Module 5: Graph Theory

Graph: A graph G consists of a finite set V of objects called vertices, a finite set E of objects called edges and a function γ that assigns to each edge a subset $\{v, w\}$, where v & w are vertices (and they may even be same).

If e is an edge, and $\gamma(e) = \{v, w\}$

We say that e is an edge between v & w.

The vertices v & w are called as the end points of e.

Example

Let $V = \{1, 2, 3, 4\}$

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

Let g be defined by

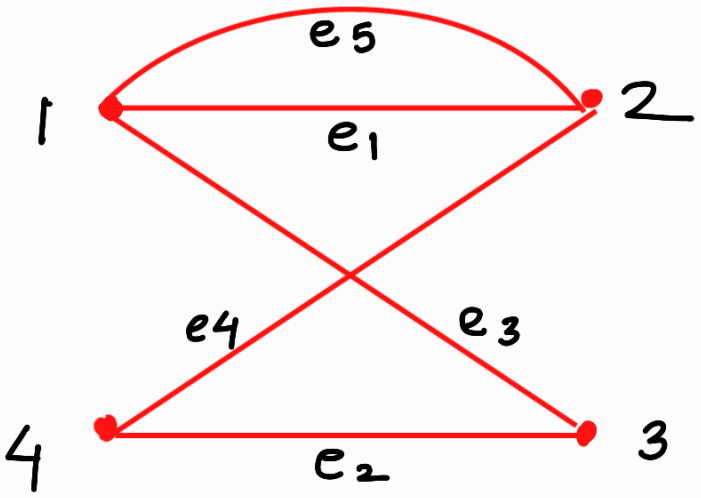
$\gamma(e_1) = \gamma(e_5) = \{1, 2\}$

$\gamma(e_2) = \{4, 3\}$

$\gamma(e_3) = \{1, 3\}$

$\gamma(e_4) = \{2, 4\}$

Then $G = (V, E, \gamma)$ is a graph as shown below:-



Degree

The degree of a vertex is the number of edges having that vertex as an end point.

Loop

A graph may contain an edge from a vertex to itself, such an edge is referred to as a loop.

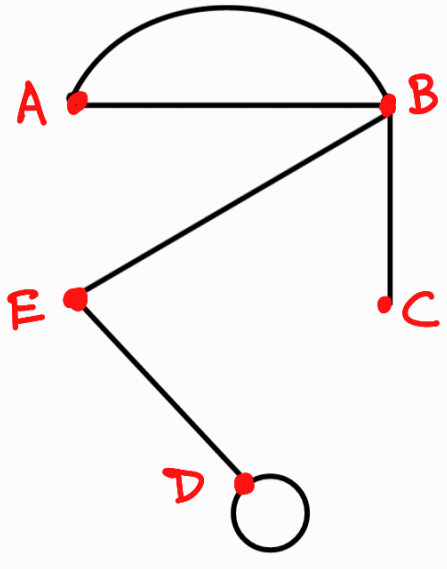
A loop will always have a degree 2, since that vertex serves as both end points of the loop.

Isolated Vertex

A vertex with degree 0 will be called an isolated vertex

Adjacent Vertices

A pair of vertices that determine an edge are called adjacent vertices.



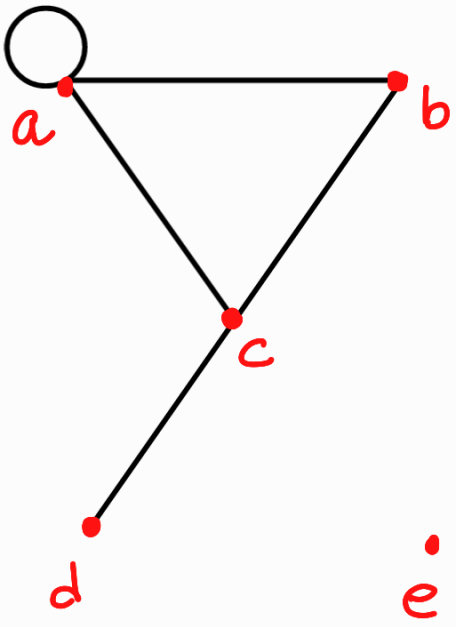
A → 2

B → 4

C → 1

D → 3

E → 2



a → 4

b → 2

c → 3

d → 1

e → 0

Vertex e → Isolated Vertex

Vertex (a, b), (a, c), (b, c), (c, d) are called adjacent vertices.

Vertex (a, d) are not adjacent.

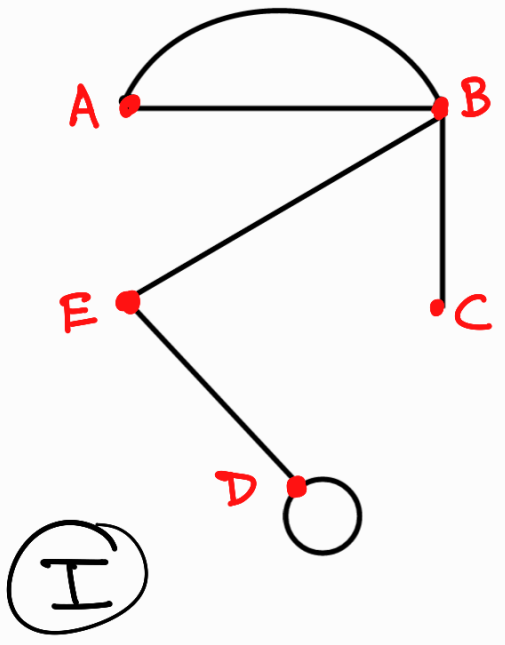
Paths

A path π in a graph is a sequence $\pi: v_1, v_2, v_3, \ldots v_k$ of vertices, each adjacent to the next and a choice of an edge between v_i & v_{i+1} so that no edge is chosen more than once.

This means that it is possible to begin at v_i and travel along edges to v_k and never use the same edge twice.

A path is called SIMPLE if no vertex appears more than once.

Example :



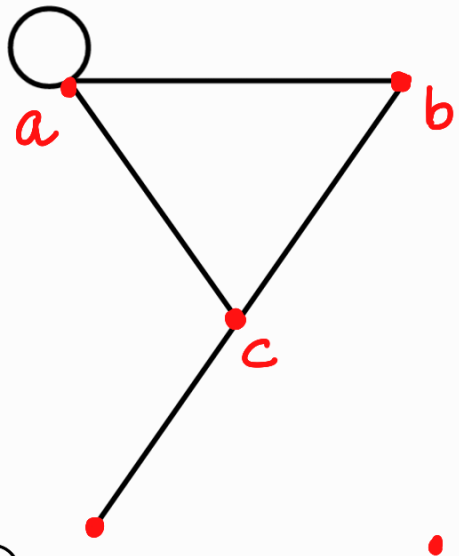
(I)

There could be multiple paths such as :-

Π_1 : D, E, B, C

Π_2 : A, B, E, D, D

Π_3 : A, B, A



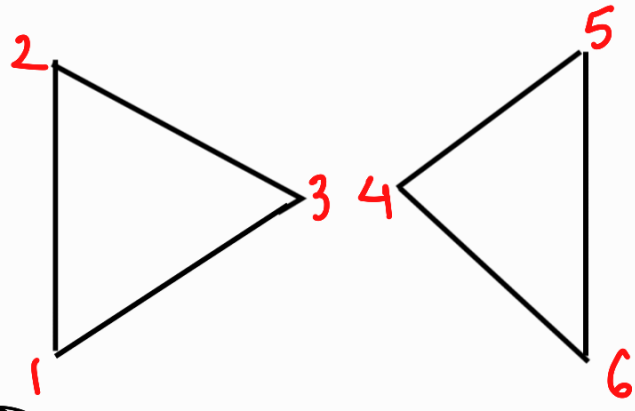
(II) d e

Π_1 : a, b, c, a

Π_2 : d, c, a, a

Π_3 : c, a, b, c, d

↳ This path is not simple.



(III)

Path 1, 2, 3, 2 will not be a path as the single edge between 2 & 3 would be travelled twice.

Circuit

A circuit is a path that begins and ends with the same vertex.

A circuit $v_1, v_2, \ldots\ldots v_{k-1}, v_1$ is simple if the vertices v_1, v_2, v_{k-1} are all distinct.

Example:

In the fig. (II), the path

π_1: a, b, c, a is a circuit & its simple.

In the fig. (III), the path

π_1 : 1, 2, 3, 1 &

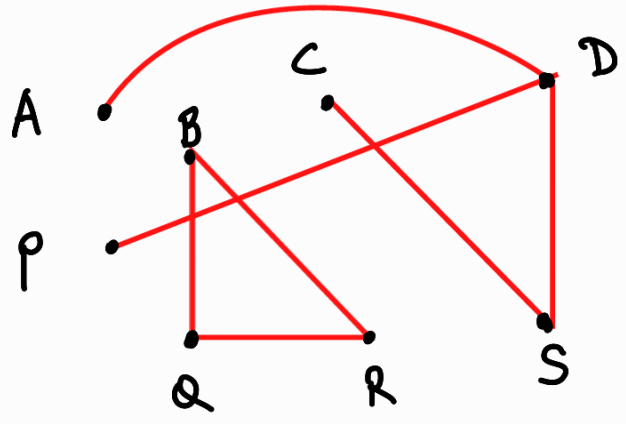
π_2 : 4, 5, 6, 4 are simple circuits

Connected Graphs

A graph is called connected if there is a path from any vertex to any other vertex in the graph, otherwise the graph is called disconnected.

If the graph is disconnected, the various connected pieces are called the components of the graph.

Q.1. Determine whether the graph is connected or disconnected. If disconnected, find its connected component.

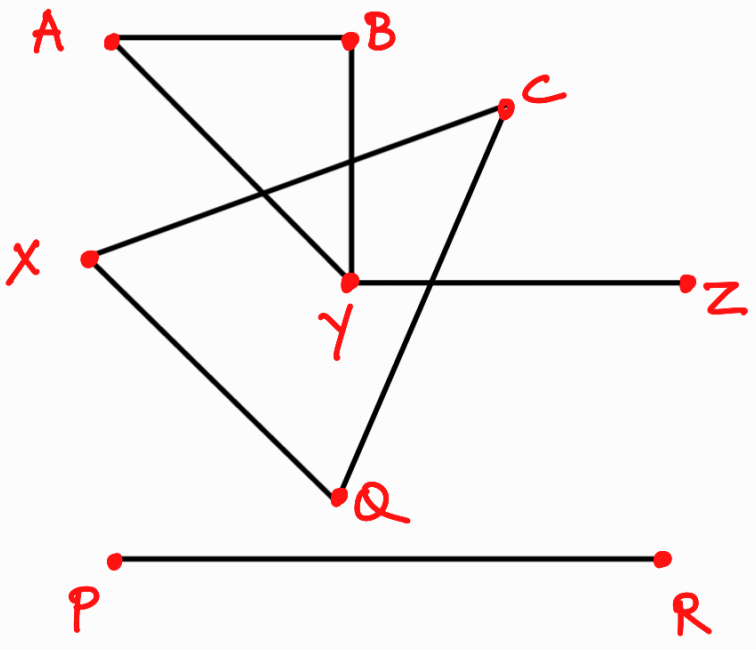


The above shown graph is not connected.

It's connected components are

{ A, D, P, S, C } and { B, Q, R }

Q.2



Not Connected

Connected Components are :-

{ A, B, y, z }, { c, Q, x } and { P, R }

Discrete Graph

D_2

D_5

For each integer $n \geq 1$, we let D_n denote the graph with n vertices and no edges.

Hence D_2 & D_5 are two discrete graphs on 2 & 5 vertices each.

Linear Graph

For each integer $n \geq 1$, we let L_n denote the graph with n vertices $\{v_1, v_2 \ldots v_n\}$ and with edges $\{v_i, v_{i+1}\}$ for $1 \leq i < n$

L_2

L_4

Subgraph

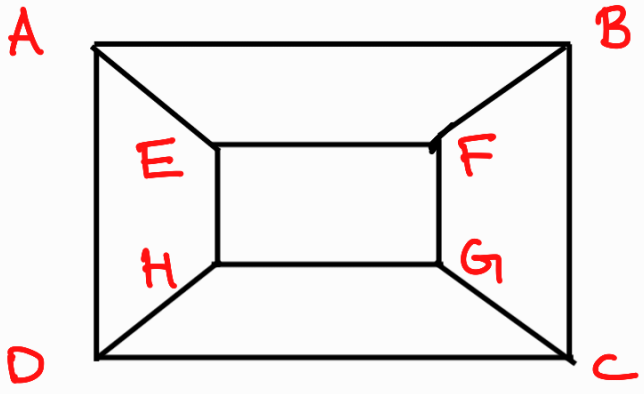
Let $G = (V, E, \gamma)$ is a graph. Choose a subset E_1 of the edges in E and a subset V_1 of the vertices in V.

So that V_1 contains all the end points of edges in E.

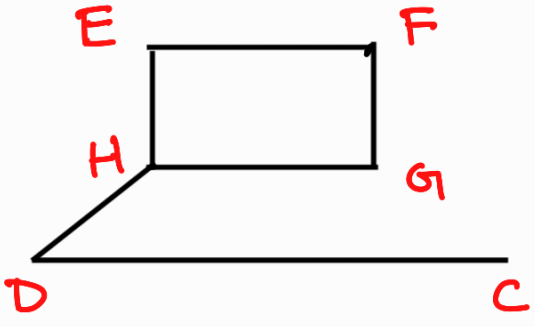
Then $H = (V_1, E_1, \gamma_1)$ is also a graph, where γ_1 is γ restricted to edges in E_1.

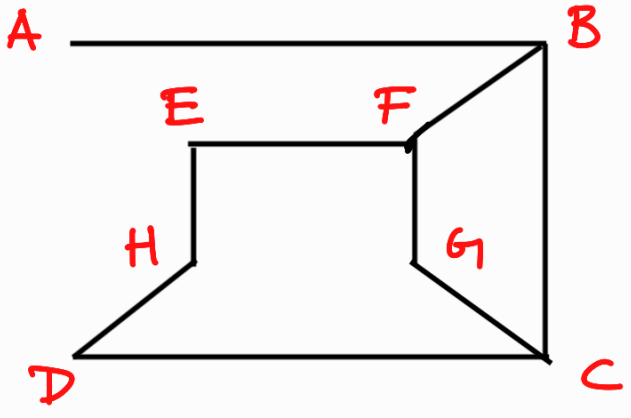
Such a graph H is called a subgraph of G.

Example



Subgraph





$V_1 = \{E, F, G, H, C, D\}$

$V_1 = \{A, B, C, D, E, F, G, H\}$

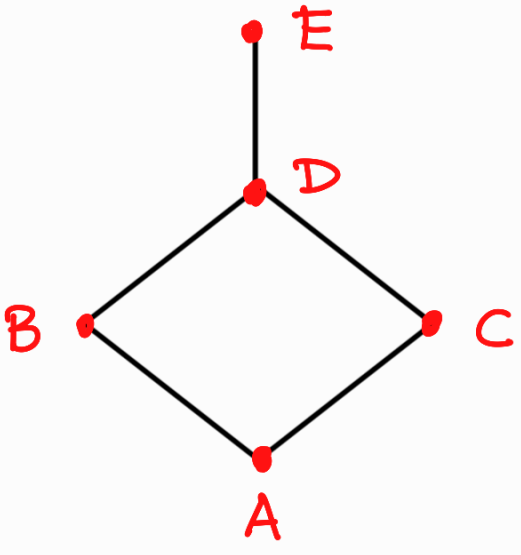
Euler & Hamiltonian Graphs

A path in a graph G is called an Euler Path if it includes every edge exactly once.

An Euler Circuit is an Euler path that is a circuit.

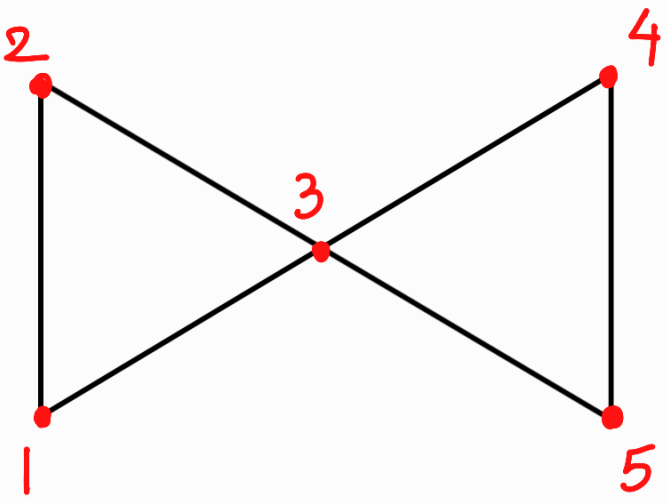
Example

1. An Euler path is π: E, D, B, A, C, D



There is no Euler Circuit

2.



One Euler Circuit in the graph is

Π: 5, 3, 2, 1, 3, 4, 5

Theorem 1:

a) If a graph G_1 has a vertex of ODD degree, there can be no Euler circuit in G_1.

b) If G is a connected graph & every vertex has an even degree, then there is an Euler Circuit.

Theorem 2:

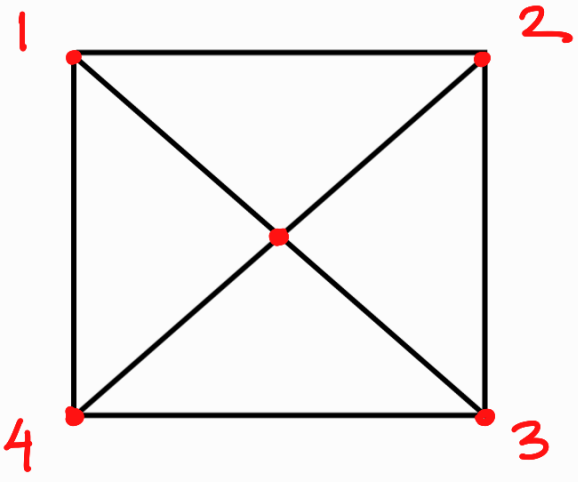
a) If a graph G1 has more than two vertices of ODD degree, then there can be no Euler Path in G1.

b) If G1 is connected & has exactly two vertices of ODD degree, then there is an Euler Path in G1.

Any Euler path in G1 must begin at one vertex of ODD degree & end at the other.

Solved examples on Euler Paths & Circuits

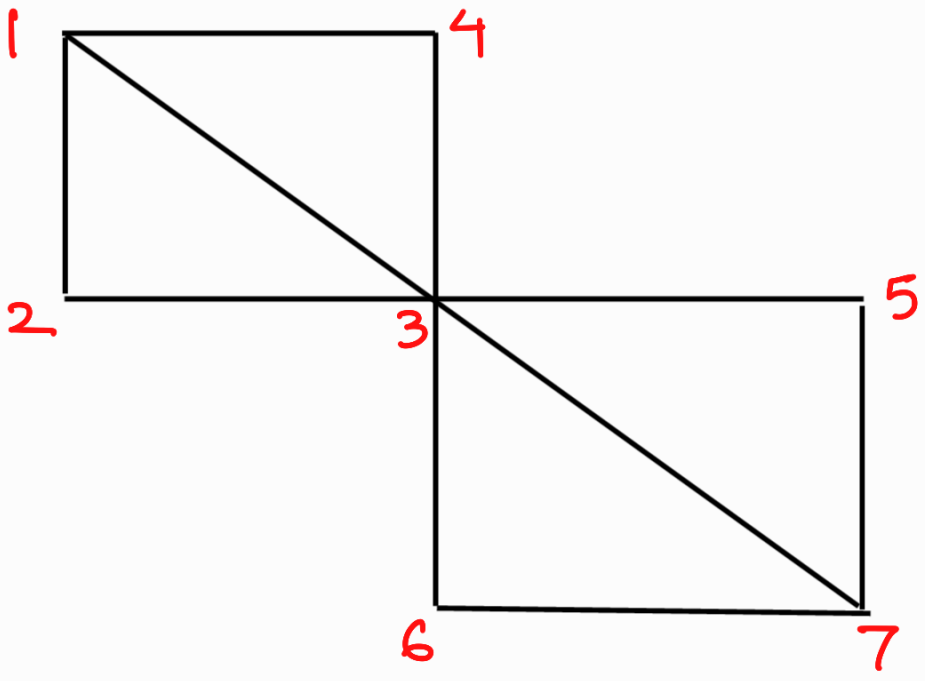
Q.1



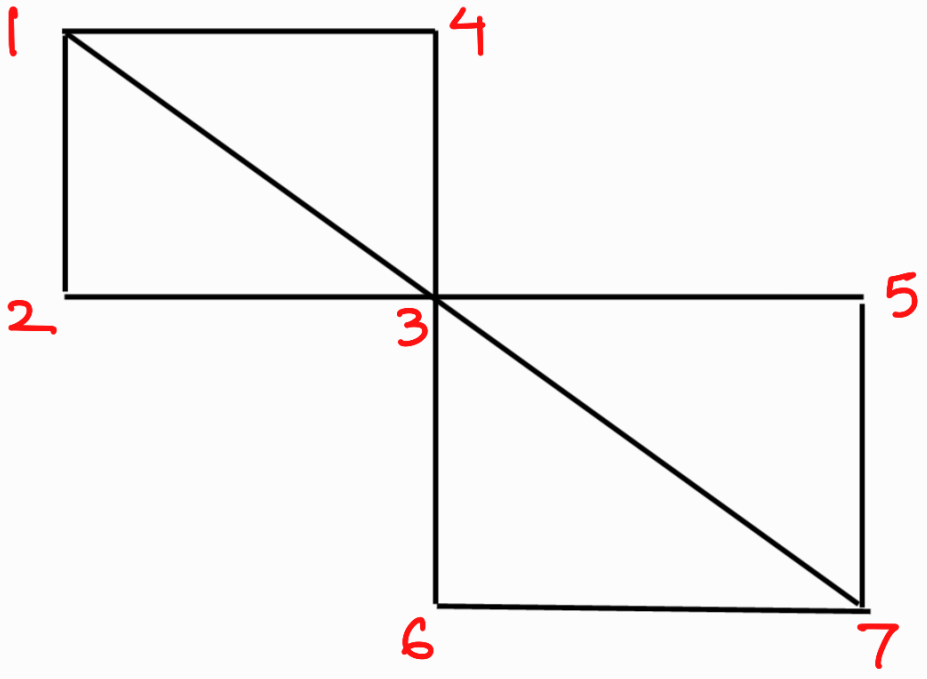
Each of the 4 vertices has degree 3.

