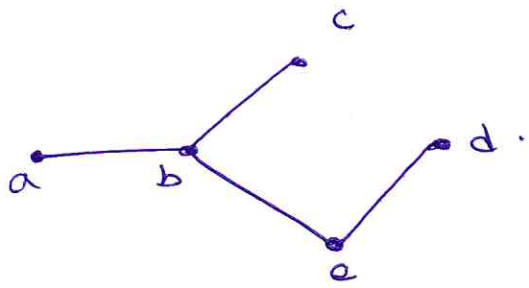
Solution :- G has 5 vertices & 5 edges.

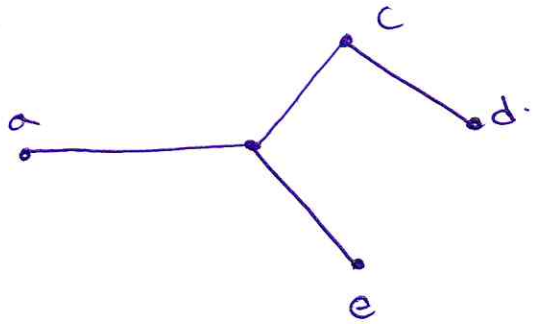
∴ It's spanning tree should contain (n-1) edges.

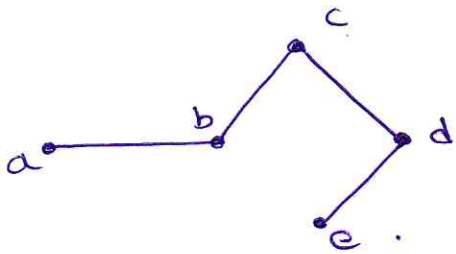
i.e 5-1 = 4 edges. We have to remove one edge which breaks the cycle b-c-d-e-b.

Thus, the spanning trees are:

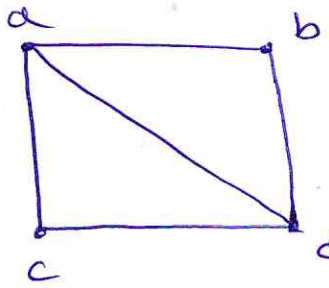








2) Draw all the spanning trees of the graph G.

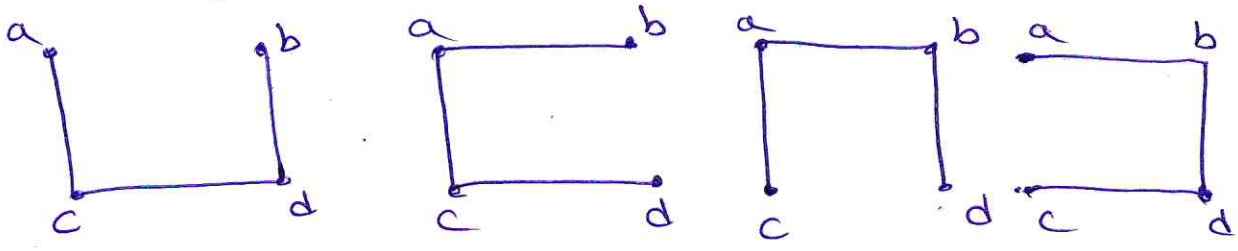


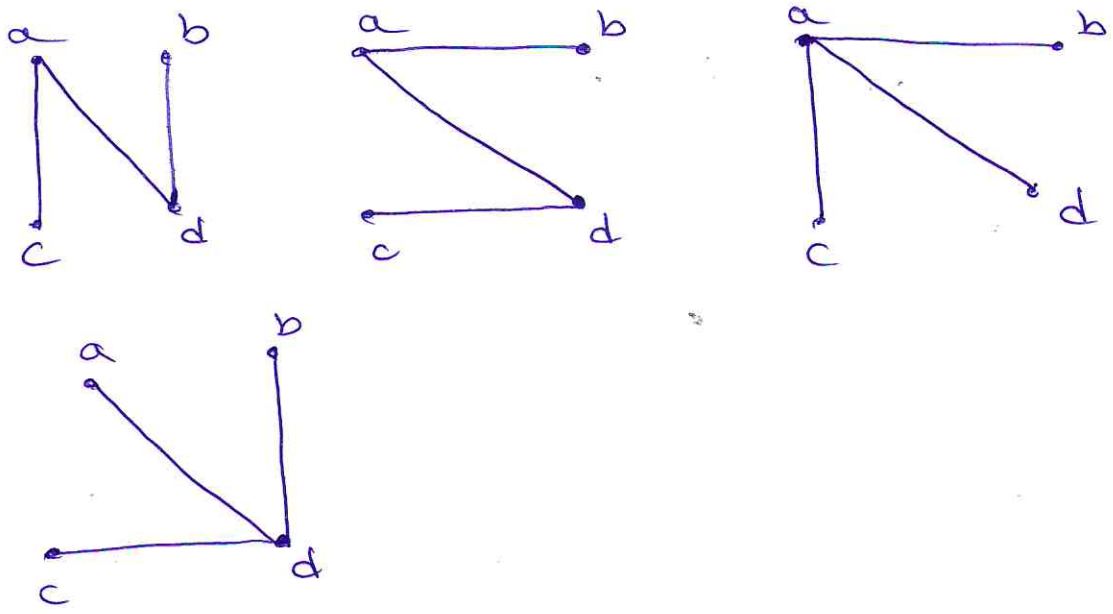
We have 4 vertices & 5 edges in G.

A spanning tree of G should contain $(n-1) = (4-1) = 3$ edges.

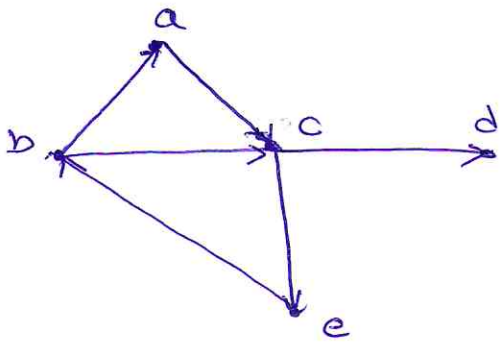
∴ we need to remove 2 edges from G such that the resulting graph has no cycles.

Following are the possible spanning trees of G.





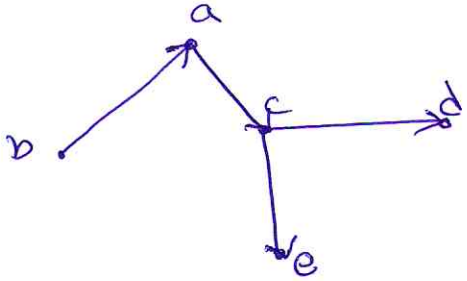
3) Find all directed spanning trees with root 'b' of the following directed graph:

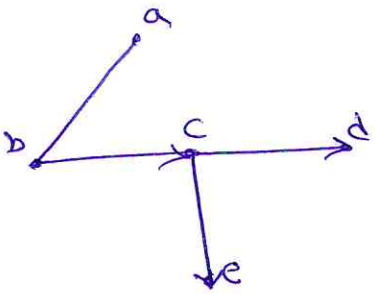


The graph has 5 vertices.

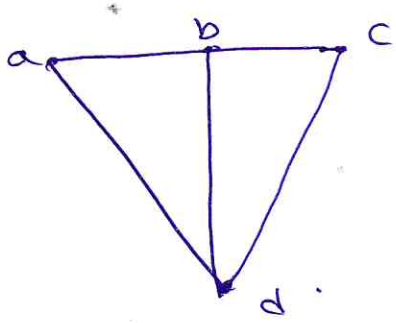
Its spanning tree should contain 5-1 = 4 edges.

Following are the directed spanning trees with root 'b':

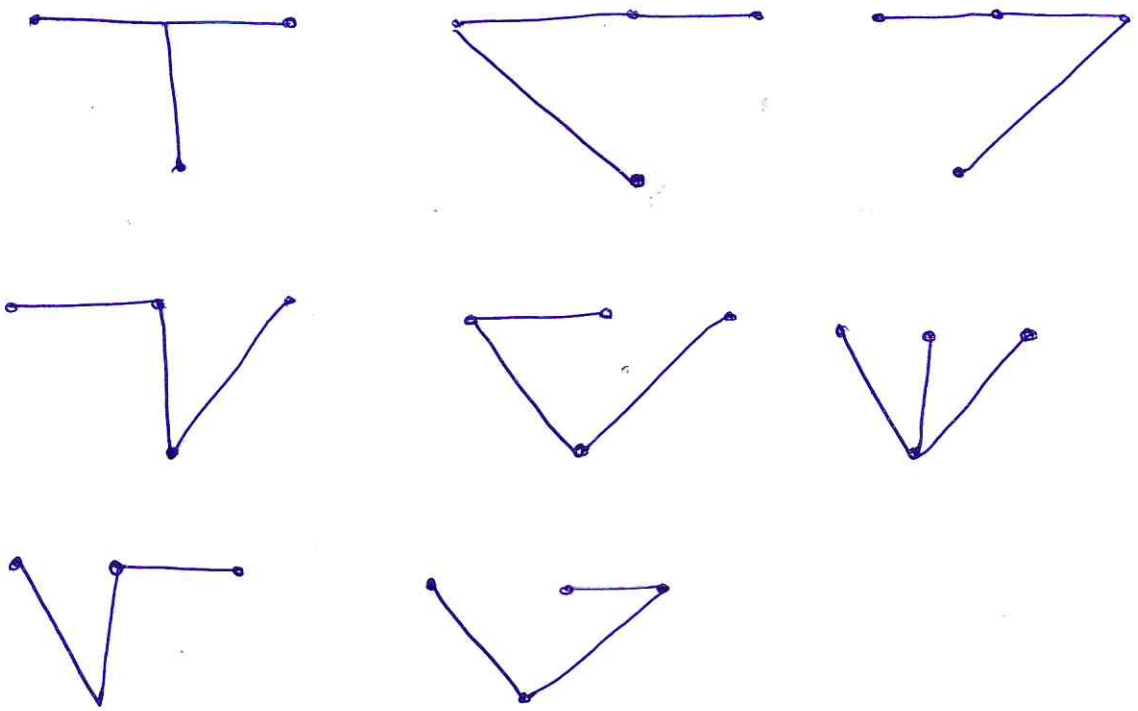




4) Find all spanning trees of a graph.



The graph G has 4 vertices & hence each spanning tree must have $(n-1) = 4-1 = 3$ edges. Thus, each tree can be obtained by deleting two of the five edges of G.



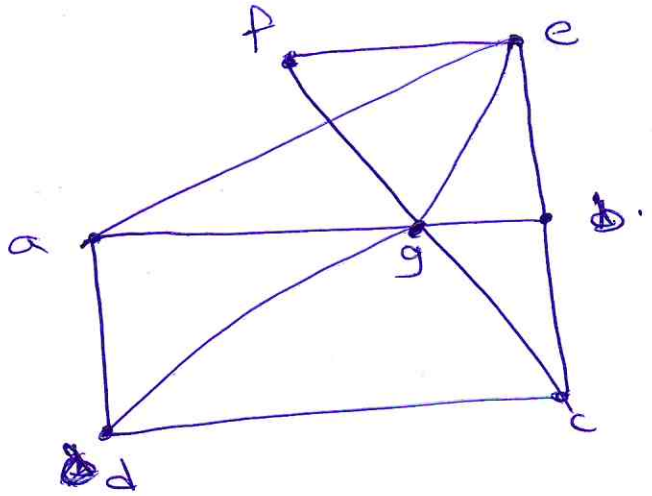
Algorithms for constructing spanning trees.