Then according to Theorem 1 & 2, there is neither an Euler Path nor an Euler circuit.

Q.2



Q.2



There are 7 vertices.

2 vertices 1 & 7 have odd degree hence it doesn't have an Euler Circuit.

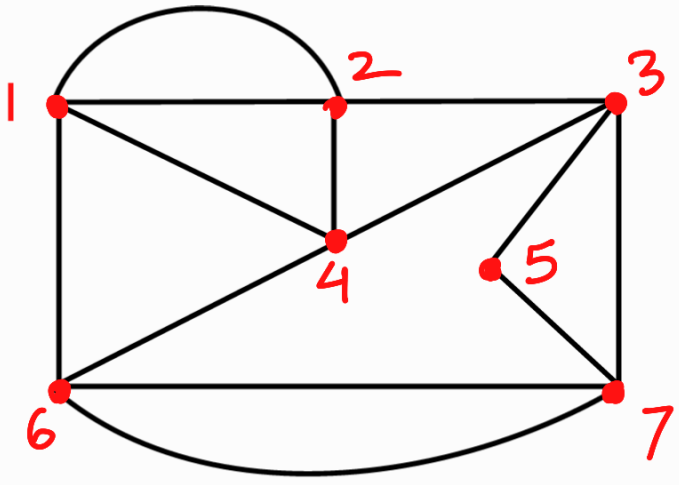
By Theorem 2(b) there is an Euler Path

Π: 1, 2, 3, 1, 4, 3, 6, 7, 3, 5, 7

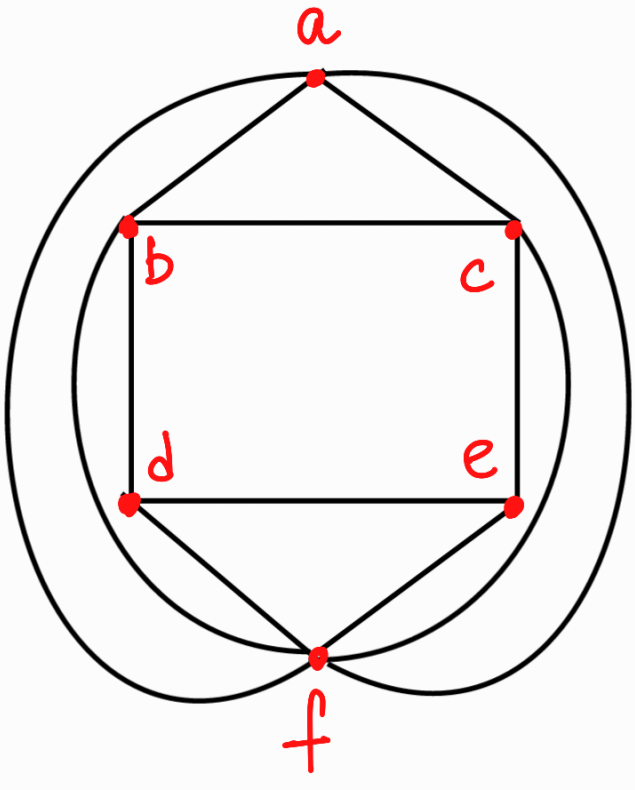
or

1, 2, 3, 1, 4, 3, 6, 7, 5, 3, 7

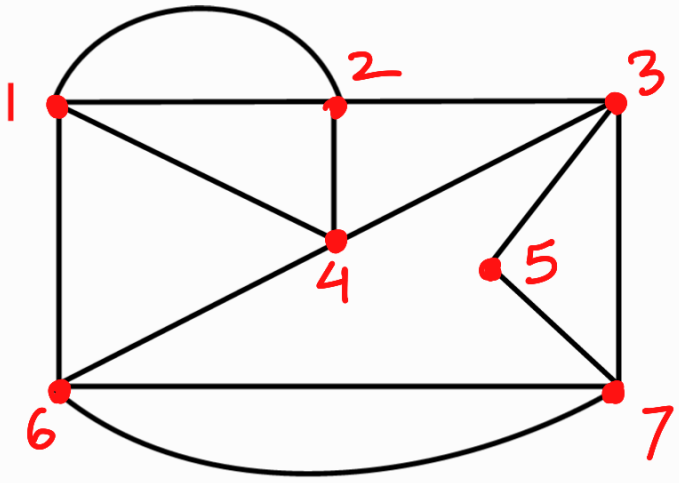
Q.3.



Q.4.



Q.3.

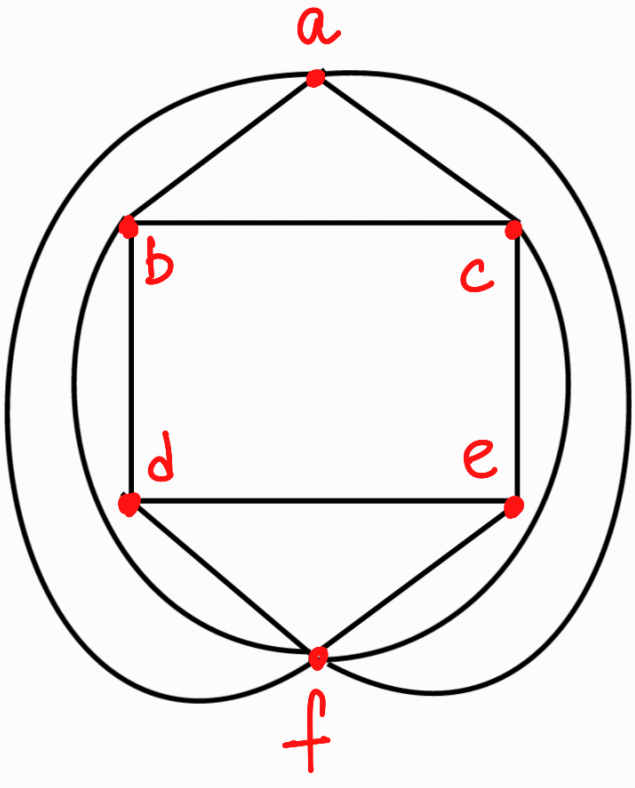


No Euler Path

Circuit :

1, 6, 7, 6, 4, 3, 5, 7,
3, 2, 4, 1, 2, 1

Q.4.

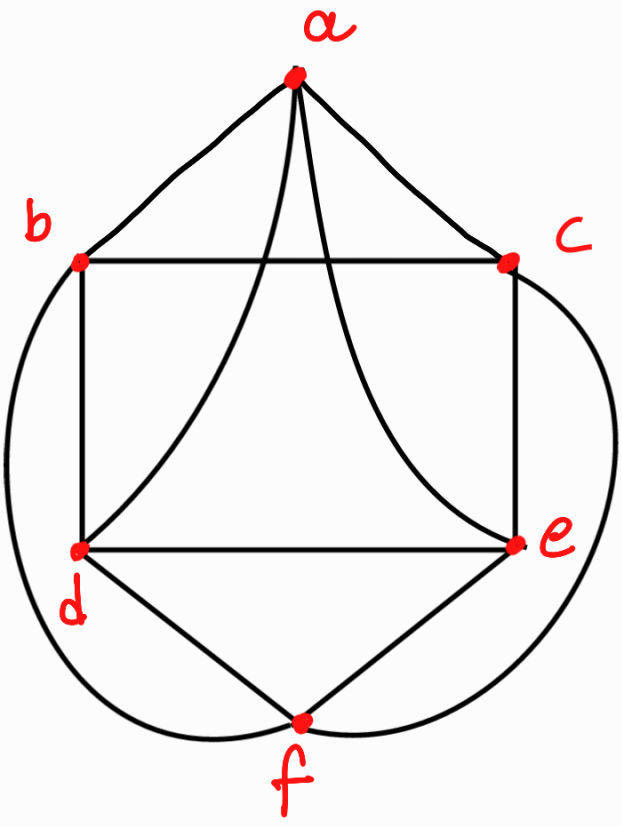


No Euler Cruit

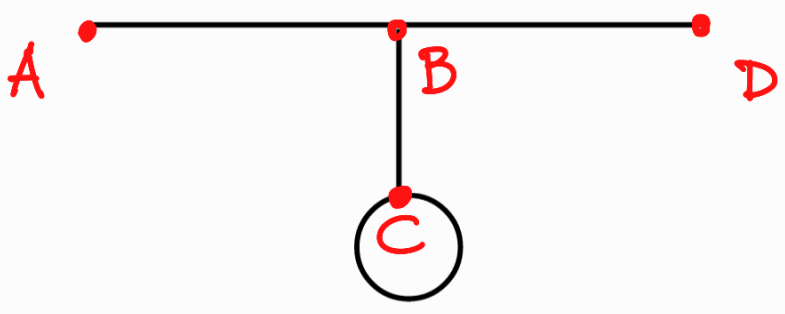
Path :

d, b, c, a, b, f, a,
f, d, e, c, f, e

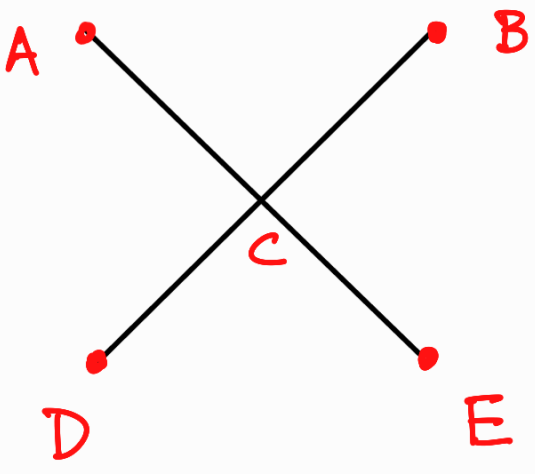
Q.3.



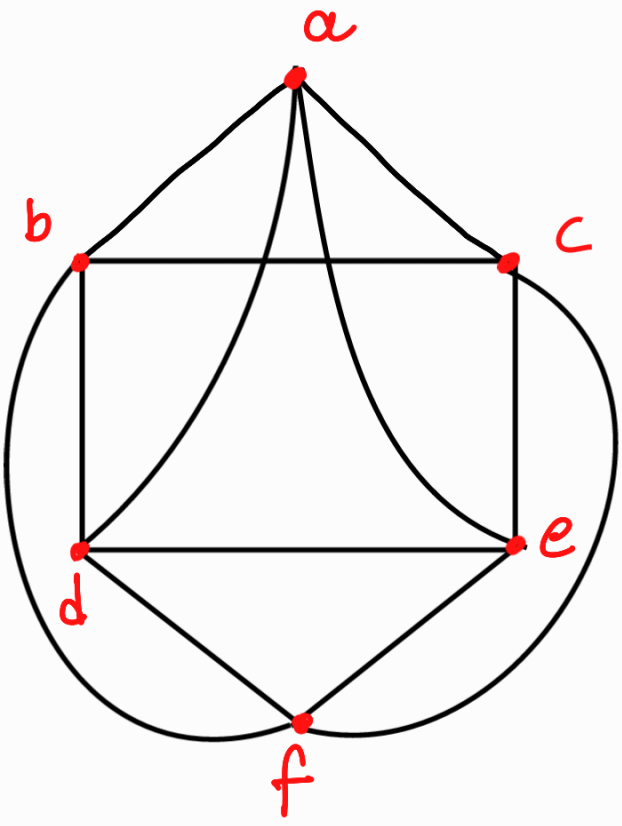
Q.4.



Q.5.



Q.3.

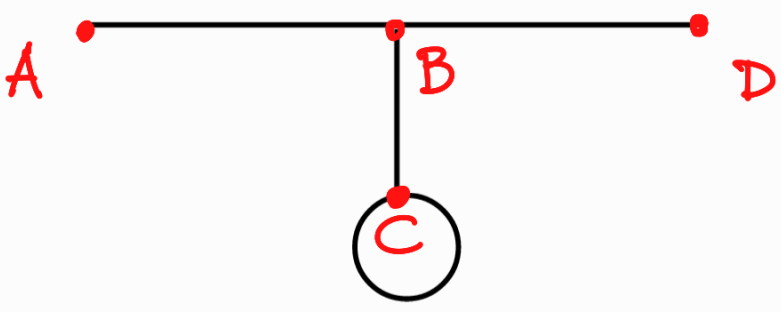


Circuit

⇓

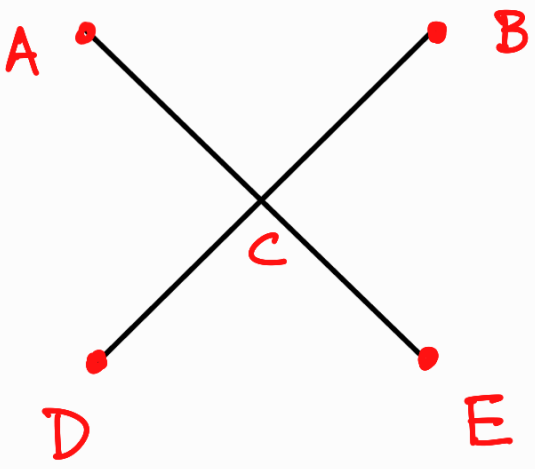
a, b, f, c, a,
d, b, c, e, f, d,
e, a

Q.4.



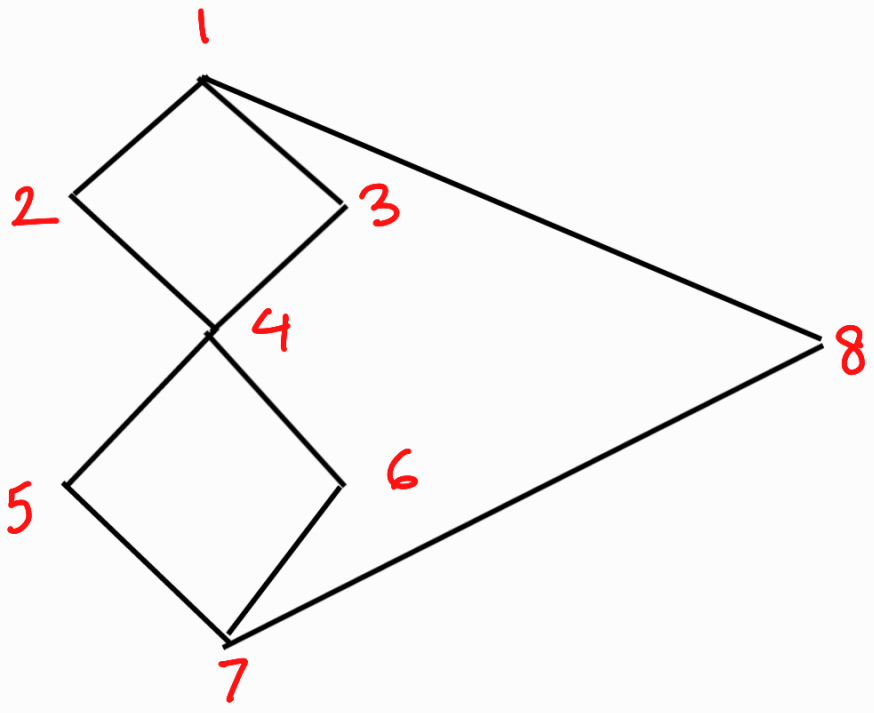
Neither

Q.5.

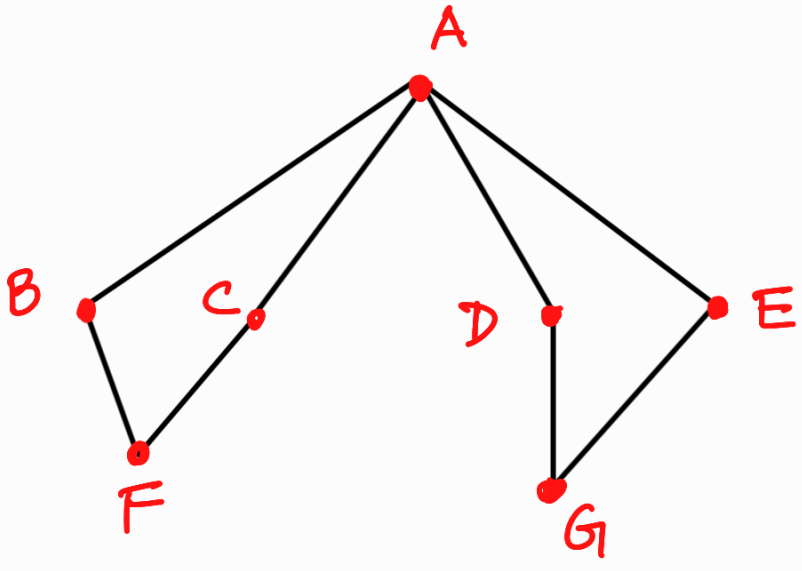


Neither

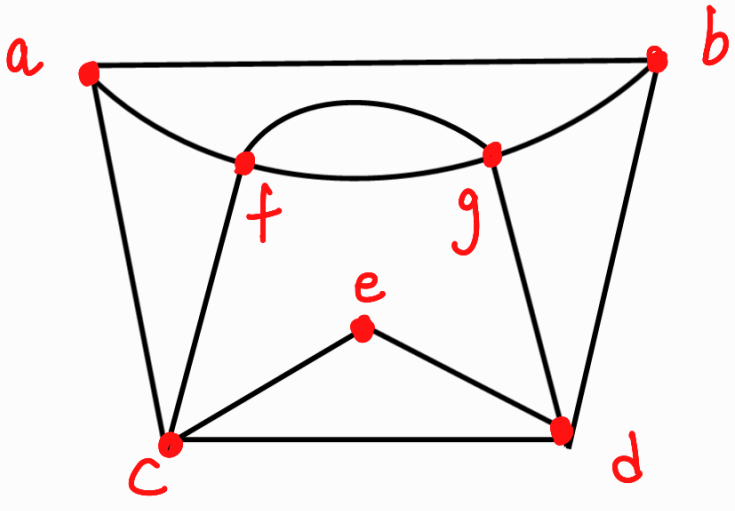
Q.6.



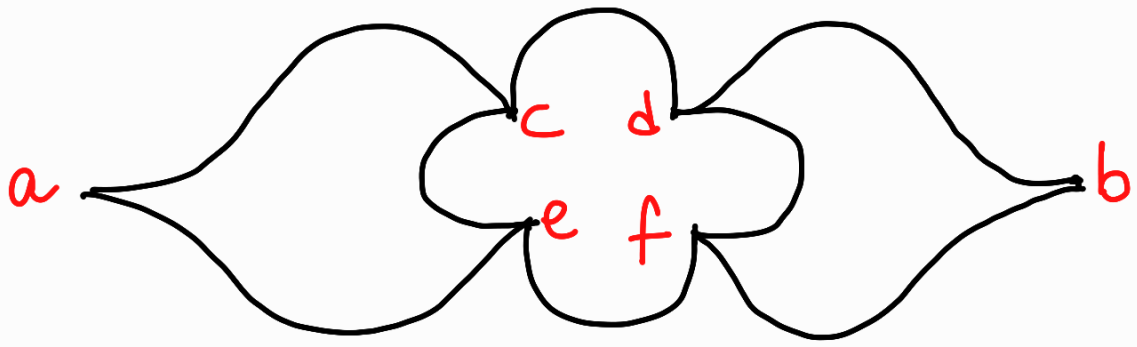
Q.7.



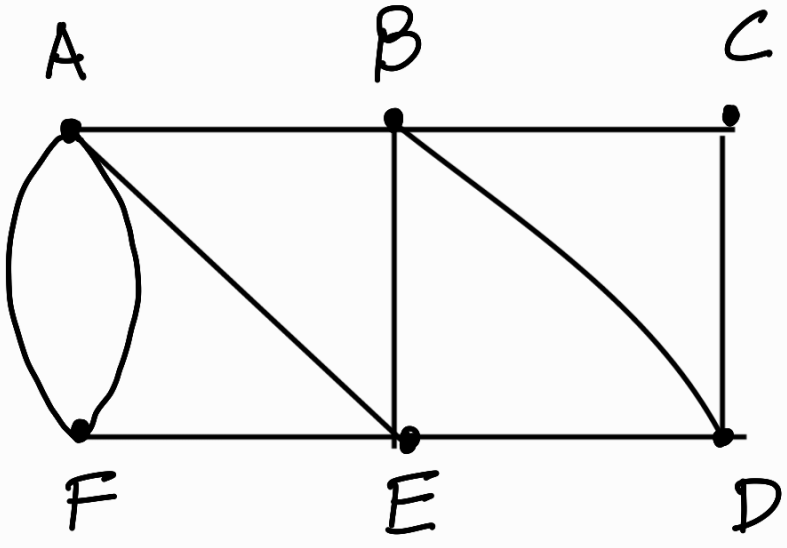
Q.8.



Q.9.



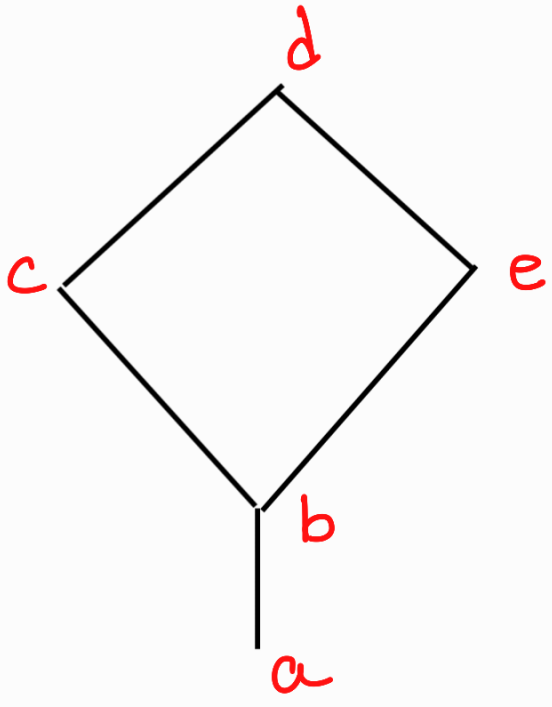
Q.10.