To find spanning trees in graphs we remove edges which destroy the cycles in the graphs. But such an algorithm is not very efficient. We have two efficient algorithms for finding a spanning tree of a connected graph. They are Breadth-First search (BFS) & Depth-First search (DFS). In these algorithms spanning trees are constructed by adding edges successively.

Breadth-First search Method (BFS)

We start at any vertex, taken as a root, & find edges incident on it. The vertices added at this stage become the vertices at level 1 in the spanning tree, arbitrarily order them. Next for each vertex at level 1, taken in order, 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.

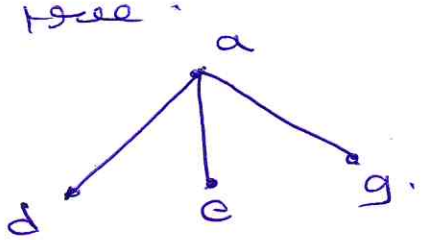
1) Use BFS to find a spanning tree for the graph shown in the following figure with the vertex ordering abcdefg & gfedcba.



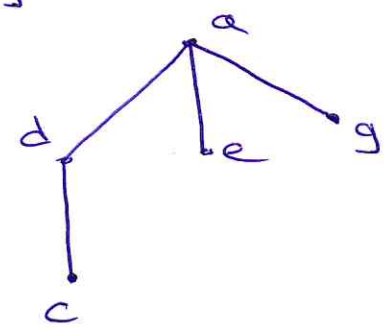
I - Let the vertex ordering be abcdefg.

i) Start with the vertex 'a' as a root.

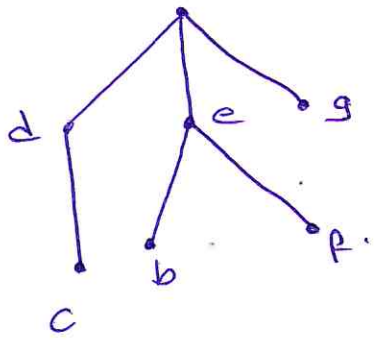
ii) Add all the edges incident on 'a'. They are {a,d}, {a,e}, {a,g}. The vertices d, e, g are at level 1 in the tree.



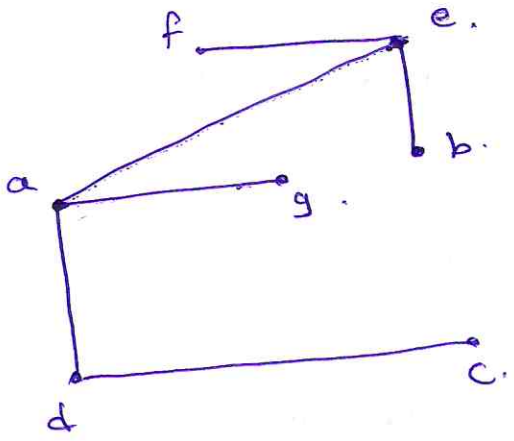
iii) Add d to {d,c} but not {d,g} as it forms a cycle.



iv) at 'e' add {e,b} & {e,f} but not {e,g} as it forms a cycle.



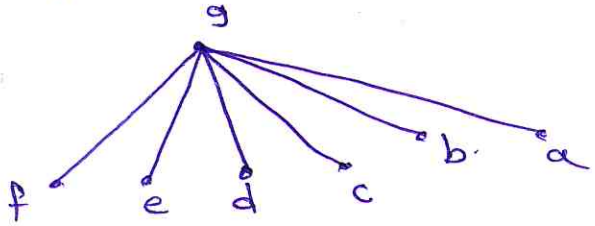
It contains all the vertices. Here it is a spanning tree. we can also draw it as a a spanning tree as follows.



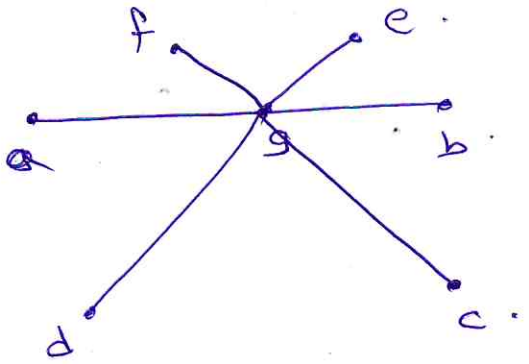
II let the vertex ordering be gfedcba.

i) we start with the vertex 'g' as root.

ii) Add all the edges incident on 'g'. They are f,e,d,c,b,a



Since it contains all the vertices, it is a spanning tree. This can be drawn as



~~Bdst~~

Breadth-First search Algorithm.

Input - A connected graph G with vertices labelled $v_1, v_2 \ldots v_n$.

Out Put - A spanning tree T for G.

Method:

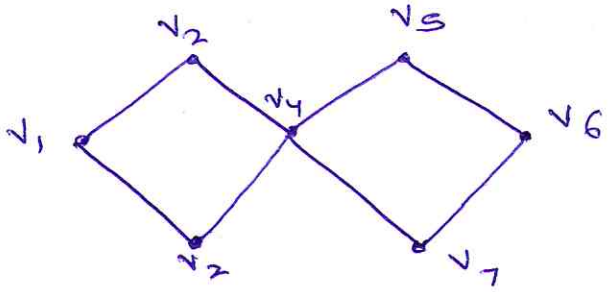
1) (Start) Let v_1 be the root of T. Form the set $V = \{v_1\}$.

2) (Add new edges) Consider the vertices V in order consistent with the original labelling. Then for each vertex 'x' of V, add the edge $\{x, v_k\}$ to T where k is the minimum index such that adding the edge $\{x, v_k\}$ to T does not produce a cycle. If no edge can be added, then stop. T is spanning tree for G. After all

the vertices of V have been considered in order, go to step 3.

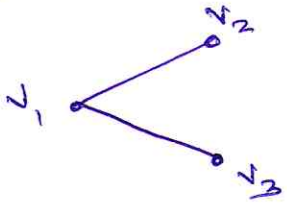
3) (update V) Replace V by all the children 'v' in T of the vertices x of V where the edges {x, v} were added in step 2. Go back & repeat step 2 for the new set V.