Hamiltonian Path & Circuits

A Hamiltonian Path is a path that contains each vertex exactly once.

A Hamiltonian Circuit is a circuit that contains each vertex exactly once except for the first vertex which is also the last.



Path a, b, c, d, e is a Hamiltonian path.

Theorem 1

G has a Hamiltonian circuit if for any two vertices u & v of G that are not adjacent, the degree of u plus the degree of v, is greater than or equal to the number of vertices.

Theorem 2

G has a Hamiltonian circuit if each vertex has degree greater than or equal to $n/2$

Theorem 3

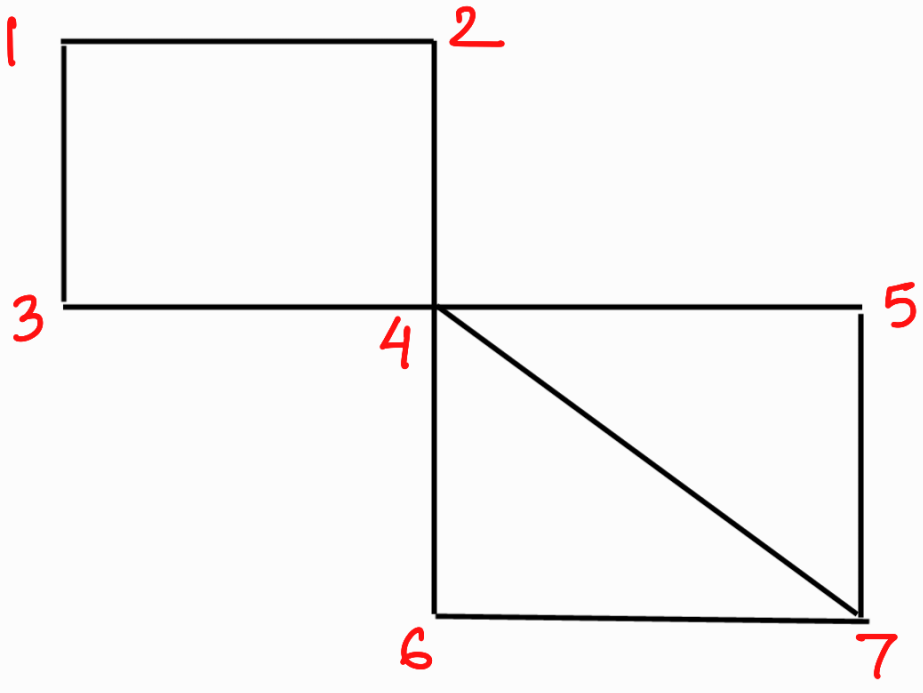
Let the number of edges of G be m. Then G has a Hamiltonian circuit, if

$$m \geq \frac{1}{2}(n^2 - 3n + 6)$$

Theorem 4

There is always a Hamiltonian path in a directed complete graph.

Q.1.



Degree of 1 + Degree of 7

$= 2 + 3 = 5$

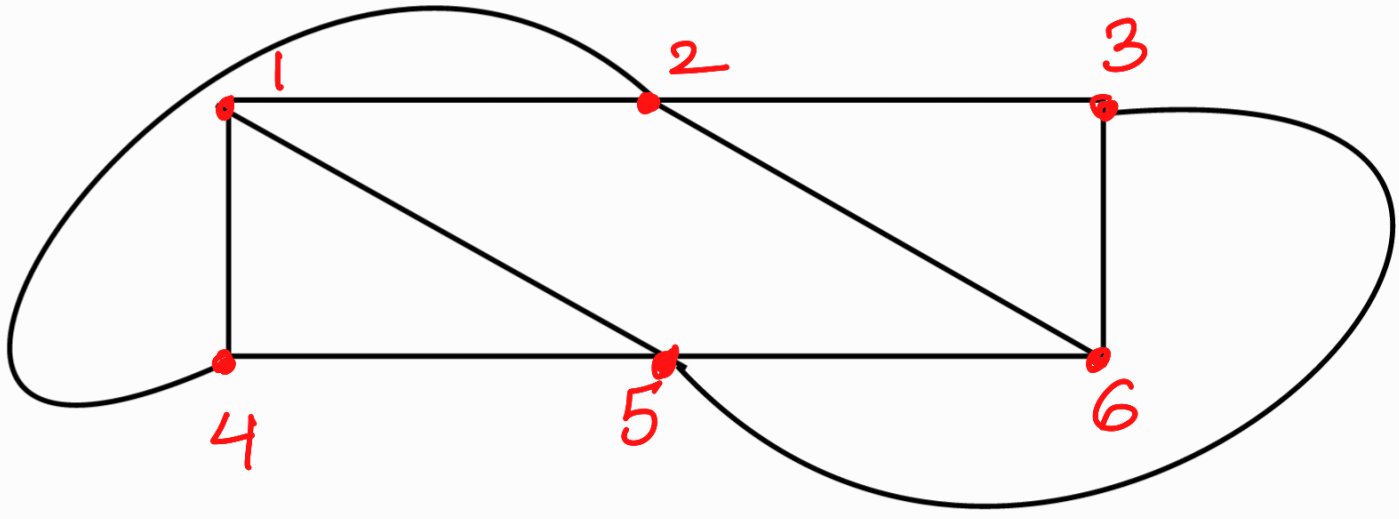
$\Rightarrow 5 \neq 7$ {$\because$ 7 = No of vertices}

$\therefore$ There is no Hamiltonian Circuit

But there is a Hamiltonian path

$\therefore \quad \pi: \quad 3, 1, 2, 4, 6, 7, 5$

Q.2.



According to Theorem 2, we can say that each vertex has a degree $\geq n/2$ $(6/2)$

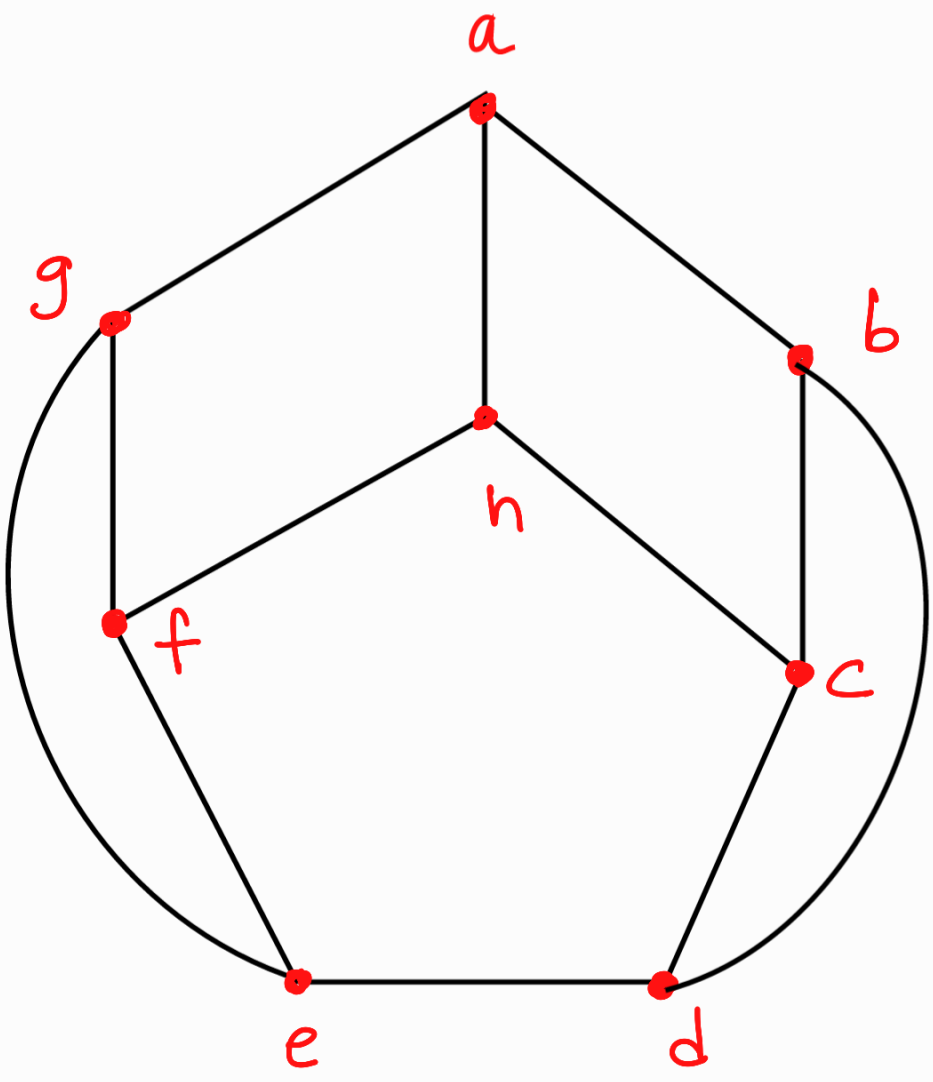
$\Rightarrow$ It's a Hamiltonian Circuit

$\pi: \quad 1, 4, 5, 6, 3, 2, 1$

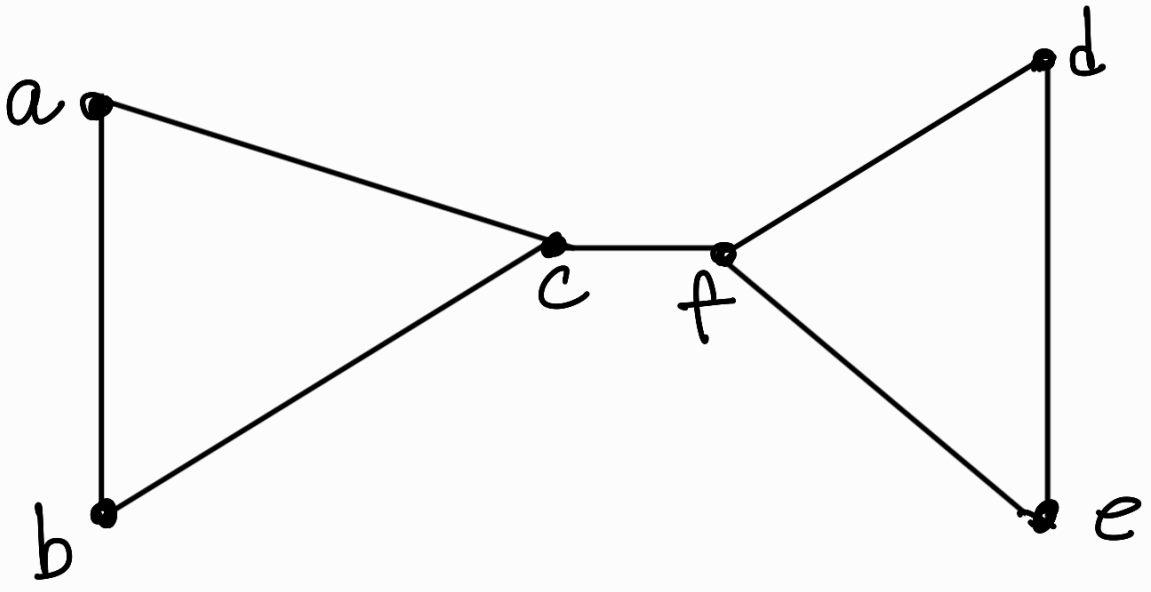
Hamiltonian Path:

1, 4, 5, 6, 3, 2

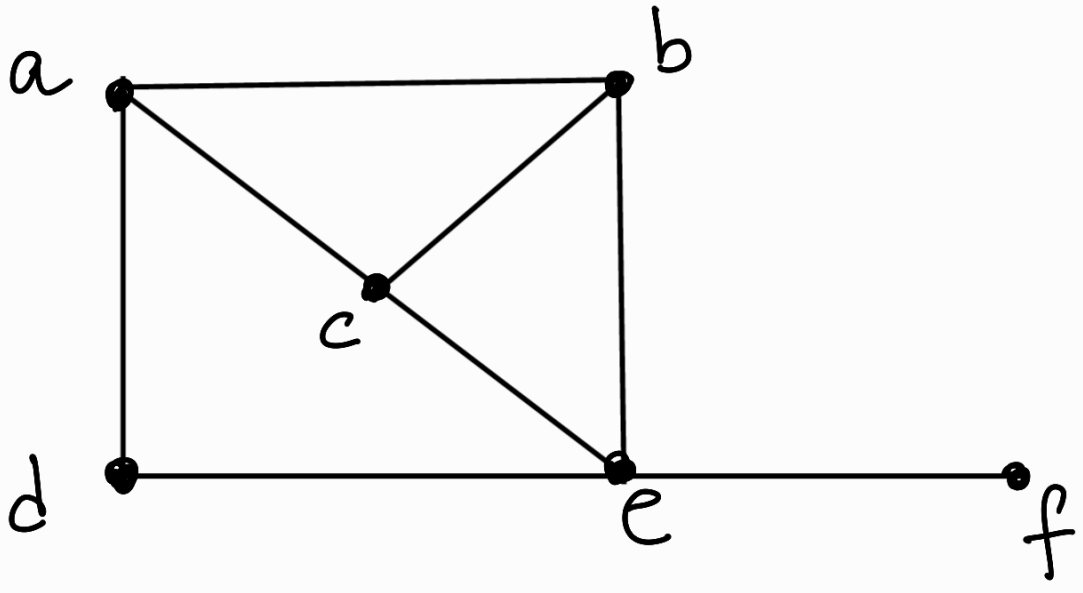
Q. 3



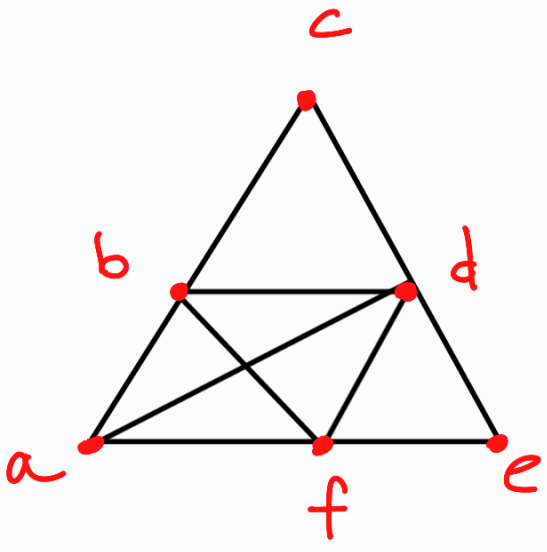
Q.4.



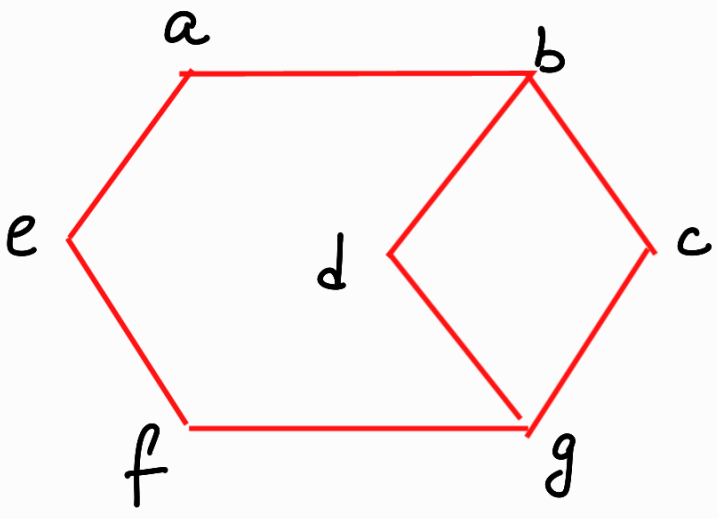
Q.5.



Q.6. Determine which of the following graph contains an Eulerian or Hamiltonian circuit.

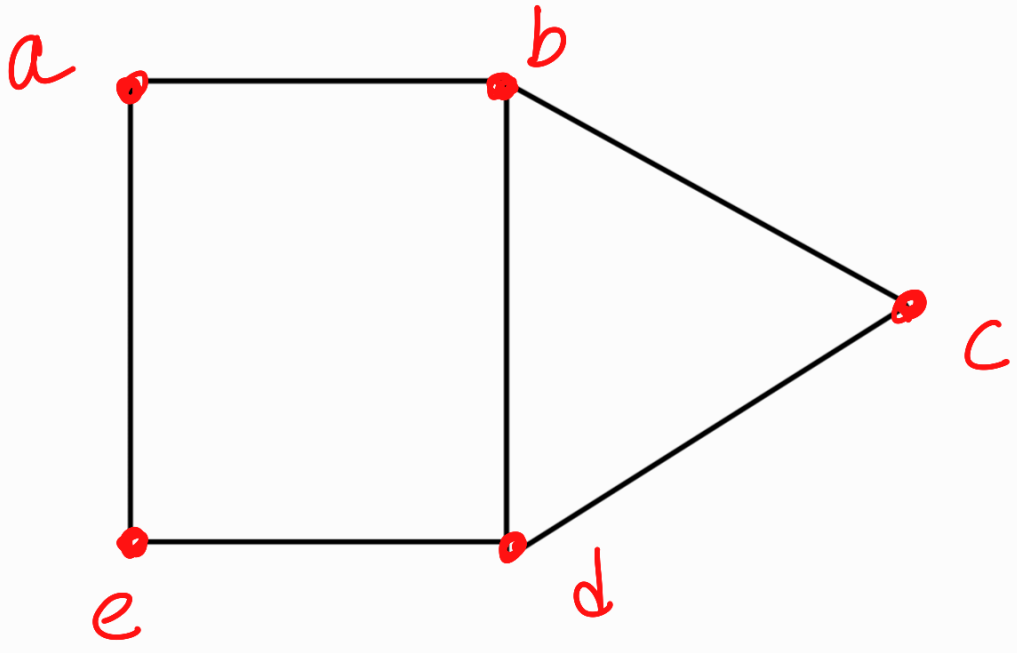


i)



ii)

Q.7.



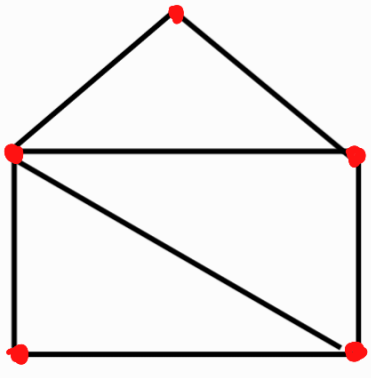
Not every Eulerian graph is Hamiltonian Graph.

Not every Hamiltonian graph is Eulerian Graph.

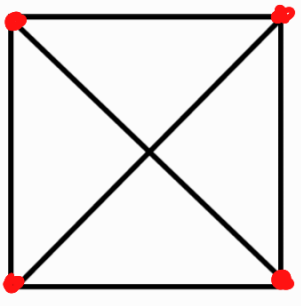
Planar Graphs

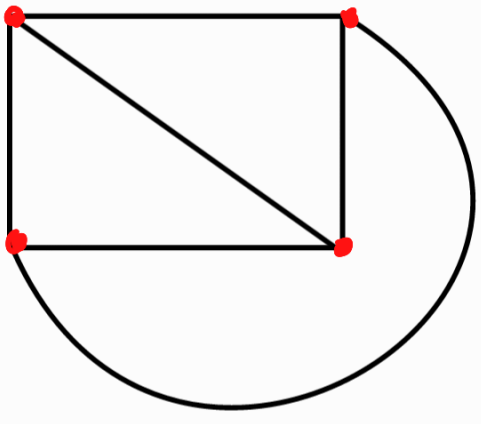
A graph is said to be planar if it can be drawn on a plane in such a way that no edges cross one another, except of course at common vertices.

Planar



Not Planar initially





But can be converted to planar

Theorem (Euler's Formula)

For any connected planar graph

$$V - e + r = 2$$

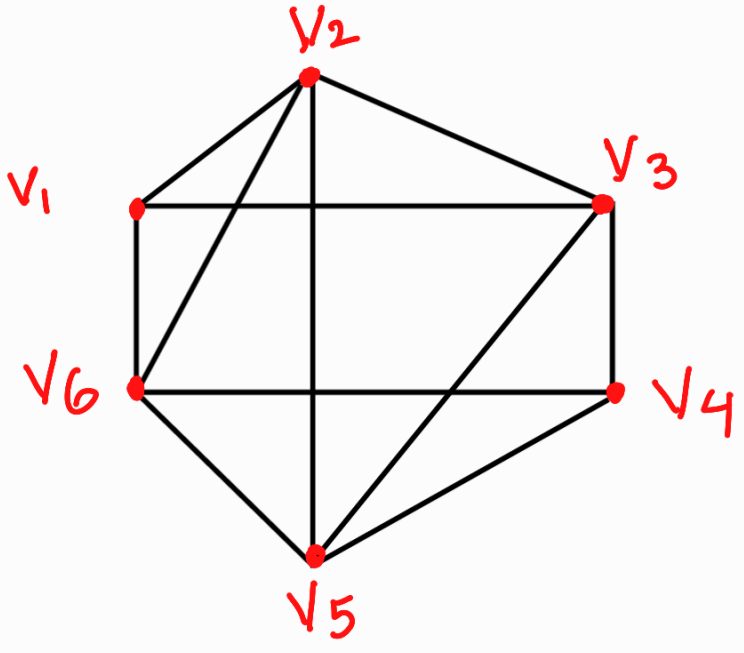
where :-

$V \rightarrow$ No. of Vertices

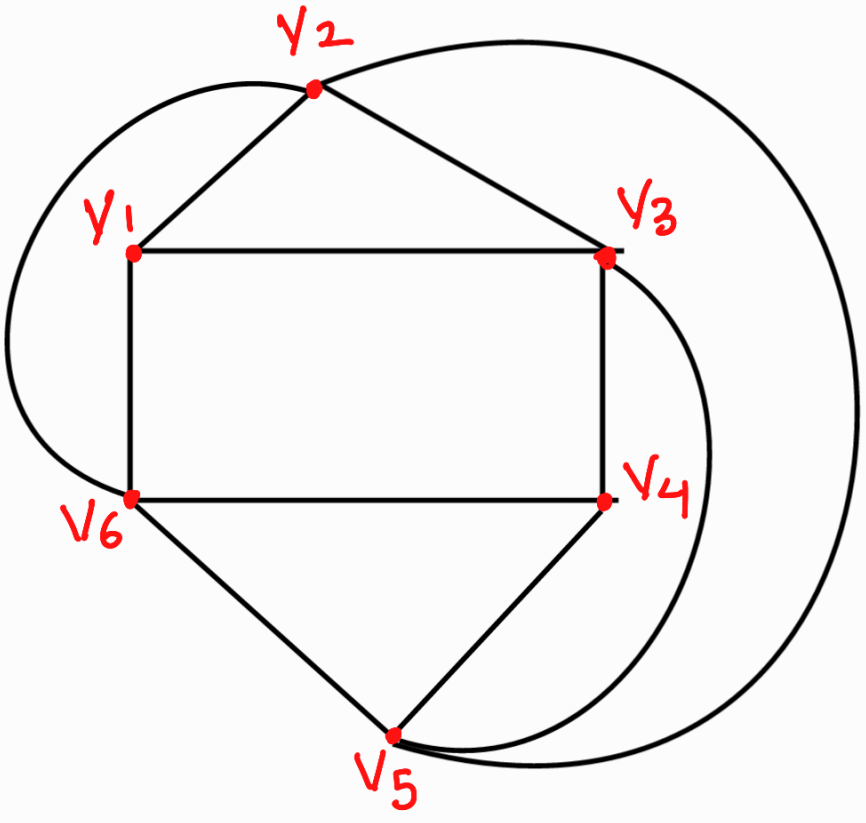
$e \rightarrow$ No. of Edges

$r \rightarrow$ No. of regions of the graph

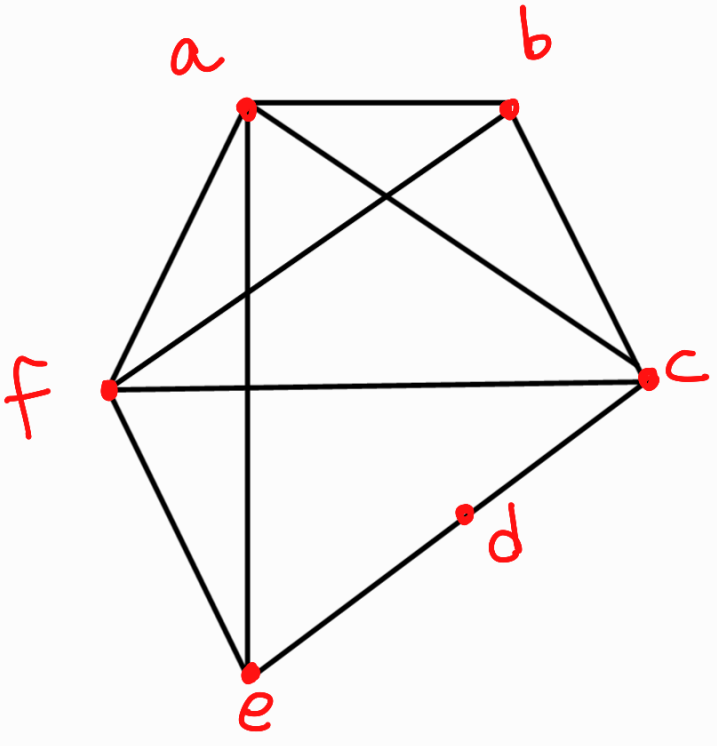
Q.1 By drawing the graph, show that the following graphs are planar graphs.



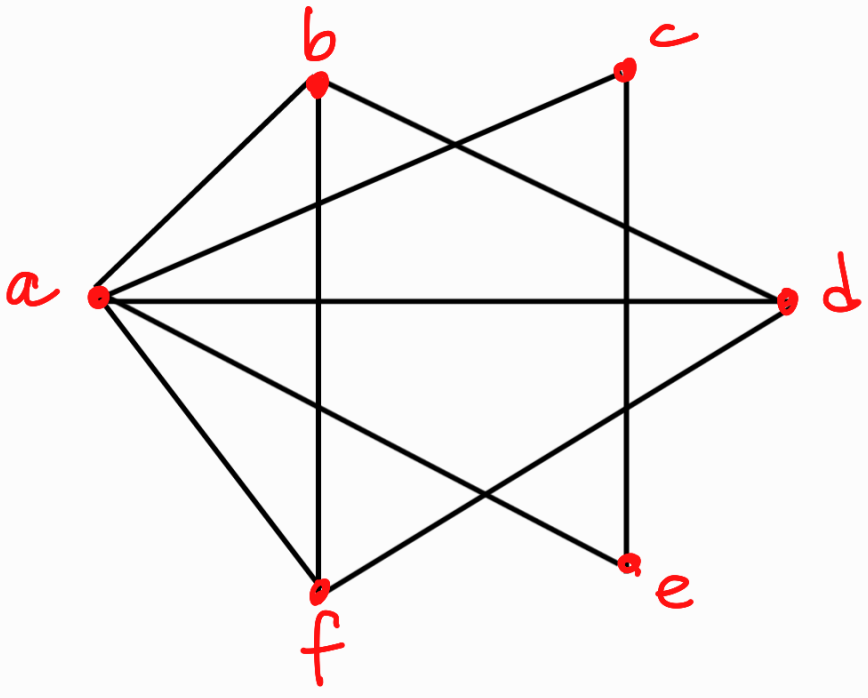
Sol:



Q.2



Q.3



Q.4. How many edges must a planar graph have if it has 7 regions and 5 nodes. Draw one such graph.

Sol. According to Euler's formula, in a planar graph,

$$V - e + r = 2$$

where v, e, r are the number of vertices, edges and regions in a planar graph.

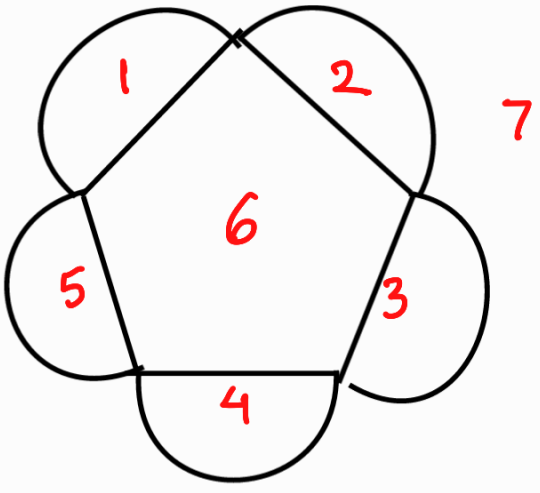
Here, $v = 5$, $r = 7$, $e = ?$

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

$$5 - e + 7 = 7$$

$$\therefore \quad e = 10$$

Hence, the given graph must have 10 edges. This graph is shown as below.



Q.5. Determine the number of regions defined by a connected planar graph with 6 vertices & 10 edges. Draw a graph.

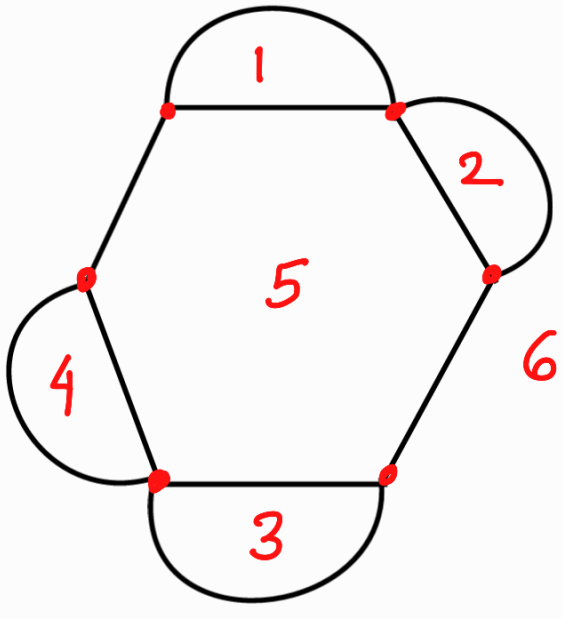
Sol: Given, $v = 6$, $e = 10$

Hence, by Euler's formula for a planar graph,

$$V - e + r = 2$$

$$\therefore \quad 6 - 10 + r = 2$$

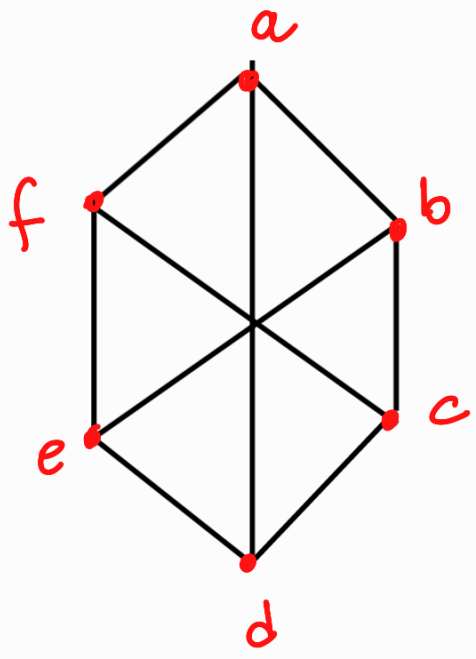
$$\therefore \quad r = 6$$



$V = 6$
$e = 10$
$r = 6$

Q.7. Which of the following graphs are planar ?

a)



$v = 6$

$e = 9$

$r = 7$

According to Euler's formula

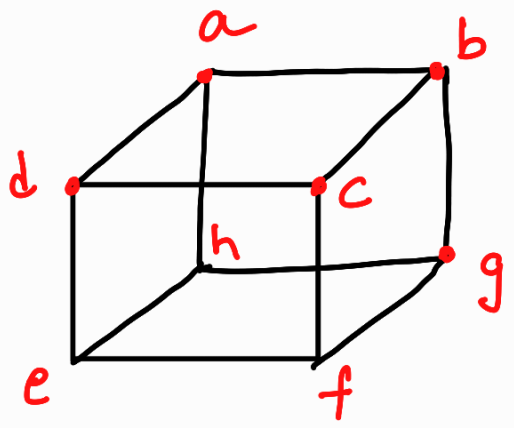
$$v - e + r = 2$$

$$\therefore \quad 6 - 9 + 7 = 2 \quad ?$$

$$4 \neq 2$$

$\therefore$ Graph is non-planar

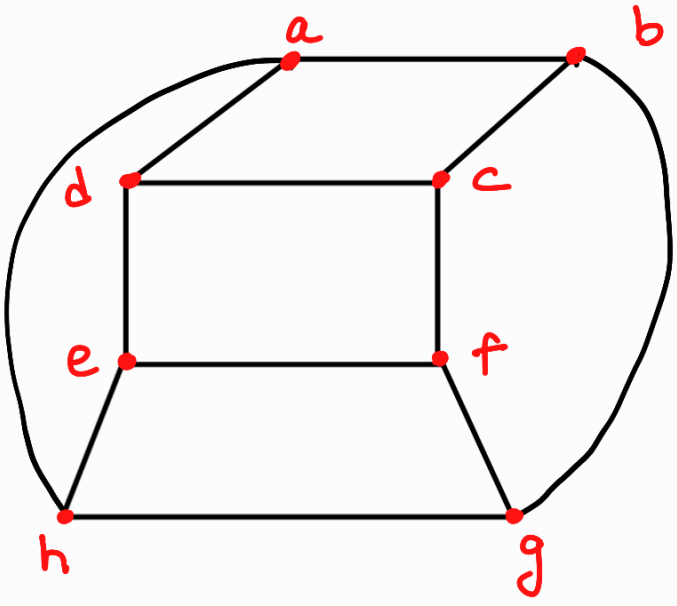
b)



In such questions, always first check if the graph could be redrawn without any intersecting edges.

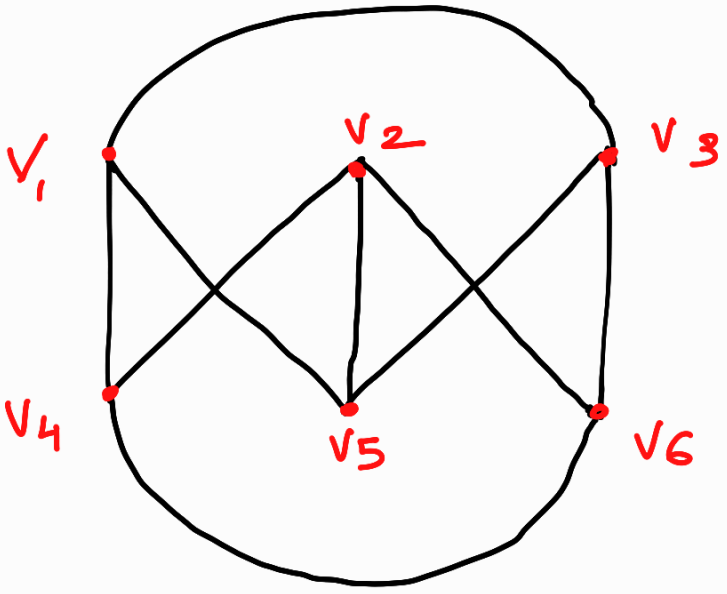
If Yes, then it is a planar graph & there is no need to prove the Euler's formula.

The above graph could be redrawn as follows.



The above graph is a planar graph be cause edges do not cross each other.

c)



Isomorphic Graphs

Two graphs are thought of as isomorphic if they have identical behaviour in terms of graph properties.

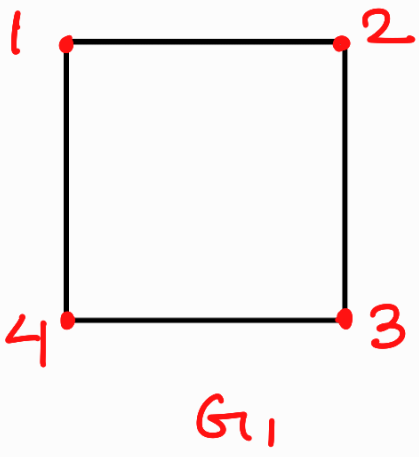
Two graphs $G_1(V_1, E_1)$ and $G'(V', E')$ are said to be isomorphic to each other if there is a one to one correspondence between their vertices and between their edges such that the relationship is preserved.

Such that :-

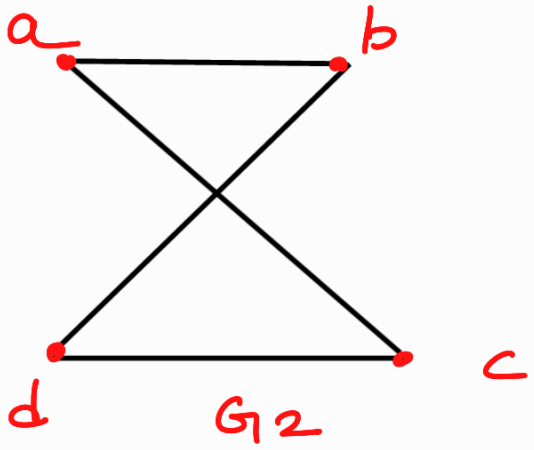
Suppose that edge e is incident on vertices v_1 & v_2 of G_1 then the corresponding edge e' in G' must be incident on the vertices v_1' & v_2' that corresponds to v_1 & v_2.

Adjacency also needs to be PRESERVED

For eg:



G_1



G_2

G_1 is isomorphic to G_2, because there is a 1 to 1 correspondence between G_1 & G_2 as follows:-

$1 \rightarrow a$

$2 \rightarrow b$

$3 \rightarrow c$

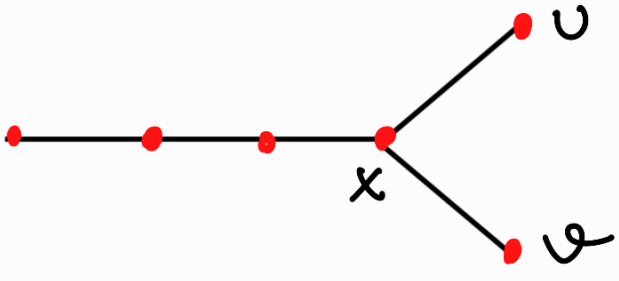
$4 \rightarrow d$

It is apparent that two isomorphic graphs must have :-

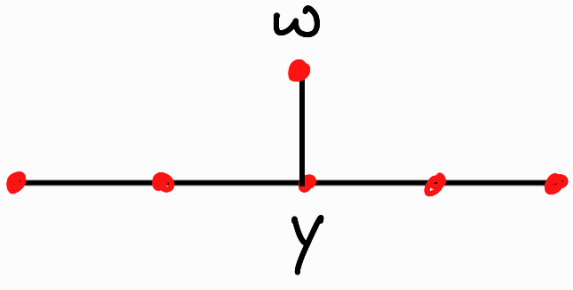
1. The same number of vertices
2. The same number of edges
3. An equal number of vertices with a given degree.

However, these conditions are by no means sufficient.

For eg.



G_1



G_2

Are they ISOMORPHIC ?

The two graphs shown satisfy all the 3 conditions mentioned above, yet they are Not Isomorphic

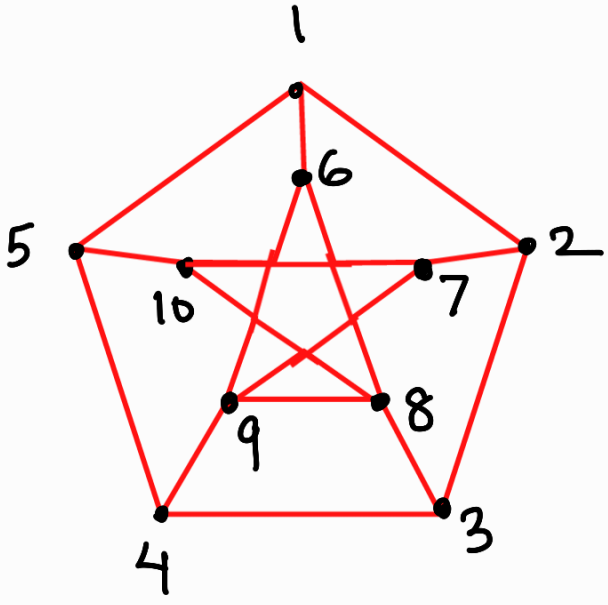
Graph G_1 has a vertex X of degree 3 which is adjacent to two pendant vertices U & V and one vertex of degree 2.

But, in G_2, the vertex y of degree 3 is adjacent to only one pendant vertex w and two vertices of degree 2.

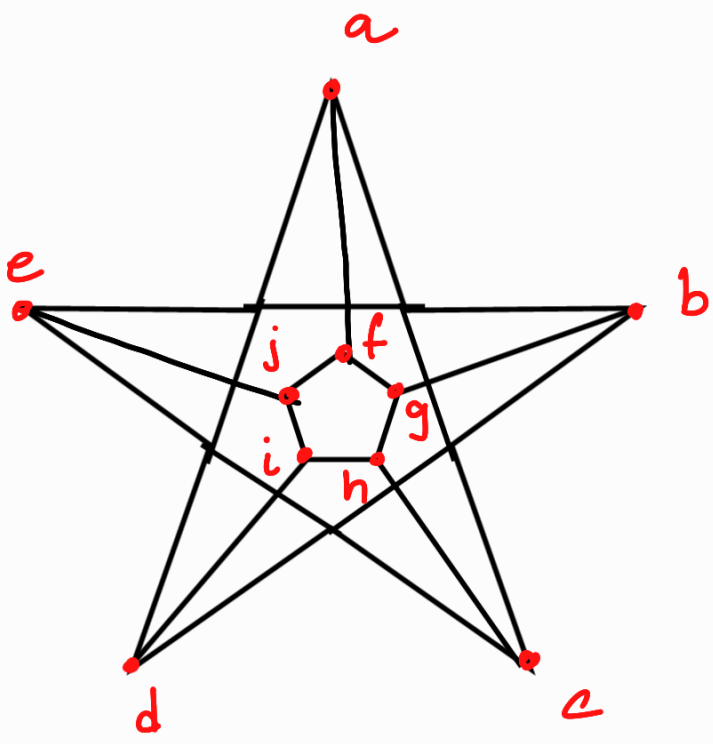
Hence, adjacency is not preserved

Therefore $G_1 \neq G_2$

Q.1. Show that the following graphs are isomorphic.

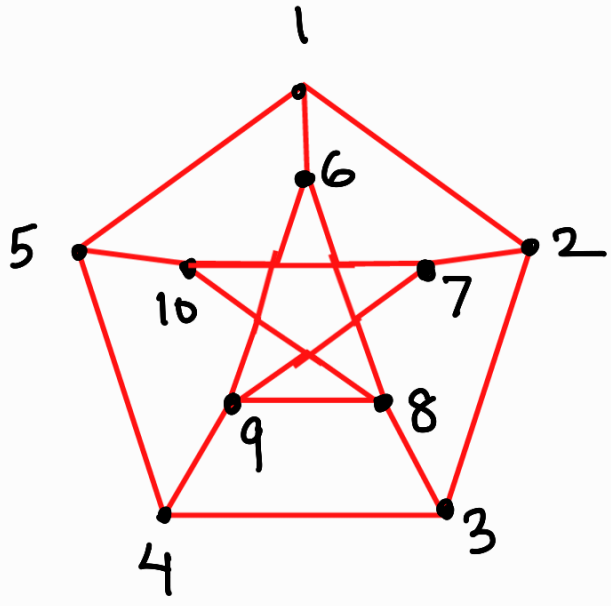


G1

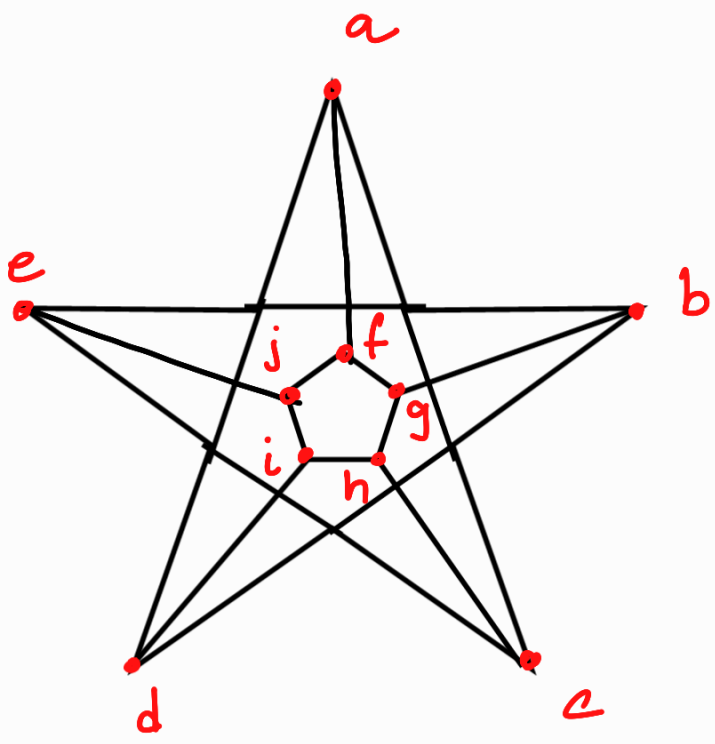


G2

Q.1. Show that the following graphs are isomorphic.