2) Find a spanning tree of a graph G, by using BFS.

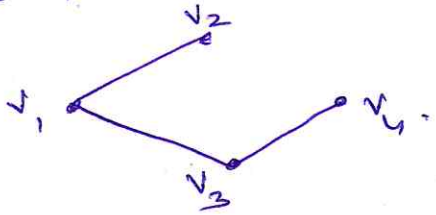


start with the vertex v_1 as a root.

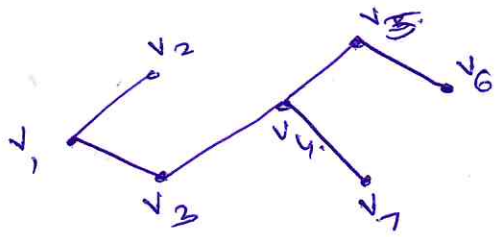
Add all the edges incident on v_1. They are $\{v_1, v_2\}$, $\{v_1, v_3\}$. The two vertices v_2 & v_3 are in level 1 in the tree.



Add v_3 to $\{v_3, v_4\}$



at v_4 add $\{v_4, v_7\}$ & $\{v_4, v_5\}$. & at v_5 add $\{v_5, v_6\}$.



It contains all the vertices. Hence it is a spanning tree.

Depth-First search Method (DFS)

~~Arbitrarily~~ Arbitrarily 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 & the vertex not already in the path. Continue this process by adding edges to this path as long as possible without producing any cles.

If the path goes through all the vertices in the graph, then the tree become 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 vertices that are not already visited. If still all the vertices are not visited, move back to another vertex & try again. Repeat this process until no

more edges can be added.

1) Use DFS to find a spanning tree for the graph with the vertex ordering abcdefg & gfedcba.

Let the vertex ordering be abcdefg.

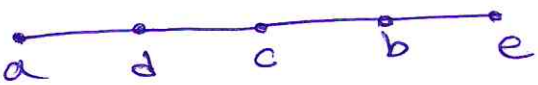
i) Start with the vertex 'a' as a root.

ii) We can not add 'b' & 'c' to 'a'. Add d to a.

iii) Now start with d. We can add b.
Add c to d.

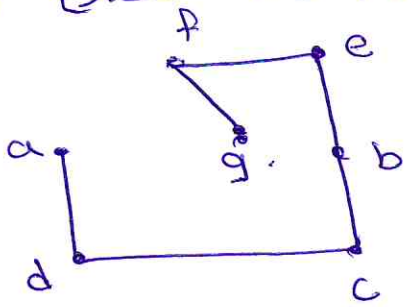
iv) We start with c. We can add b to c

v) Start at b. We remain with vertices e, f, g. We can add e to b



vi) we can add f to e then g to f.

It contains all the vertices, hence it is spanning tree as follows:



Let the vertex ordering be gfedcba

i) we start with vertex g as root.

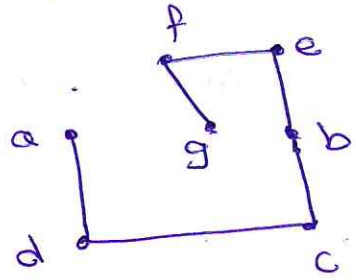
ii) we add f to g & then e to f.

iii) we can not add d & c to e. Add b to e

iv) we remain with vertices d, c & a. we can not add d to b but c to b.

v) The remaining vertices are d & a. Add d to c & then a to d.

It contains all the vertices. Hence it is a spanning tree as follows.



Depth-First search Algorithm:

Input: A connected graph G with vertices labelled $v_1, v_2, v_3 \ldots, v_n$.

Output: A spanning tree T for G.

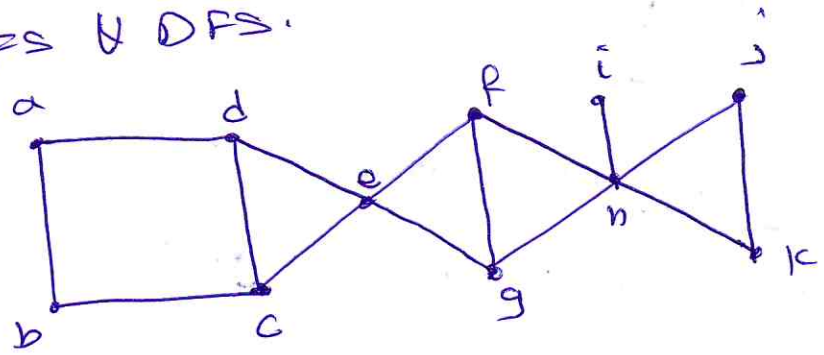
Method:

1. (Visit a vertex). Let v_1 be the root of T & let $L = v_1$

2. For all vertices adjacent to L, choose the edge $\{L, v_k\}$, where k is the minimum index such that adding $\{L, v_k\}$ to T does not create a cycle. If no such edge exists, go to step 3. Otherwise, add edge $\{L, v_k\}$ to T & set $L = v_k$. Repeat step 2 at the new value for L.

3. (Back track or terminate). If x is the parent of L in T, set $L = x$ & apply step 2 at the new value of L. If, on the other hand, L has no parent in T (so that $L = V_1$) then the depth-first search terminates & T is a spanning tree for G.

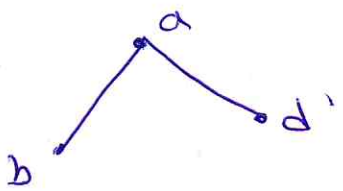
i) Find a spanning tree for the graph using BFS & DFS.



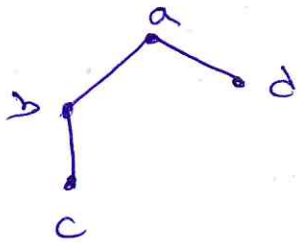
Using BFS

Let the vertex ordering be a b c d e f g h i j k.