G1



G2

Here, both the graphs contain 10 vertices & 15 edges.

The number of vertices of degree 3 are 10 in both the graphs.

Also, the adjacency is preserved.

The one-to-one correspondence between vertices is given by:-

Function $f : V \longrightarrow V^*$

where

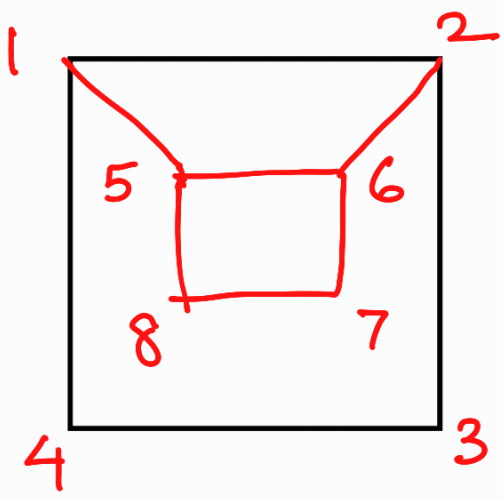
$$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

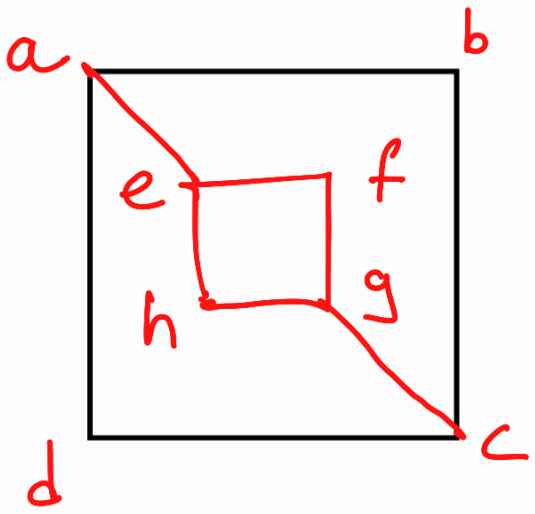
$$V^* = \{a, b, c, d, e, f, g, h, i, j\}$$

Such that $\{a, b\}$ is an edge in G_1 if & only if $\{f(a), f(b)\}$ is an edge in G_2

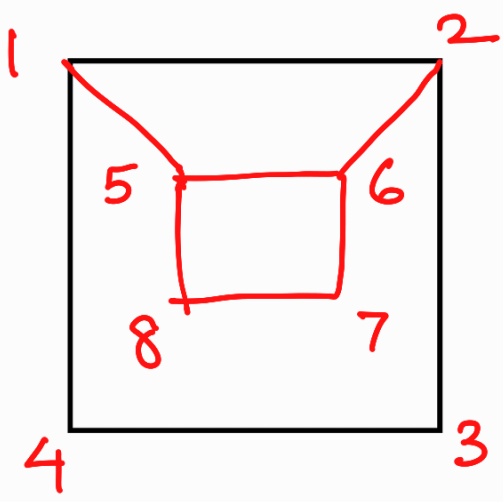
$\therefore f = \{(1, f), (2, g), (3, h), (4, i), (5, j), (6, a), (7, b), (8, c), (9, d), (10, e)\}$

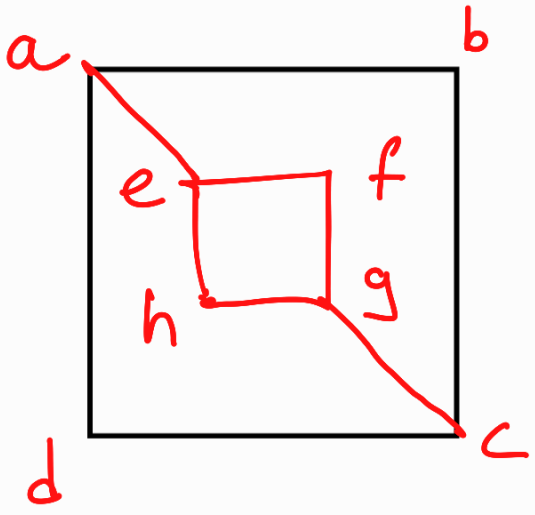
Q.2





Q.2





Here, both the graphs G_1 & G_2 contain 8 vertices & 10 edges.

The number of vertices of degree 2 in both the graphs are 4.

Also the number of vertices of degree 3 in both the graphs are 4.

For adjacency -

Consider vertex 1 of degree 3 in G_1.

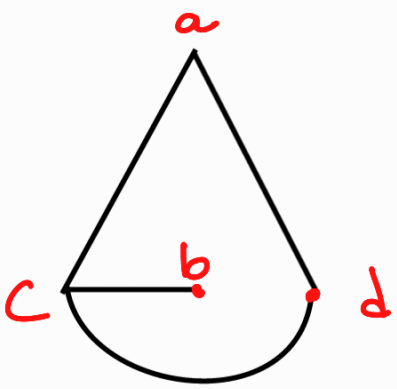
It is adjacent to 2 vertices of degree 3 & 1 vertex of degree 2.

But in G_2, there does not exist any vertex of degree 3 which is adjacent to two vertices of degree 3 & 1 vertex of degree 2.

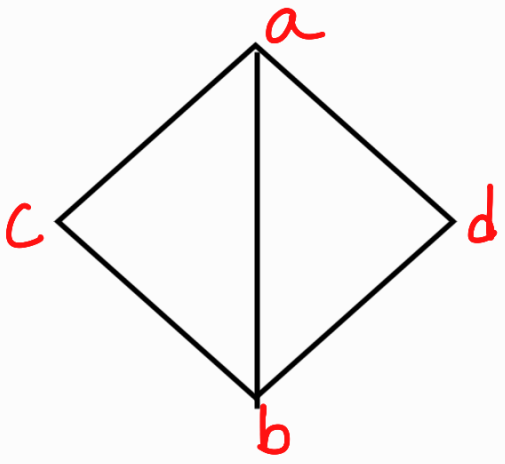
Hence the adjacency is not preserved.

Hence the given graphs are NOT ISOMORPHIC

3)

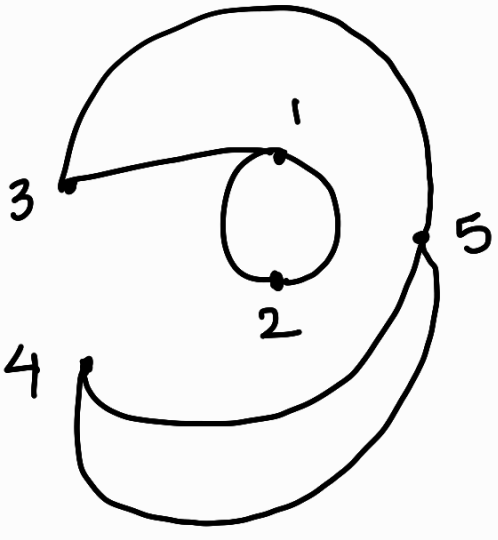


G_1

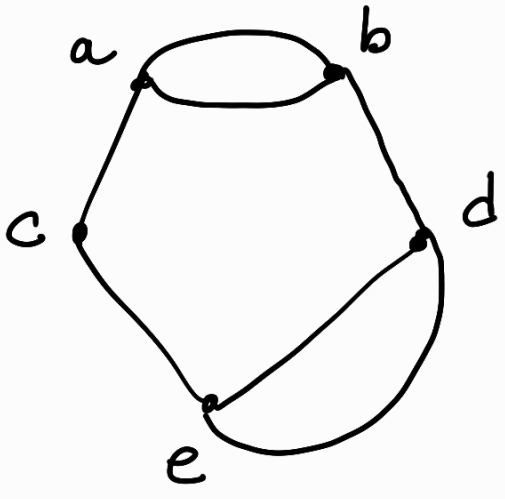


G_2

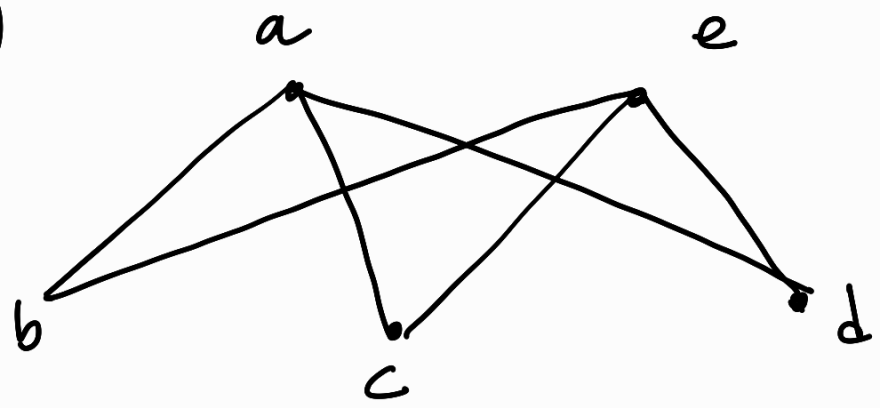
4)



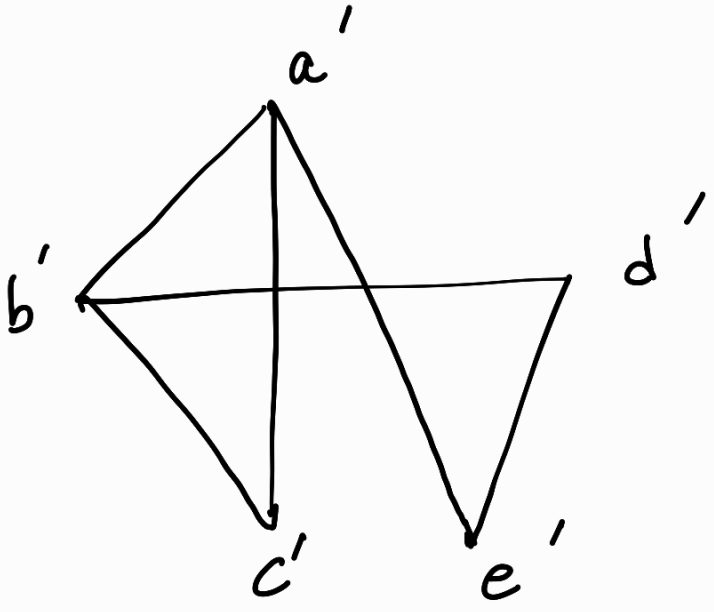
G_1



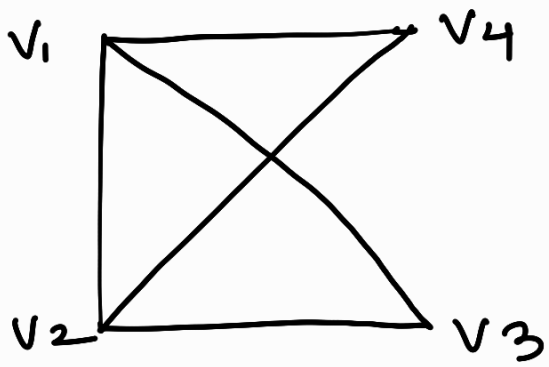
5.)

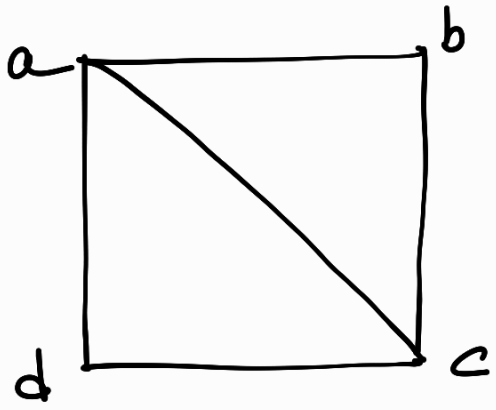


G_1



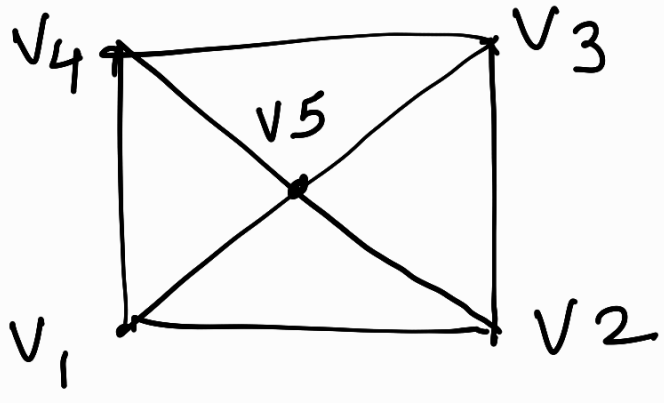
6)

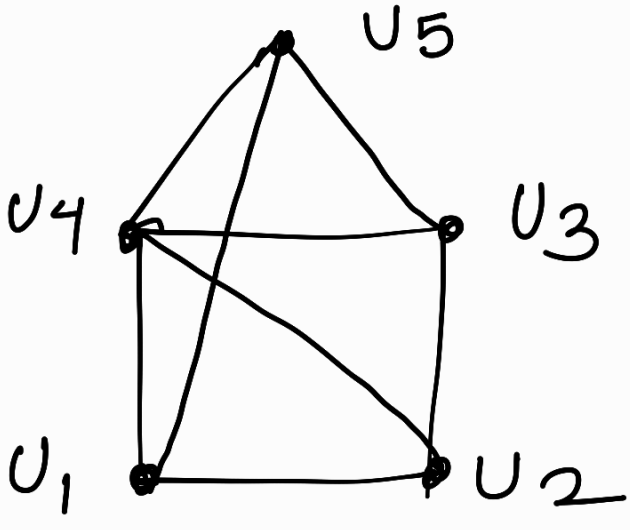




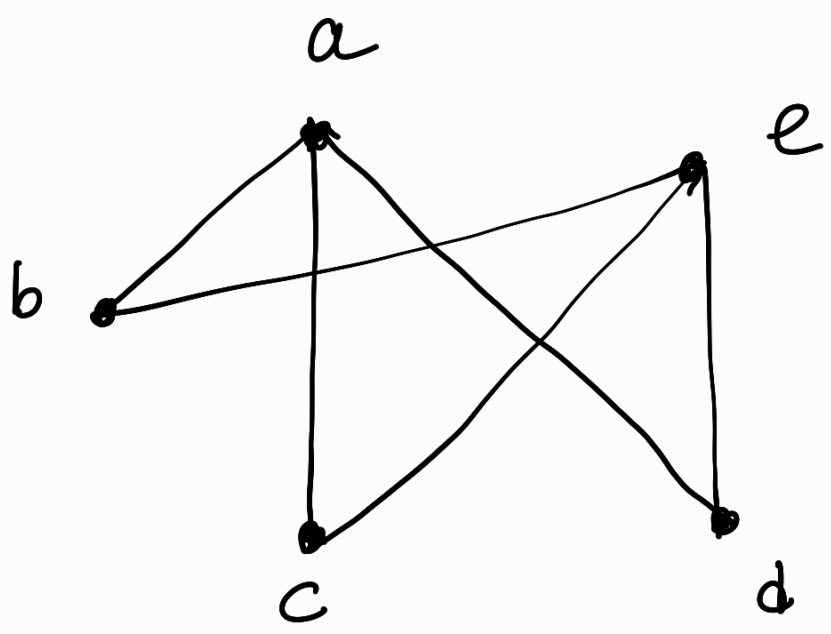
Unsolved (Practice)

1)

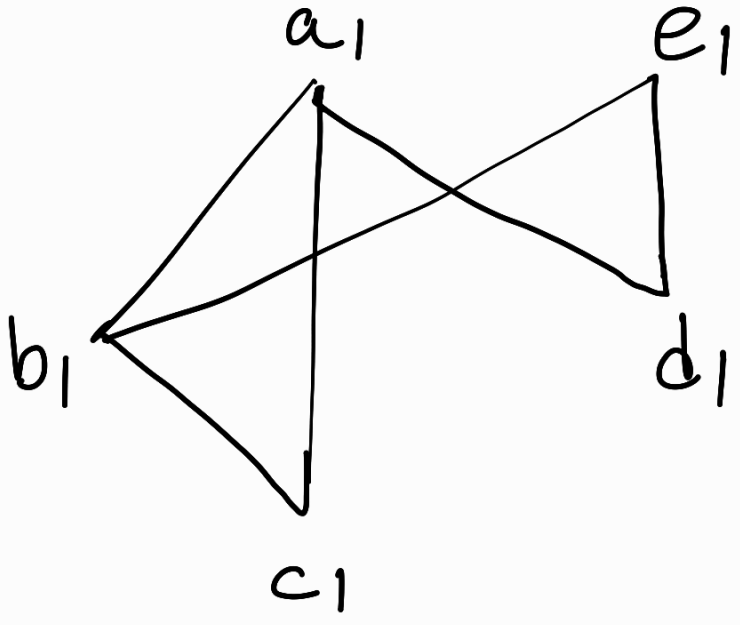




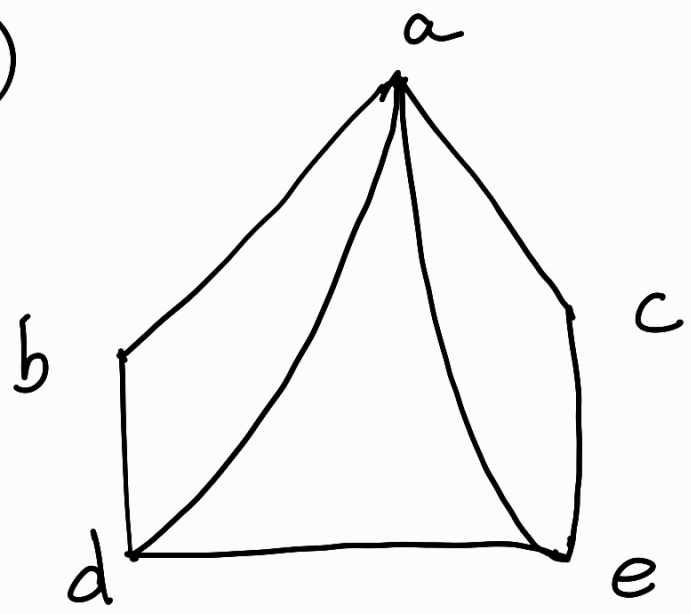
2)



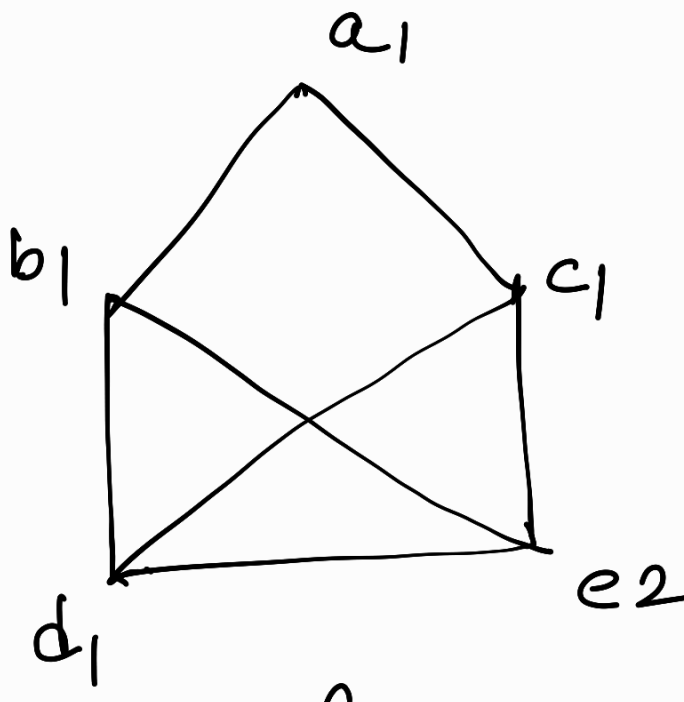
G_1



3)



G_1



G_2

Graph Colouring

A vertex colouring or simply, a coloring of G is an assignment of colours to the vertices of G such that adjacent vertices have different colours.

We say that G is n-colourable if there exists a colouring of G which uses n colors.

The minimum number of colors needed to paint G is called the chromatic number of G and is denoted by $\chi(G)$.

Graph Coloring problems may have almost any meaning.

For ex: If the graph represents a connected grid of cities, each city can be marked with the name of the airline having the most flights to & from that city.

In this case, the vertices are the cities & the colours are airline names.

Welch - Powell Algorithm

Step 1. Order the vertices of G according to decreasing degrees.

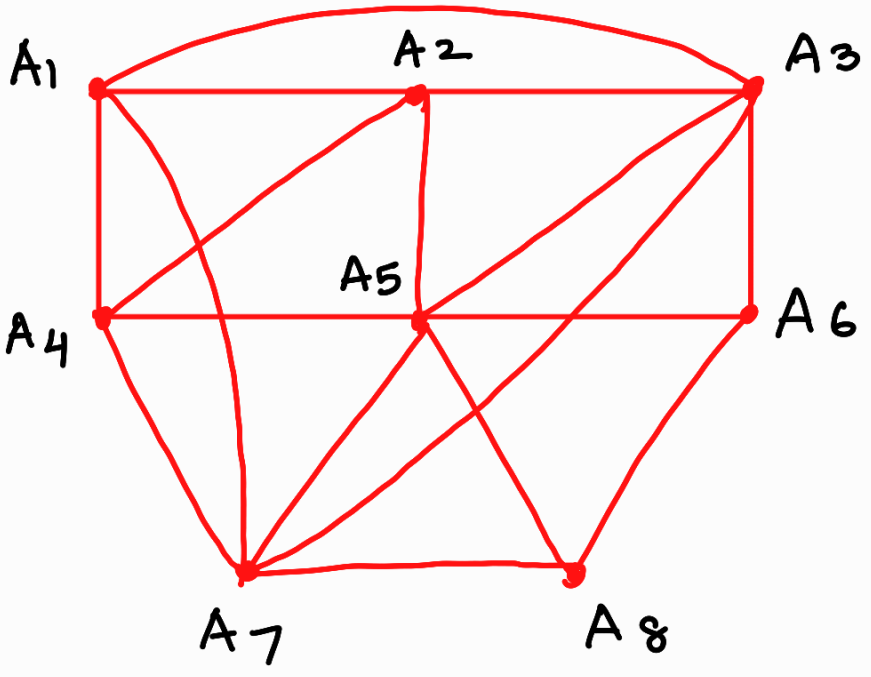
Step 2. Assign the first color C_1 to the first vertex and then, in sequential order, assign C_1 to each vertex which is not adjacent to a previous vertex which was assigned C_1.

Step 3. Repeat Step 2 with a 2nd colour C_2 & the subsequence of non-coloured vertices

Step 4. Repeat Step 3 with a third colour C_3, then a fourth colour C_4 and so on until all vertices are colored.

Step 5. Exit

Q.1. Consider the graph shown below. Use Welch – Powell algorithm to obtain a coloring of G.



Step 1: Order the vertices

$A_5, A_3, A_7, A_1, A_2, A_4, A_6, A_8$

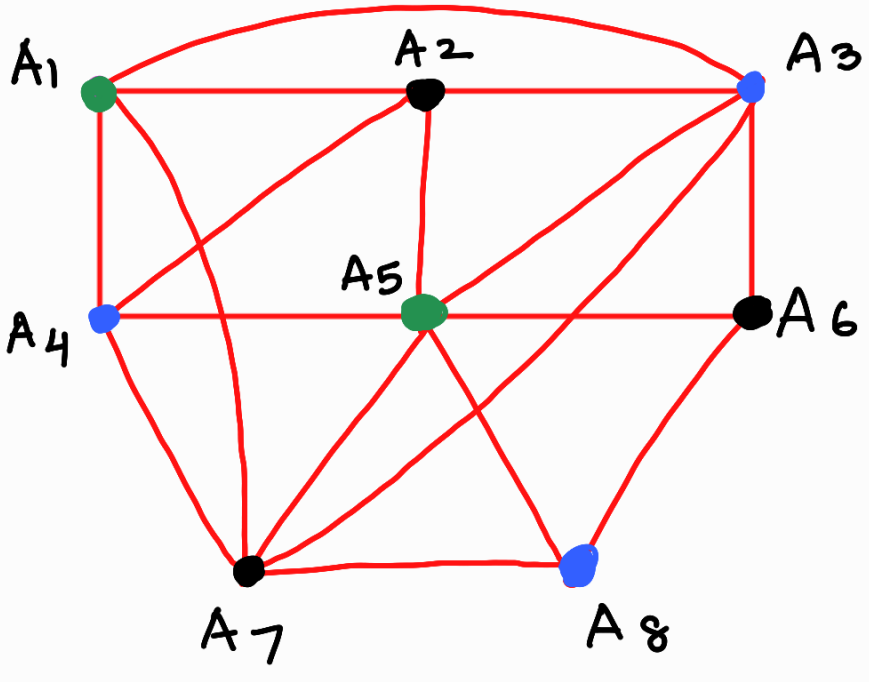
Step 2 to 4: The first colour C_1 is assigned to vertices A_5, A_1.

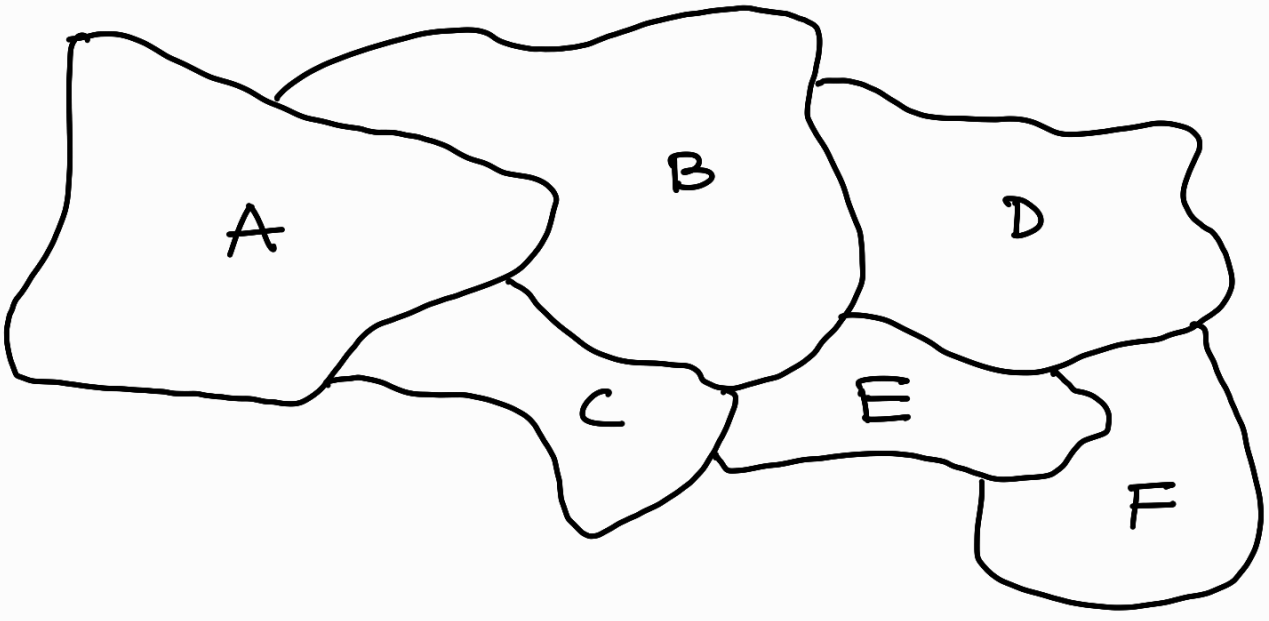
The 2nd colour is assigned to vertices A3, A4, A8

The 3rd colour is assigned to A7, A2, A6

All the vertices have been assigned a color & so G1 is 3-colourable.

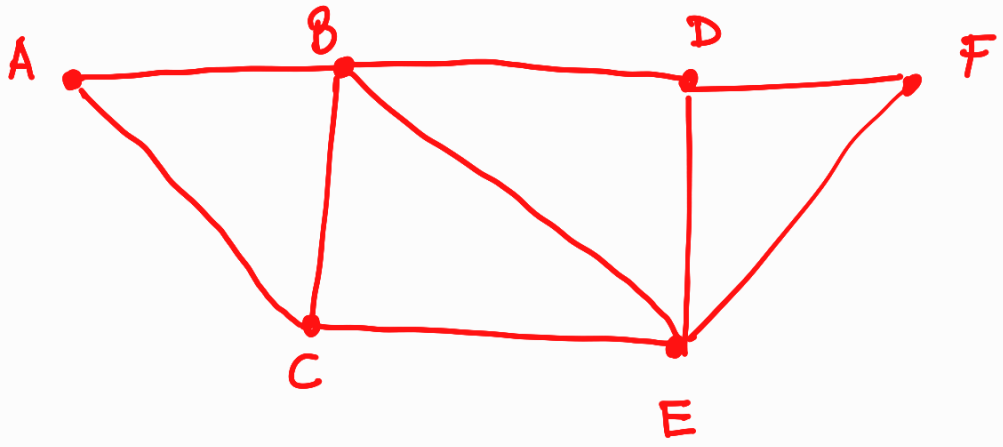
$$\therefore \quad \chi(G_1) = 3$$





Given a map M, construct a graph G_M with one vertex for each region and an edge connecting any two vertices whose corresponding regions share a common boundary.

Then the proper colorings of G_M correspond exactly to the colorings of M.



Now Apply Welch-Powell algorithm to find the chromaticity.

Vertex	Degree
A	2
B	4
C	3
D	3
E	4
F	2

Vertices in descending order of degrees

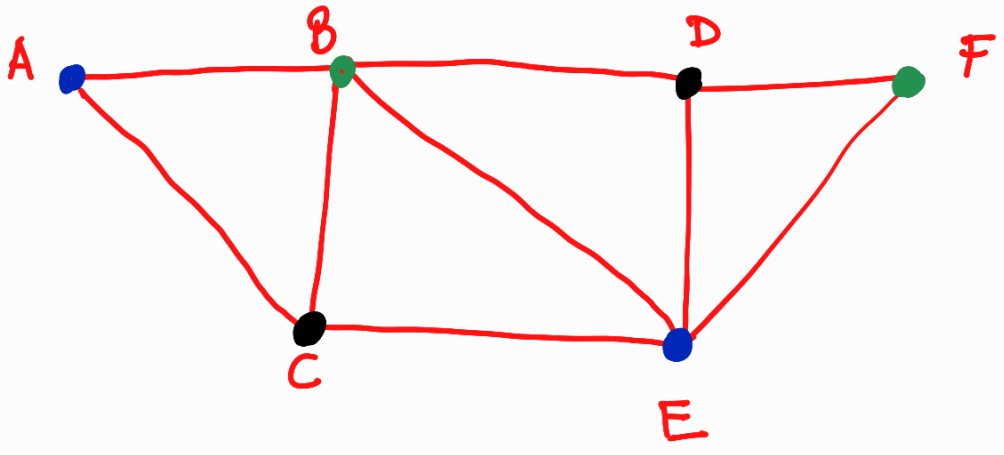
B, E, C, D, A, F

4 4 3 3 2 2

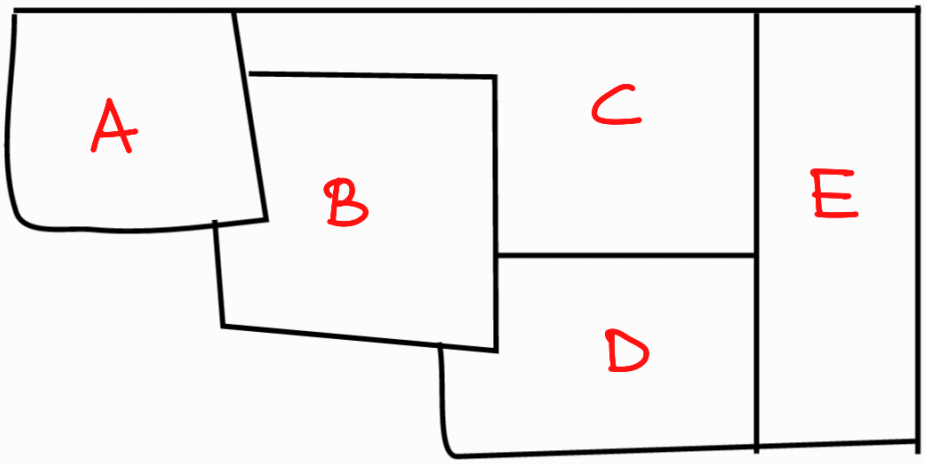
Assign Colour 1 to B & F.

Assign Colour 2 to E & A

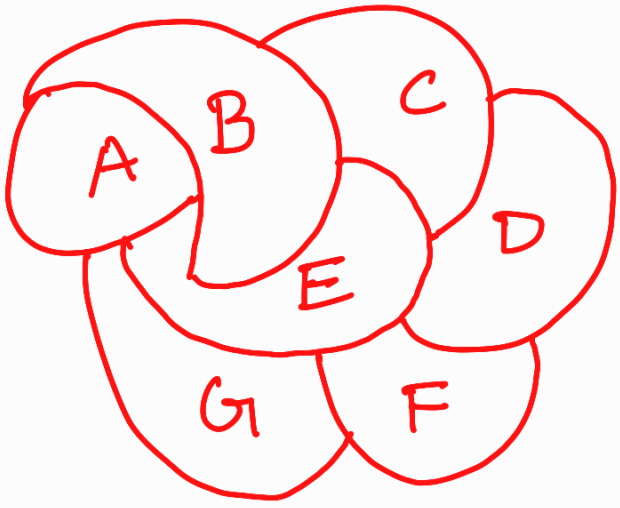
Assign Colour 3 to C & D



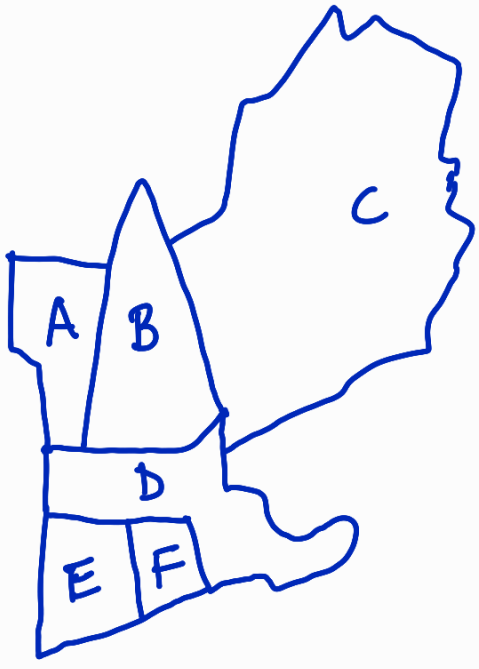
Q. 3.



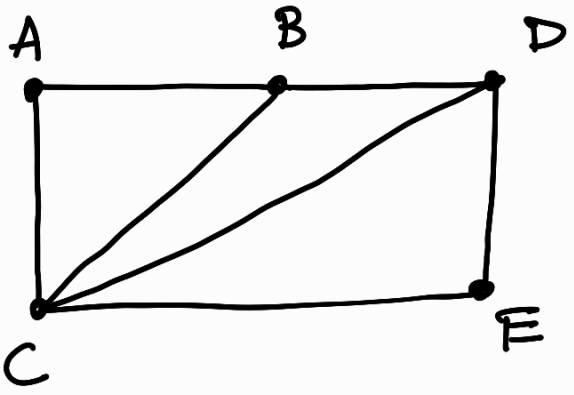
Q.4.



Q.5.



Sol. 3



Vertex	Degree
A	2
B	3
C	4
D	3

∴ Vertices in decreasing order of degrees

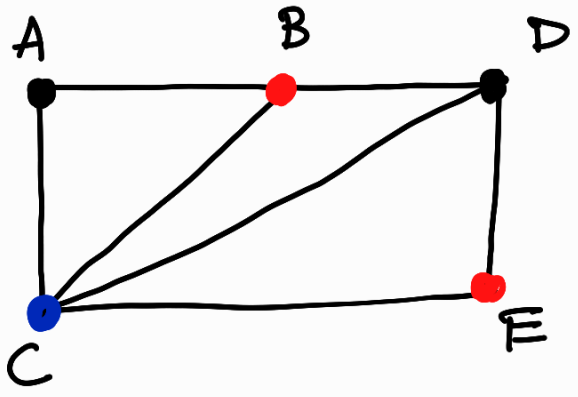
⇒ C, B, D, A

4 3 3 2

Assign Colour 1 to C.

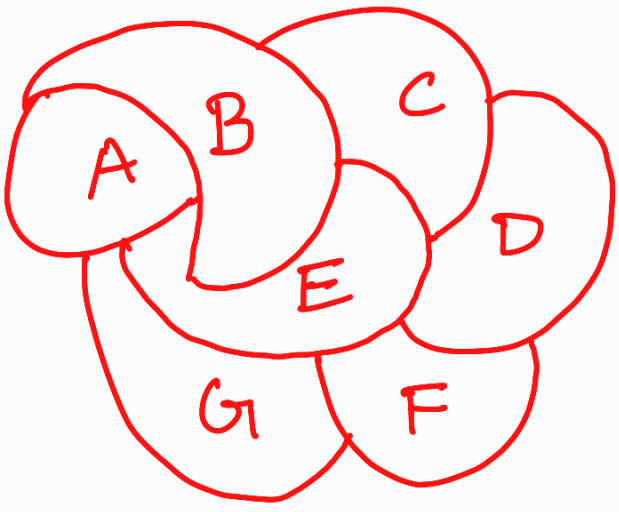
Assign Colour 2 to B & E

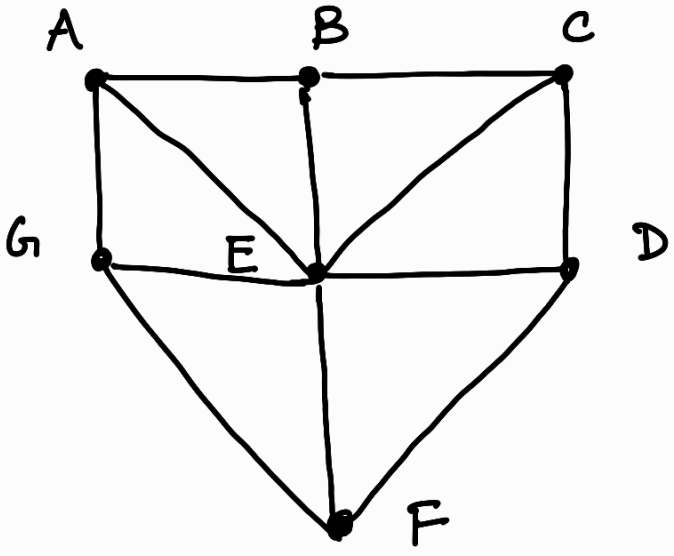
Assign Colour 3 to D & A



$\therefore \quad \chi(G) = 3$

Q.4





Vertex	Degree
A	3
B	3
C	3
D	3
E	6
F	3
G	3

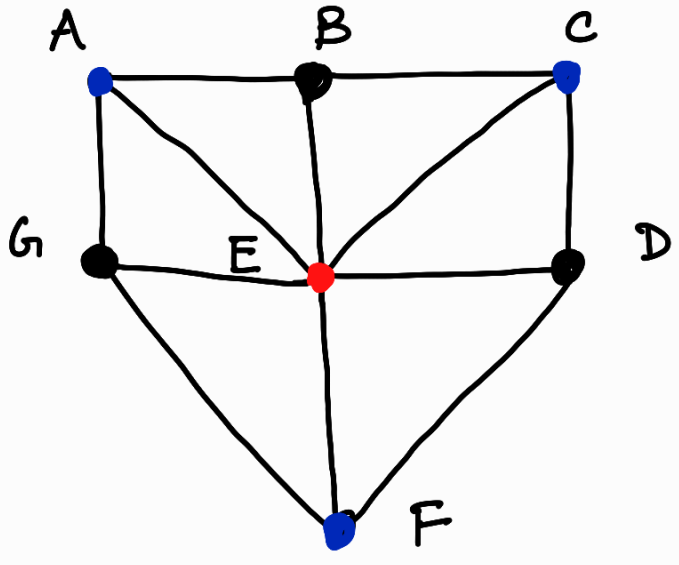
Therefore sequence of vertex is:-

E, A, B, C, D, F, G

We assign colour 1 to vertex E.

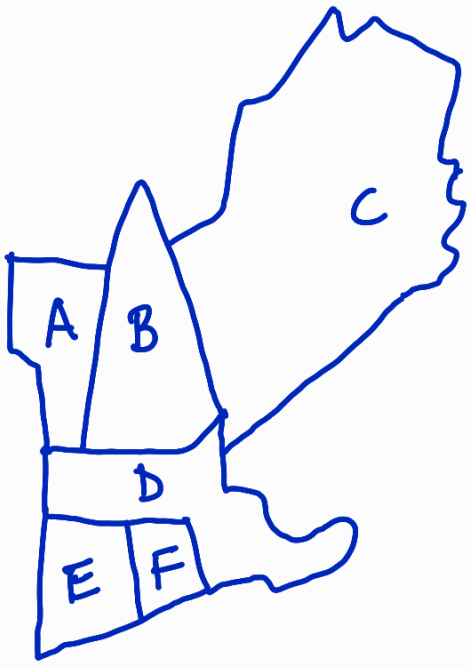
We assign colour 2 to vertex A, C, F

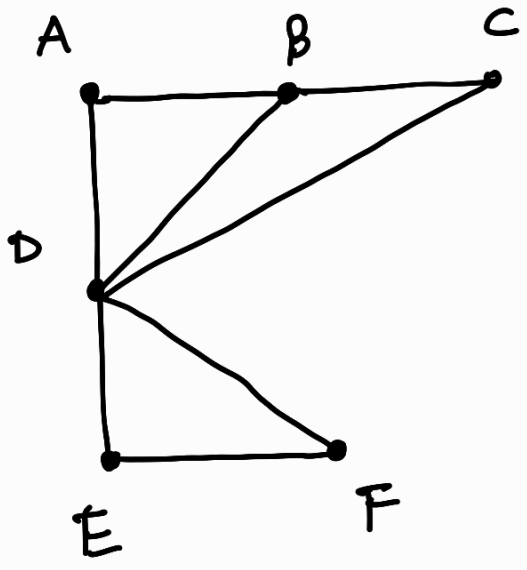
We assign colour 3 to vertex B, D, G



$\therefore \quad \chi(G) = 3$

Q.6.





Vertex	Degree
A	2
B	3
C	2
D	5
E	2
F	2

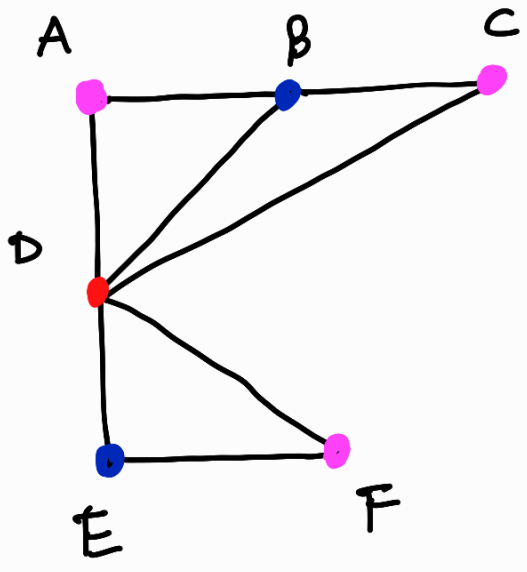
∴ Vertices in Sequence would be

D, B, A, C, E, F

We assign colour 1 to D.