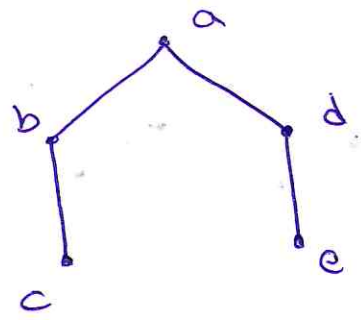
i) We start with root as 'a'. Add the edges {a, b} & {a, d}.



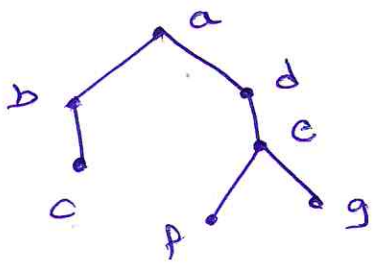
ii) At b add the edge {b, c}.



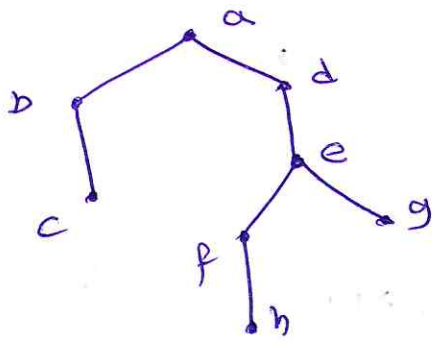
iii) At d, we can not add {d,c} as it forms a cycle. Add {d,e}



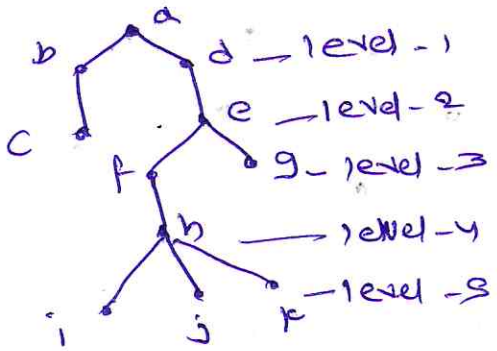
iv) At c we can not add any more edges. At e add {e,f} & {e,g}



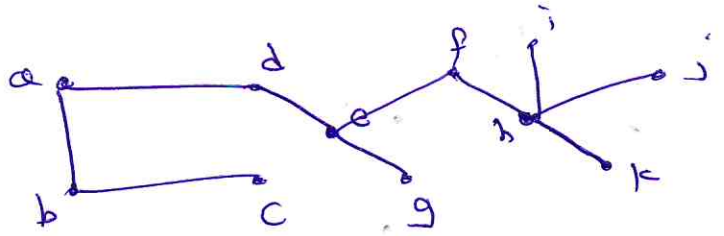
v) At f add {f,g}.



vi) At g we can not add {g,h} as if forms a cycle. At h add {h,i} , {h,j} & {h,k}.



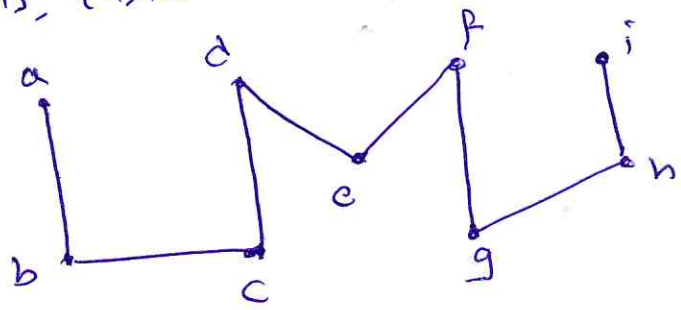
Thus we visited all the vertices. It is a spanning tree. we can write it as



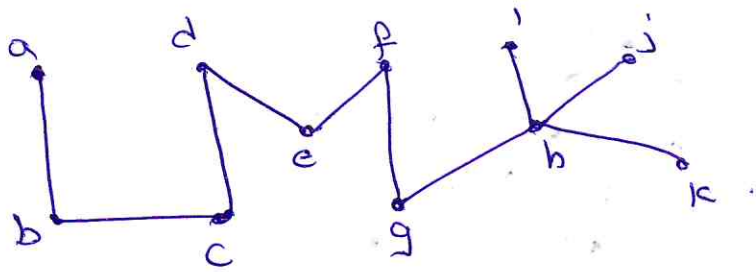
Using DFS

Let the vertex ordering be abcdefghijk.

i) we start with 'a' as a root. we add edges successively {a,b}, {b,c}, {c,d}, {d,e}, {e,f}, {f,g}, {g,h}, {h,i}.

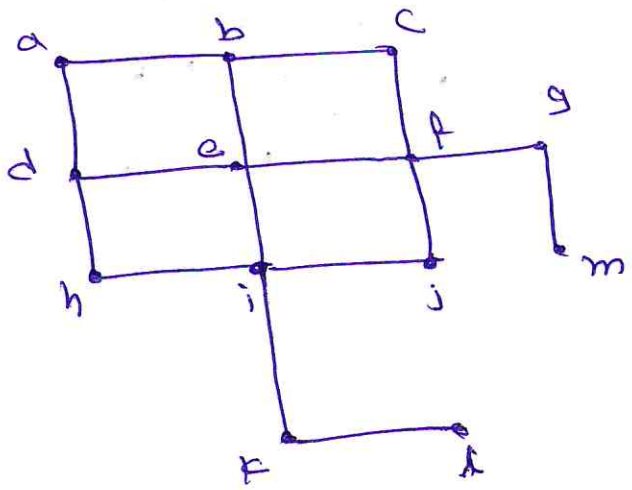


ii) At 'i' we can not add any more edges. visit previous ~~vertex~~ vertex h. we can add {h,j}. At 'j' we can not add any edges. Again visit previous vertex h & add {h,k}.



we visited all the vertices hence it is a spanning tree.