We assign colour 2 to B & E

We assign colour 3 to A, C & F

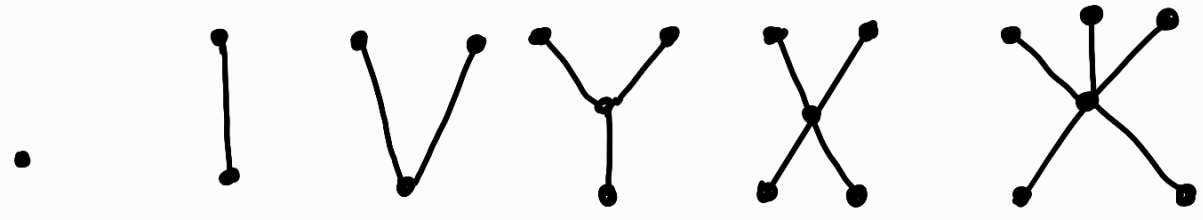


$\therefore \quad \chi(G) = 3$

Trees

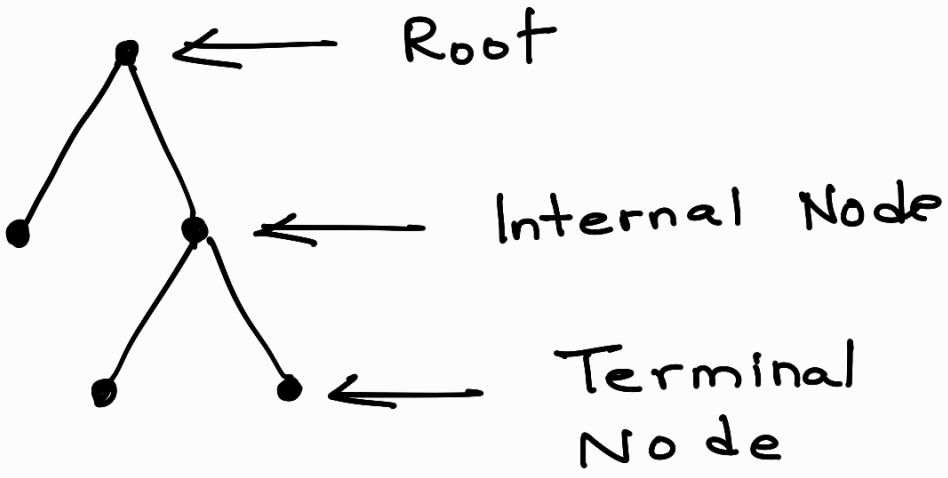
A connected / non isolated graph which does not have a circuit or cycle is called a tree.

For example



A vertex of degree 1 is called a terminal node or a leaf.

A vertex of degree greater than 1 is called an internal node or a branch node.



Isomorphic Trees

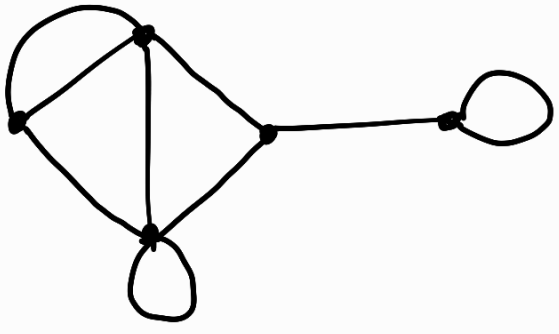
The same rules which are applicable to graphs are applicable to trees as well.

Spanning Trees

Let G be a graph. A subgraph G' of G is called a spanning subgraph if G and G' have the same set of vertices.

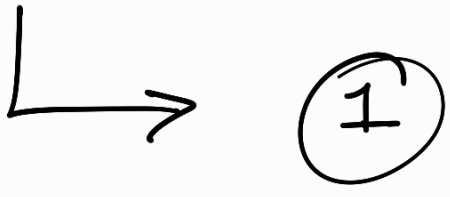
In particular, Let G be a connected graph. A spanning subgraph of G which is a tree is called a spanning tree.

Example Find the spanning trees of the graph.



The given graph has $n = 5$ vertices & edges $e = 9$

Every spanning tree has n vertices & $(n-1)$ edges



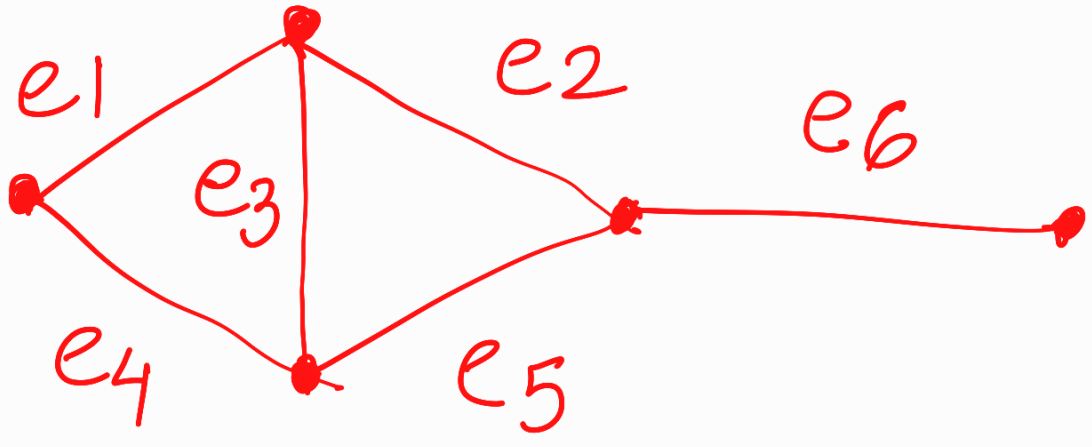
$\therefore$ We must have $n = 5$ & $(n-1) = 4$

Vertices Edges

$\therefore$ We have to remove $9 - 4 = 5$ edges.

We first remove the two loops. Then the multiple edges & get the graph as shown below:-

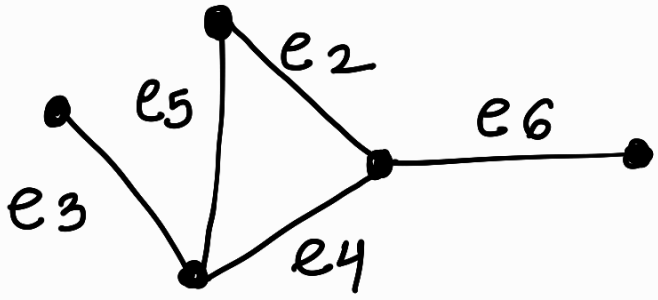
Step 1: Remove Cycles



G_1

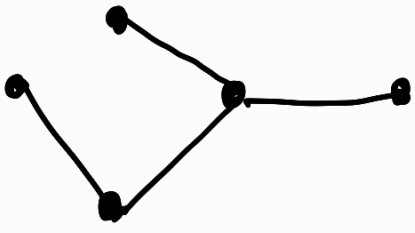
Step 2: Remove the edges one by one to break the cycles.

We first remove the edge e_1 to break the cycle on the left and get the following graph.

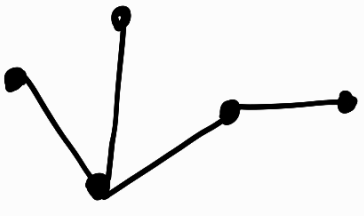


G_2

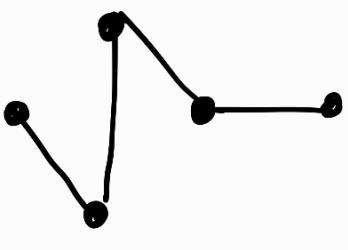
Step 3: Now, we remove the edge e_5 or e_2 or e_4 to break the 2nd cycle as follows:-



G_3'



G_3''



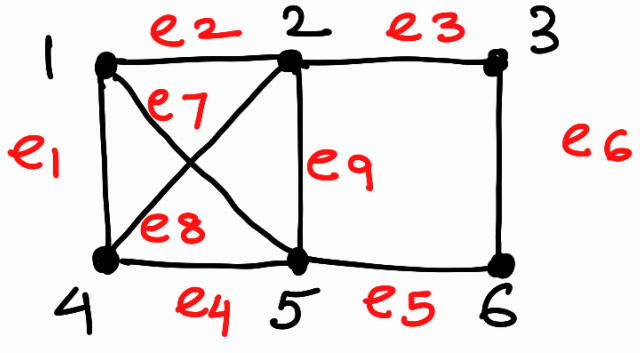
G_3'''

G_3', G_3'' & G_3''' are all spanning trees.

(Not the only possible solution)

solutions may vary if the starting edge to be removed varies.

Q.2 Find Spanning Trees for the following graph.



Sol: No. of Vertices = 6

No. of Edges = 9

According to ①,

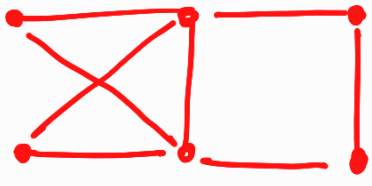
No. of edges in the spanning tree should be $(n-1) \Rightarrow 6-1 \Rightarrow 5$

So, first remove cycles if any & then start removing edges starting from each edge creating a cycle.

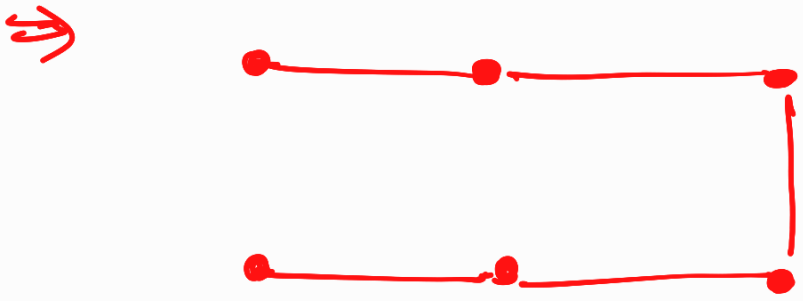
We need to remove

$9 - 5 = 4$ edges.

We start with e_1



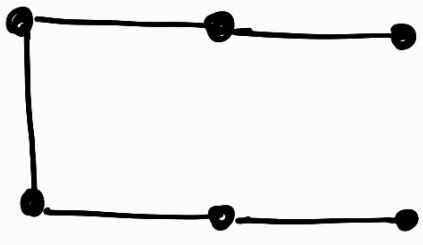
Then remove all those edges which may create a cycle

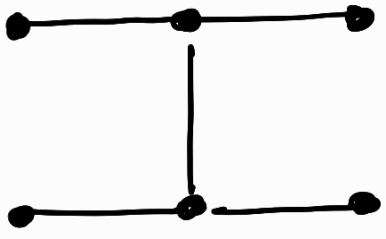


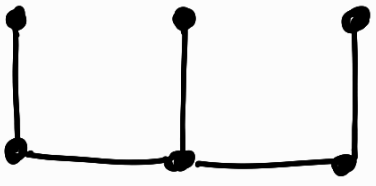
This is one of those solutions.

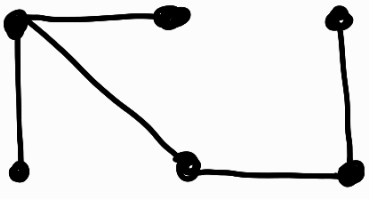
Obtain the remaining spanning trees as well. (Atleast 5-6)

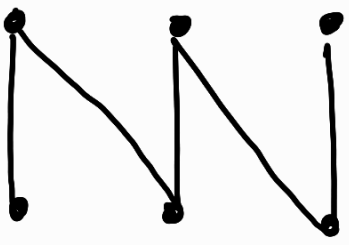
depends on Question

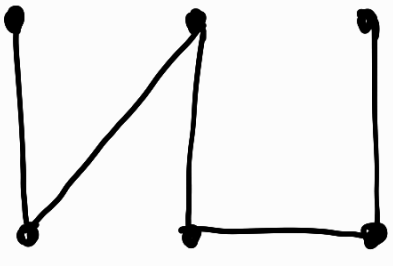












Minimal Spanning Tree

Let $G = (V, E)$ be a connected graph. Let $e \in E$ & $C(e)$ be a non-negative number associated with e, g_t is called the cost or the weight of e.

When costs or weights thus are associated with all edges, the graph is called a weighted graph.

A spanning tree T of the graph (G) such that the total cost $\sum C(e)$ is minimum is called as a **minimal spanning tree.**

Krustkal's Algorithm.

Suppose we are given a connected weighted graph. First of all, name all the vertices.

Now, plot all the vertices on a plane.

Also list all the edges in the increasing order of their weights

Step 1: Get the first edge by joining the vertices topping the above list.

Step 2: Get the second edge by joining the vertices second in the list.

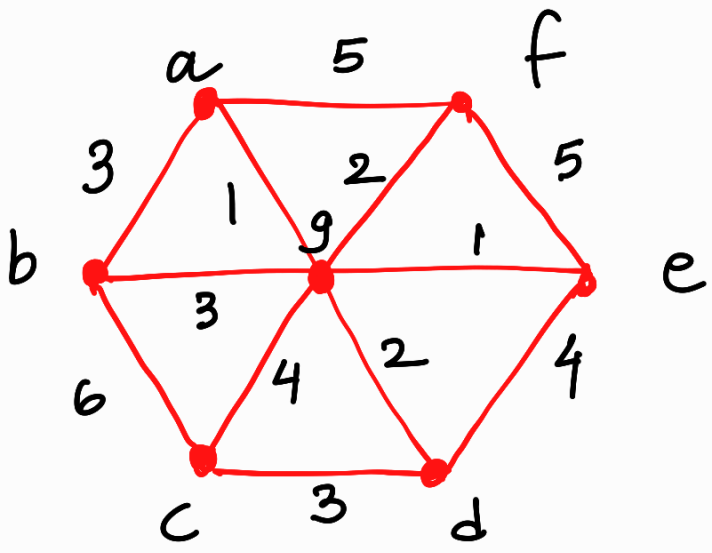
Step 3: Consider the next edge in the list.

If it does not form a cycle, then join the vertices else reject it.

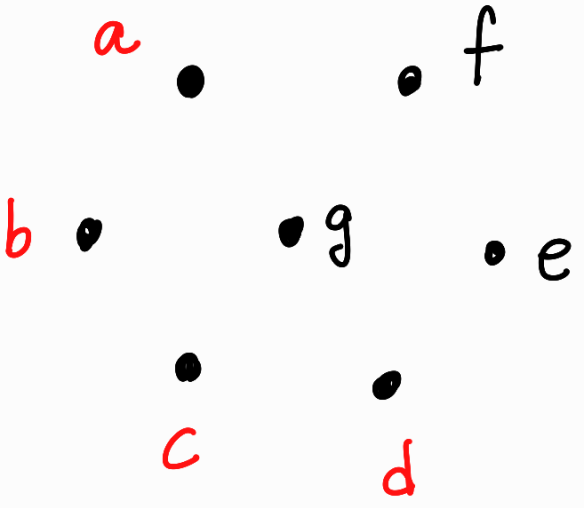
Step 4: Repeat the above step till you get a spanning tree.

This is the required Minimal Spanning Tree

Q.1.



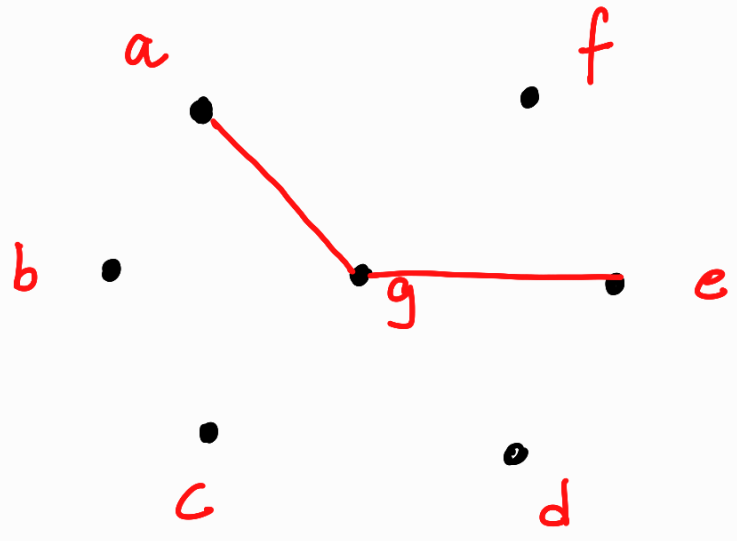
First name the vertices & plot them.



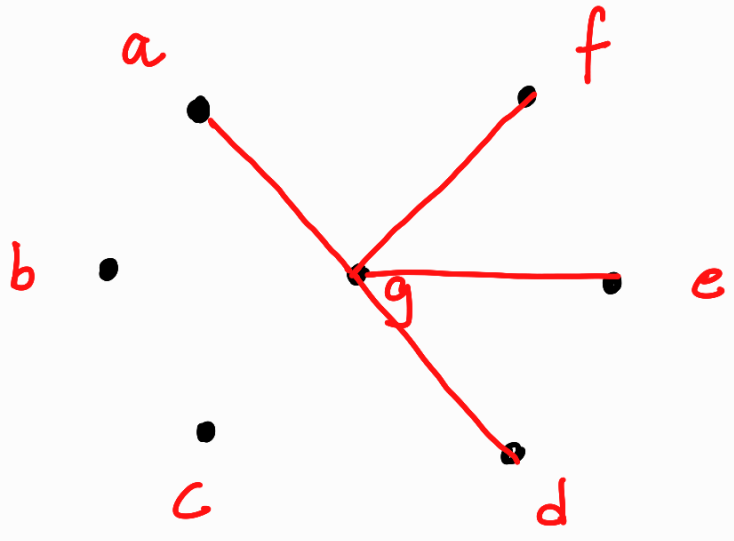
Now, list the edges in the increasing order of their costs.

1. {a, g}, 2. {g, e}, 3. {g, f},
4. {g, d}, 5. {g, b}, 6. {b, a}
7. {c, d}, 8. {c, g}, 9. {d, e}
10. {e, f}, 11. {a, f}, 12. {b, c}

Step 1: Since, there are 2 pairs of vertices $\{a, g\}$ & $\{g, e\}$ with minimum cost, we join them.



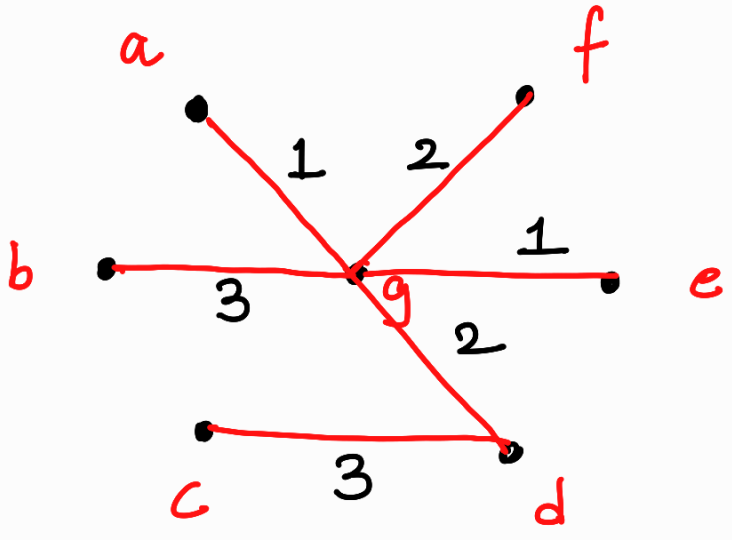
Step 2: The two pairs of vertices having weight 2 are $\{g, f\}$ & $\{g, d\}$. Since they don't form cycles, we join them.



Step 3: There are 3 pairs of vertices having weight 3.

They are {g, b}, {b, a} & {d, c}.

We join the vertices g, b & d, c but we won't join the vertex b, a because it will form a cycle. So we reject this pair of vertices.

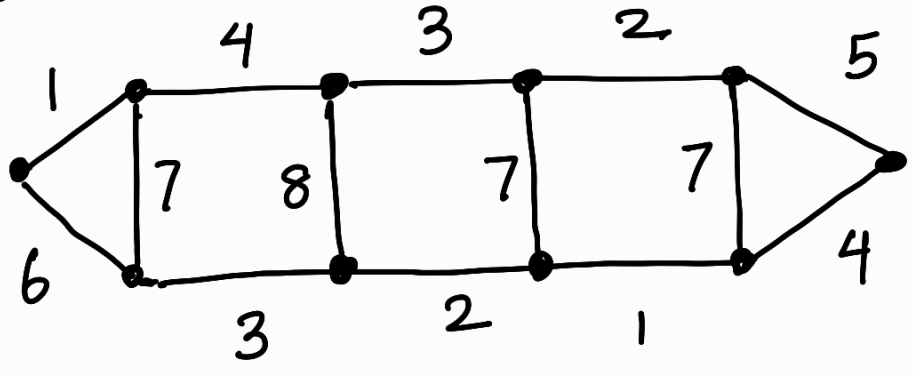


Since, all the vertices are now joined, this is the required minimal spanning tree.

By adding the costs, we get, the minimal cost

$$1+1+2+2+3+3 = 12$$

Q.2



Prim's Algorithm

Step 1: Select any arbitrary vertex, say V_0 as tree root.

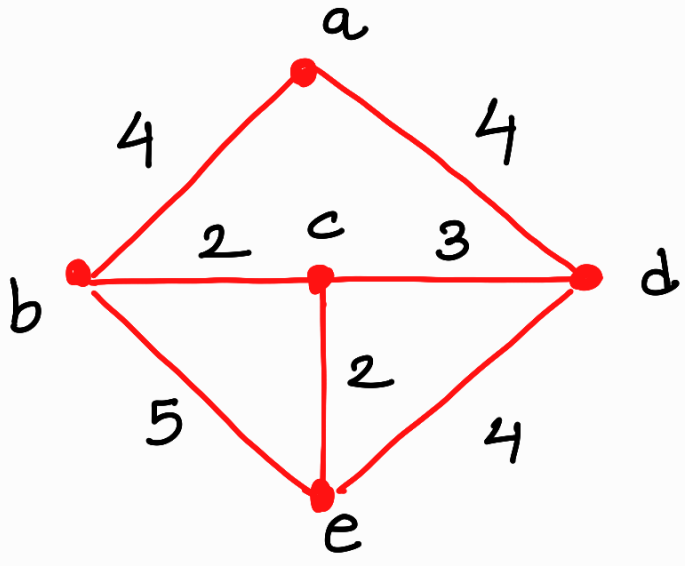
Step 2: Find edge $e_1 = (V_0, V_i)$ in E such that its one end vertex V_0 is in T & it's weight is minimum.