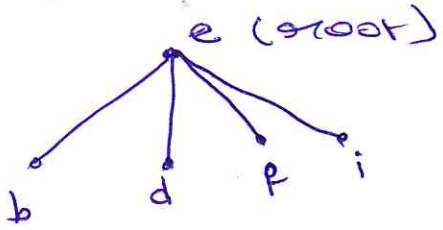
2) Using the BFS & DFS method, determine the spanning trees T for the graph G with e as the root of T



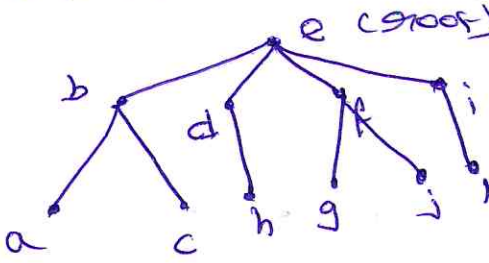
Using BFS method

Let the vertex ordering be eabcdfghijklm

i) We take 'e' as root of T, we find b d f & i at level 1.

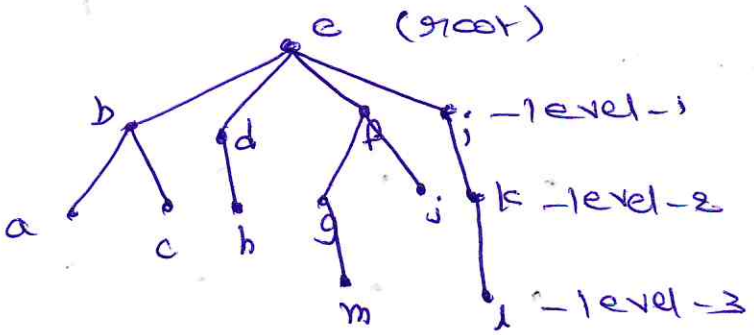


ii) The remaining vertices are acghjklm. The level 2 vertices for 'b' are 'a' & 'c' ~~and~~ 'i' for 'd' is h, for 'f' are g & j and i only one vertex k.

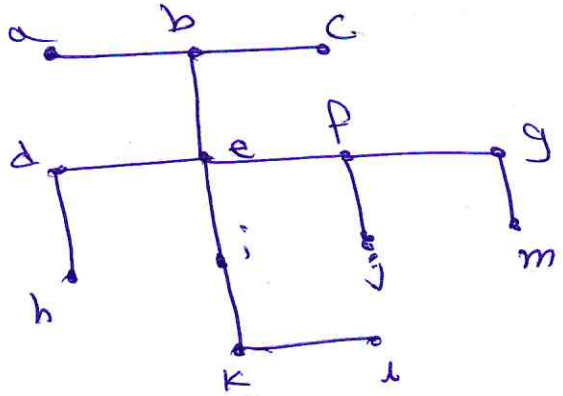


iii) The remaining vertices are l, m

At level 3 we add m to g & l to k.



we visited all the vertices. Hence it is a spanning tree. It can also be written in original form as

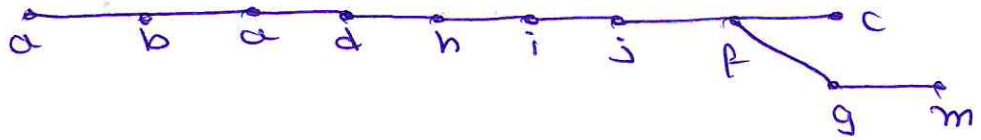


Using DFS method:

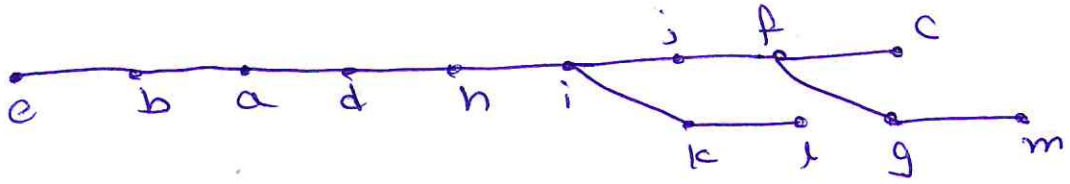
Let the vertex ordering be eabcdfghijklm

i) we take 'e' as a root of T. we start at 'e' & successively add edges {e,b}, {b,a}, {a,d}, {d,h}, {h,i}, {i,j}, {j,f} & {f,c}. we stop at 'c' since no more edges could be added.

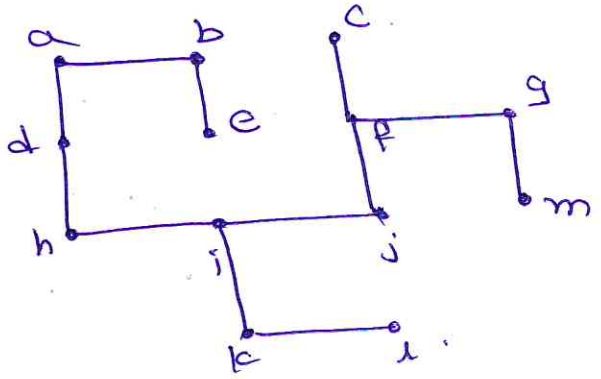
ii) we back track to vertex 'f' & find that edges {f,g} & {g,m} could be successively added. At m we cannot add any more edges.



iii) we back track to g, we cannot add any edges at g. Next back to 'f', no more edges added at 'f'. Again back to 'j', no edges can be added at 'j'. Now back to 'i', we can add successively the edges {i,k} & {k,l}.



we find that all the vertices are included. Here it is the spanning tree. It can be written in original form as



Minimal spanning trees:

Let G be a weighted graph. A minimal spanning tree of G is defined as a spanning tree of G with minimum weight. The ~~weighted graph G~~

Algorithms for minimal spanning trees:

Many methods are available for finding a minimal spanning tree in a given graph. Two methods, namely, Kruskal's algorithm & Prim's algorithm of finding a minimal spanning tree for a connected weighted graph with no weight is negative.
These two algorithms are examples of greedy algorithms

i) Kruskal's algorithm:

In this algorithm, we first select an edge of G which has the lowest weight among the edges in G which are not loops. Then for each successive step select another edge of smallest weight that does not make any cycle

with the previously selected edges. Continue this process until (n-1) edges have been selected & these edges form an acyclic subgraph T of G which is a minimal spanning tree of G.

The algorithm involves the following steps:

Input: A connected weighted graph G.

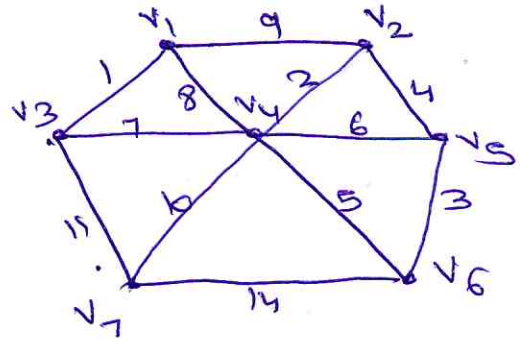
output: A minimal spanning tree.

Step (1)- Choose an edge of minimal weight. If there is more than one edge of minimal weight then choose arbitrarily one of these edges.

Step (2)- At each stage, choose (from the edges not yet chosen) the edge of lowest weight whose inclusion will not create a cycle.

Step (3)- If G has n vertices then stop after (n-1) edges have been chosen. Otherwise repeat step (2).

1) Find the minimal spanning tree of the graph by using kruskal's algorithm:



Solution:

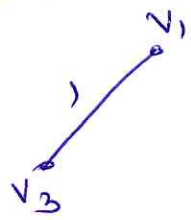
weights of edges

Edge	(v_1, v_3)	(v_2, v_4)	(v_5, v_6)	(v_2, v_5)	(v_4, v_6)	(v_4, v_7)
weight	1	2	3	4	5	6

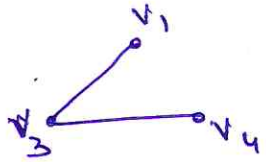
Edge	(v_3, v_4)	(v_1, v_4)	(v_1, v_2)	(v_4, v_7)	(v_3, v_7)	(v_6, v_7)
weight	7	8	9	10	11	14

The minimal spanning trees are calculated as follows:

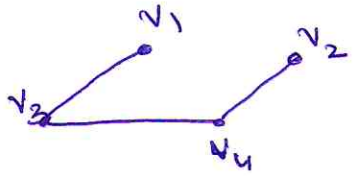
Step (1): choose the edge (v_1, v_3) which has the minimal weight.



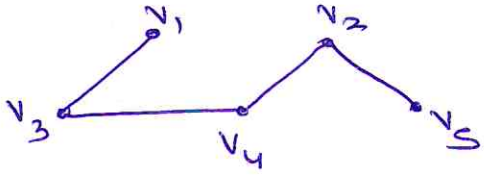
Step (2): Add the next edge (v_3, v_4)



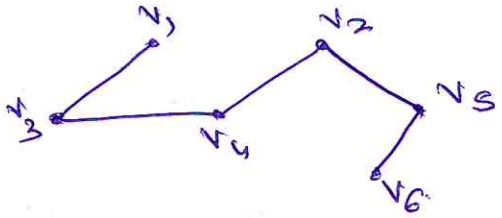
Step (3): Add the next edge (v_4, v_2).



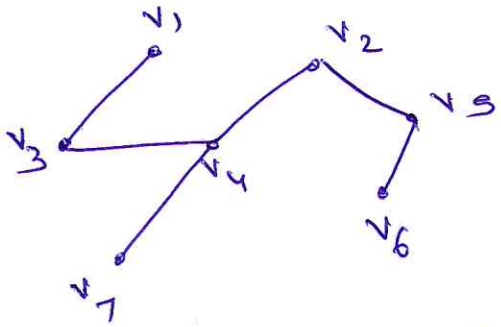
Step (4): Add the next edge (v_2, v_5)



Step (5): Add the edge (v_5, v_6)



Step (6): Add the edge (v_4, v_7)



Since the graph G has 7 vertices, it is enough to choose only 6 edges. Therefore, we stop the algorithm.

Prim's algorithm.

The algorithm starts at a designated vertex & choose an edge with minimum weight. Consider this edge & associated vertices as part of the desired tree. Then an edge with minimum weight which has not yet been selected is searched in which one of its nodes is in the tree while the other node is not. The process terminates when $(n-1)$ edges have been selected from a graph of 'n' nodes to form a minimal spanning tree.

The algorithm consists of the following steps:

Input: A connected weighted graph G.

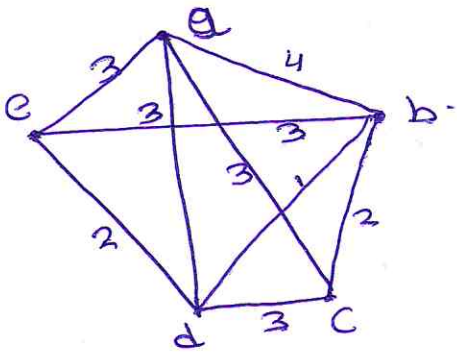
Out Put: A minimal spanning tree T.

Step (1): select any vertex & choose the edge and minimum weight from G.

Step (2): At each stage, choose the edge of smallest weight joining a vertex already included to vertex not yet included.

Step (3): Continue until all vertices are included.

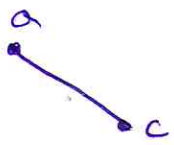
Find the minimal spanning tree of the weighted graph by using Prim's algorithm.



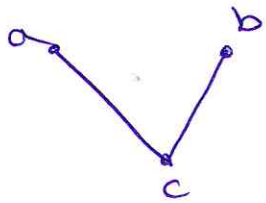
Solution:

The following steps need to be carried out to determine the minimum spanning tree of the given weighted graph:

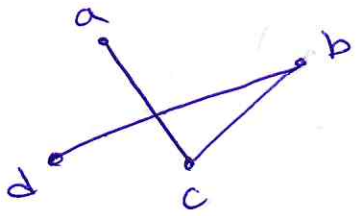
step (1): choose the vertex 'a'. Now, edge with minimum weight incident on 'a' is (a,c), (a,d), (a,e). let us choose the edge (a,c).



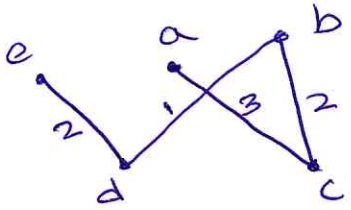
step (2): Now $w(a,b) = 4$, $w(a,d) = 3$, $w(a,c) = 3$, $w(c,b) = 2$ & $w(c,d) = 3$. choose the edge with minimum weight. The edge (c,b) has minimal weight.



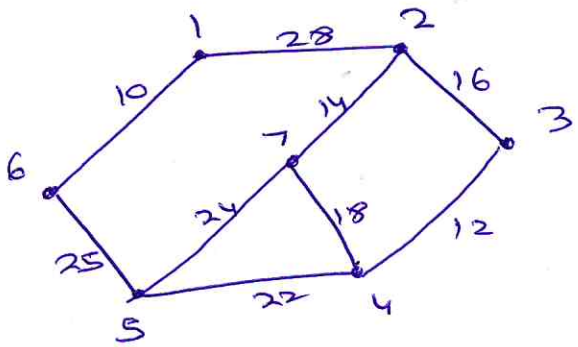
Step (3): Again, $w(a,c) = 3$, $w(b,d) = 1$ & $w(c,d) = 3$. Choose the edge (b,d), which has the minimum weight.



Step (4): Choose the edge (d,c). Hence all the vertices are covered. Thus, the minimal spanning tree is

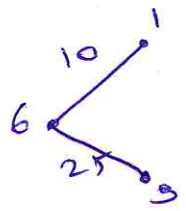


Construct the minimum spanning tree (MST) for the given graph using Prism's algorithm.

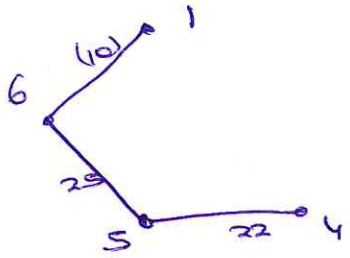


Step (1)

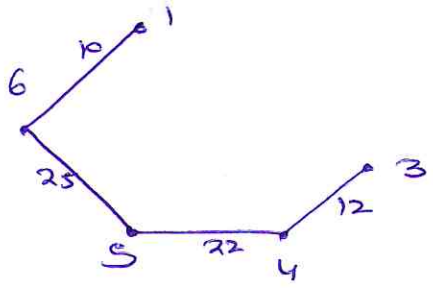
Step (2)



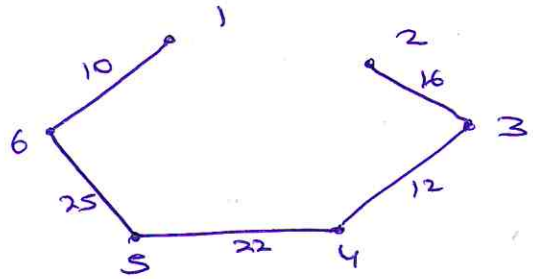
Step (3)



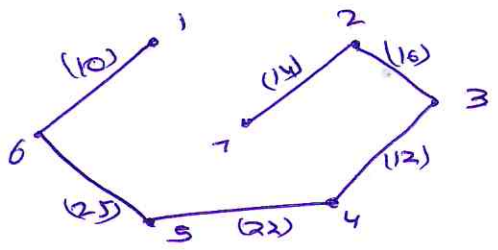
Step (4)



Step (5)



Step (6)



∴ all the vertices have been included in the MST so we stop.

Cost of MST = Sum of all edges weights

= 10+25+22+12+16+14

= 99 units.

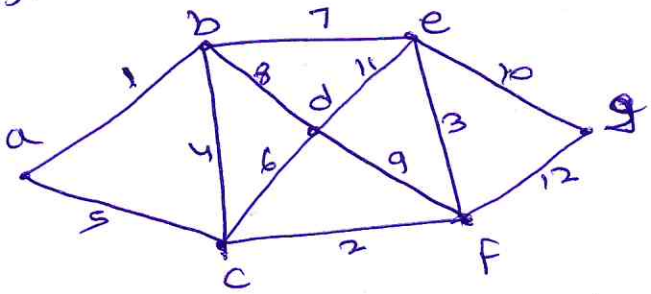
Prim's Algorithm - It starts to build the minimum spanning ~~tree~~ tree from any vertex in the graph.

Kruskal's Algorithm - It starts to build the minimum spanning tree from the vertex carrying minimum weight in the graph.

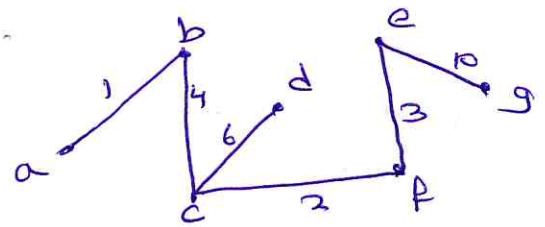
Prim's algorithm grows a solution from a random vertex by adding the next cheapest vertex to the existing tree.

Kruskal's algorithm grows a solution from the cheapest edge by adding the next cheapest edge

Using Prim's algorithm, find the cost of MST of the given graph.



The MST obtained by Prim's algorithm on the given graph is



Cost of MST

= sum of all edge weights

= 1 + 4 + 2 + 6 + 3 + 10 = 26 units