Step 3: Select the next edge $e_i = (V_i, V_j)$ such that its one end vertex V_i is in T and the other end vertex V_j is not in T (e_i should not form a cycle) and also the weight of the edge e_i is as small as possible

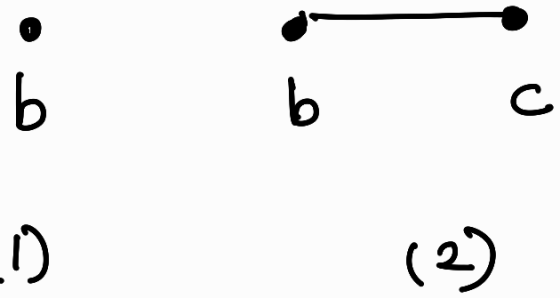
Join the edge e_i & the vertex v_j to T.

Step 4 Repeat the step 3 until all the vertices of G are in T.

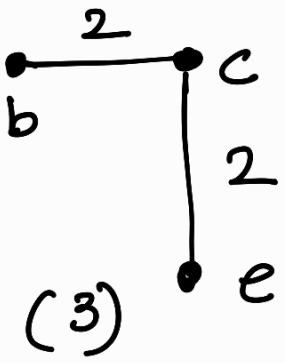
Q.1 Find the minimal spanning tree using Prim's Algorithm.



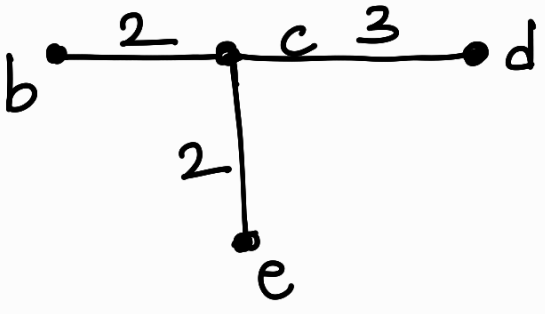
We start with vertex b.



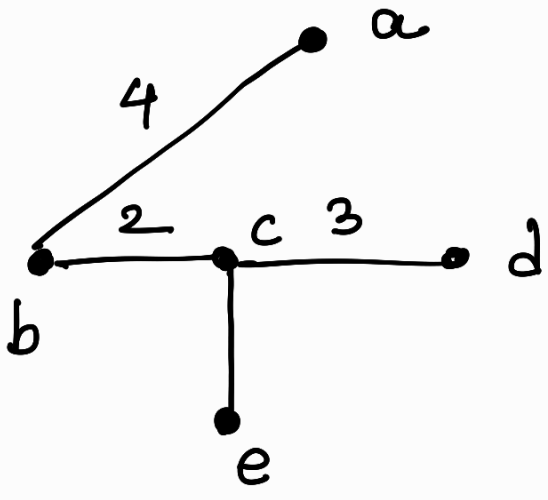
(1) (2)



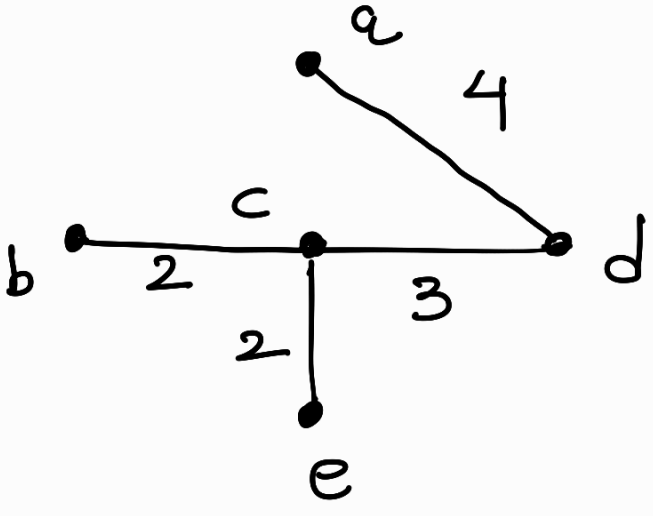
(3)



(4)



OR



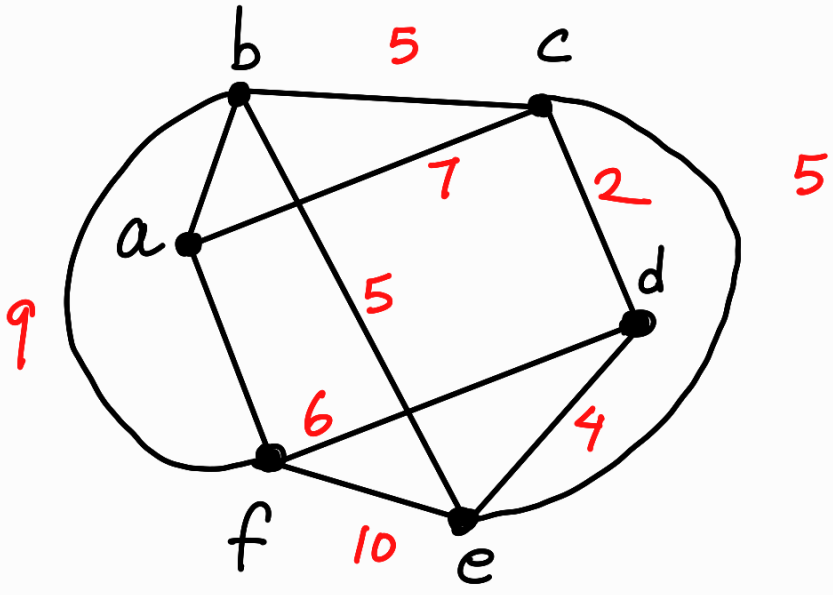
∴

Minimal Cost

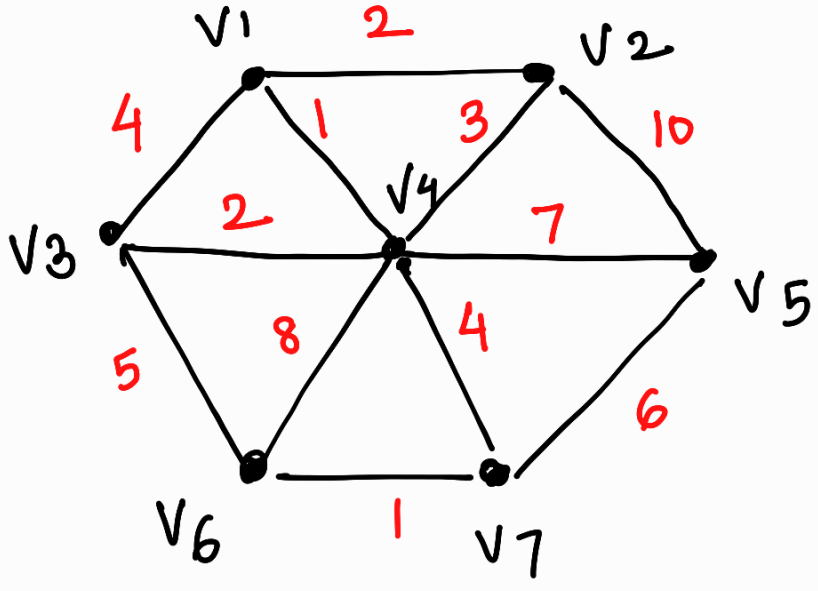
$= 2+2+3+4$

$= 11$

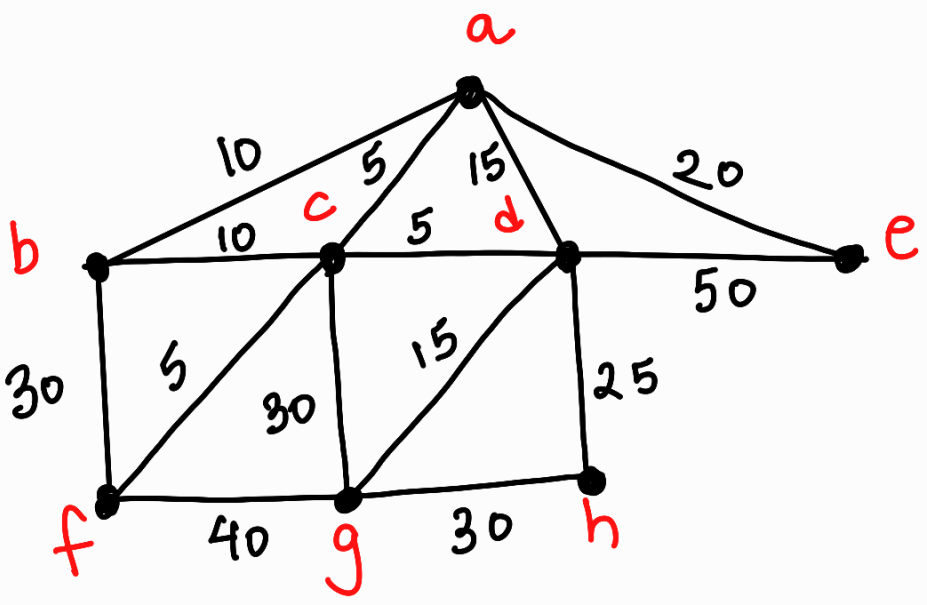
Q.2



Q.3



Q. Use Prim's & Kruskal's algorithm to find the minimal spanning tree.



Rooted Trees

A rooted tree T is a tree graph with a designated vertex r called the root of the tree.

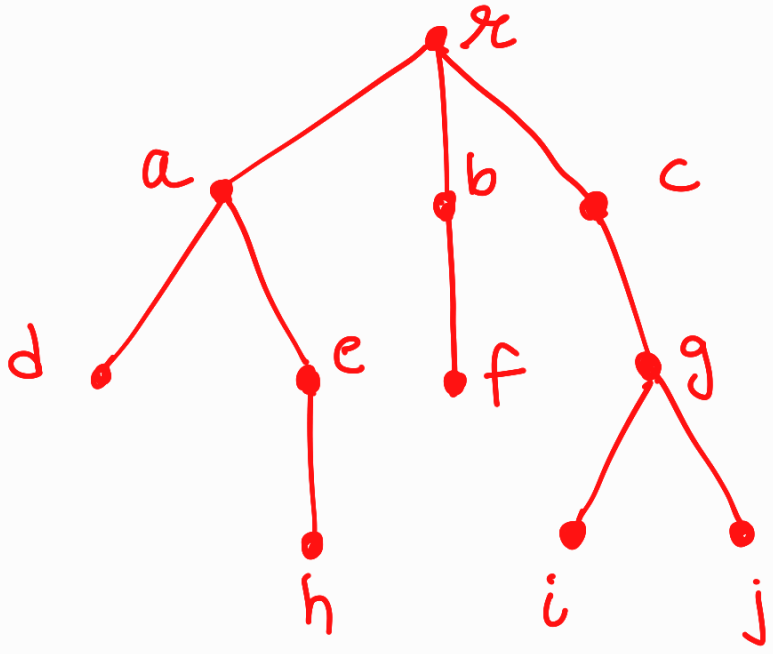
Since there is a unique simple path from the root r to any other vertex v in T, this determines a direction to the edges of T.

Thus T may be viewed as a directed graph.

Any tree may be made a rooted tree by simply selecting one of the vertices of the root.

Consider a rooted tree T with root r.

The length of the path from the root r to any vertex v is called the level (or depth) of v, and the maximum vertex level is called the depth of the tree.



We say that a vertex u precedes vertex v or that v follows u if there is a (directed) path from v to u.

In particular, we say that v immediately follows u (if u, v) is an edge, that is, if v follows u and v is adjacent to u.

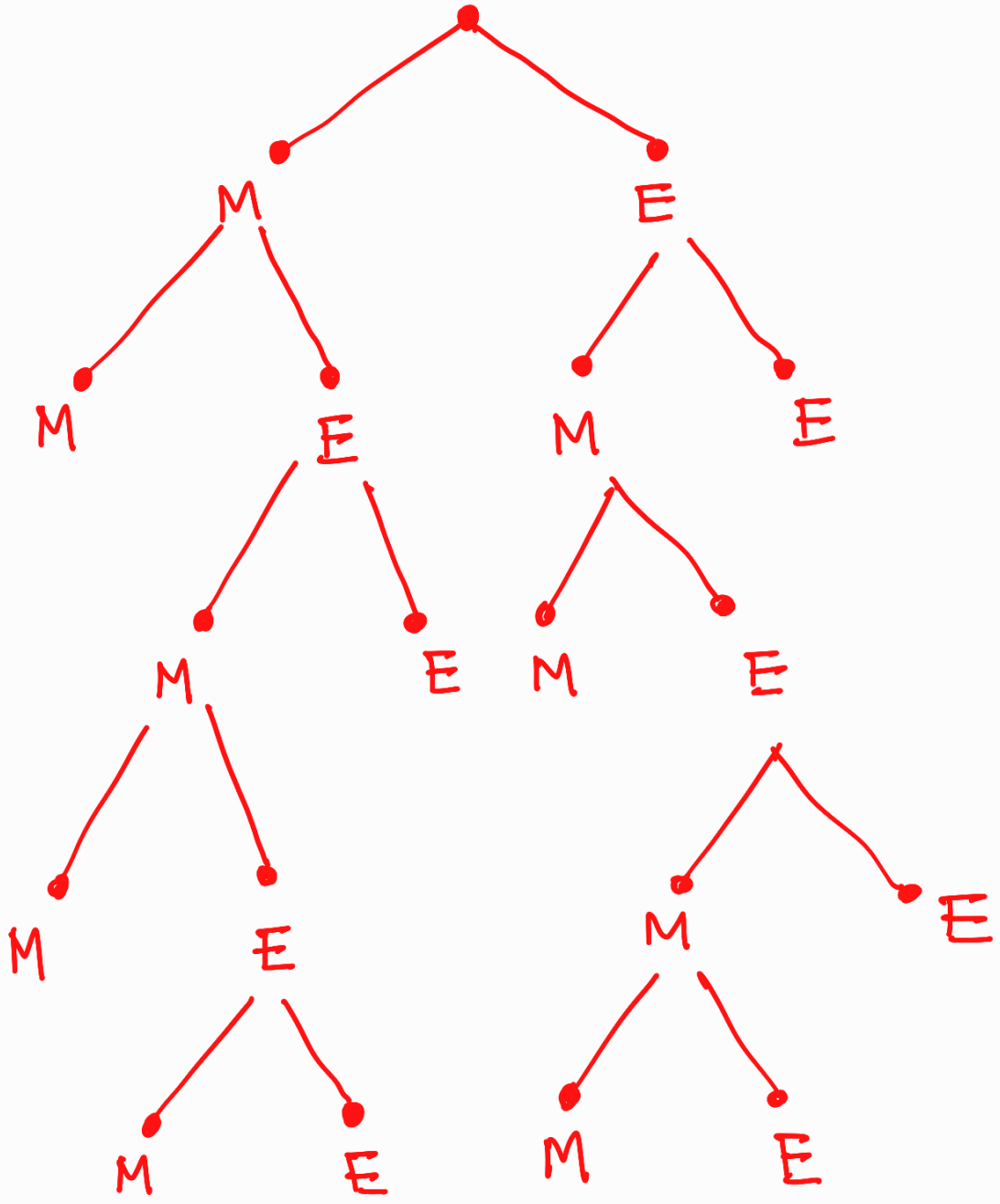
Every vertex v other than the root, immediately follows a unique vertex, but that v can be followed by more than one vertex.

For example,

vertex j follows c but immediately follows g.

Also, both i & j immediately follow g.

Q. Suppose Marc & Eric are playing a Tennis Tournament, such that the first person to win two games in a row or who wins a total of three games wins the tournament. Find the number of ways the tournament can proceed.



There are total 10 Leaf nodes.

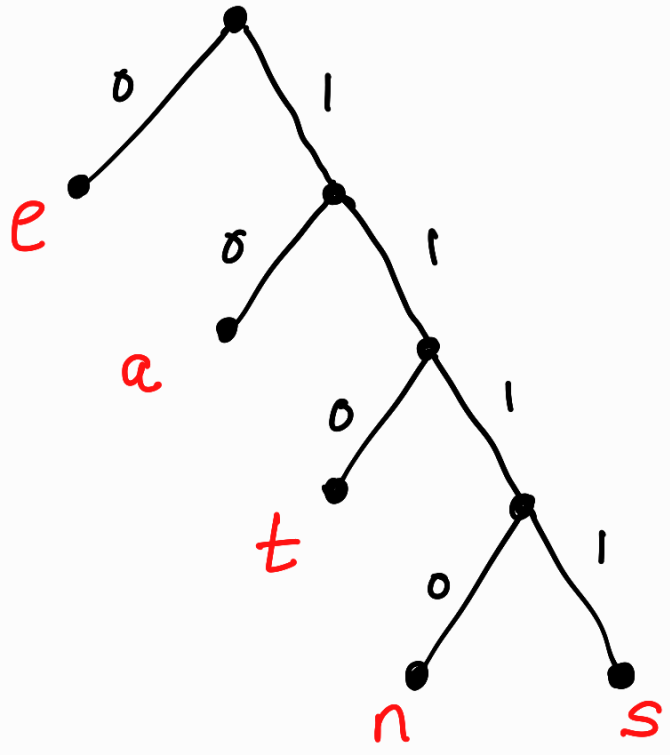
∴ There are 10 different ways in which the tournament can occur. MM, MEE, MEMM, MEMEM, MEMEEE, EE, EMM, EMEE, EMEMM, EMEME

Prefix Codes

A prefix code can be represented using a binary tree, where the characters are the labels of the leaves in the tree.

The edges of the tree are labelled so that an edge leading to a left child is assigned a 0, and an edge leading to the right child is assigned a 1.

The bit string used to encode a character is the sequence of labels of the edges in the unique path from the root to the leaf that has this character as its label.



For example

e → 0

a → 1 0

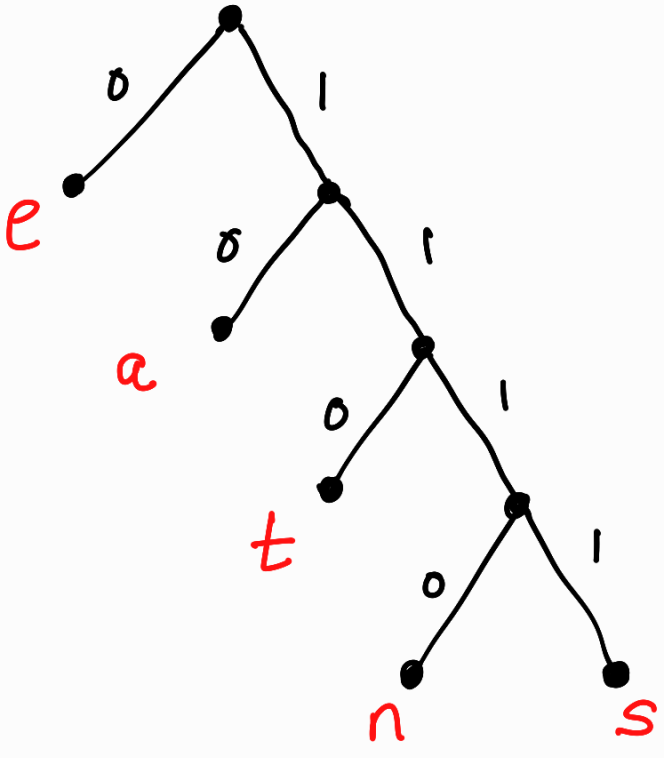
t → 1 1 0

n → 1 1 1 0

s → 1 1 1 1

The tree representing a code can be used to decode a bit string.

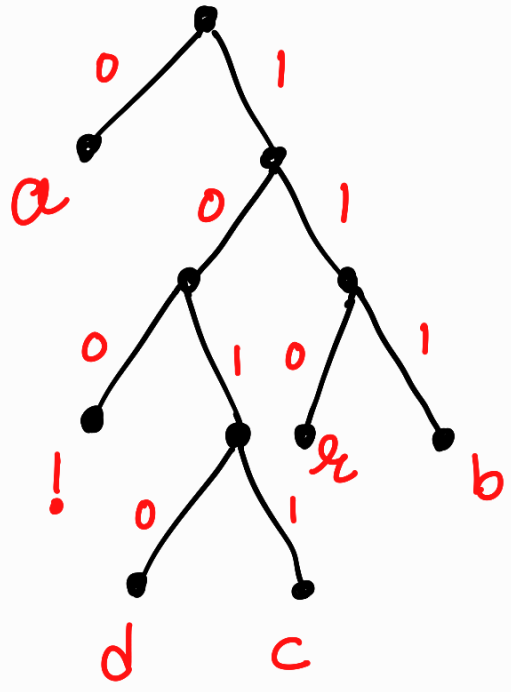
For instance, consider the word encoded by 11111011100 using the code according to the above tree.



11111011100

_____ _ ___ _

Q..



a → 0

b → 111

! → 100

r → 110

d → 1010

c → 1011

Decode:

0 111 110 0 10 110 10 100 1111 100 100

Huffman Tree (Optimal Prefix Code)

Huffman Coding is a losless data compression algorithm.

The most frequent character gets the smallest code & the least frequent gets the largest code.

There are mainly two major parts of the Huffman Coding.

1. Build a Huffman Tree from i/p characters
2. Traverse the Huffman Tree and assign codes to characters.

Steps to build a Huffman Tree

Input is an array of unique characters with their frequency of occurrences and output is the Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes.
2. Extract two nodes with the minimum frequency from the min heap.
3. Create a new internal node with a frequency equal to the sum of the two nodes frequencies.

Make the first extracted node as its left child & the other extracted node as its right child.

Add this node to the heap.

4. Repeat steps #2 & #3 until the heap contains only one node.

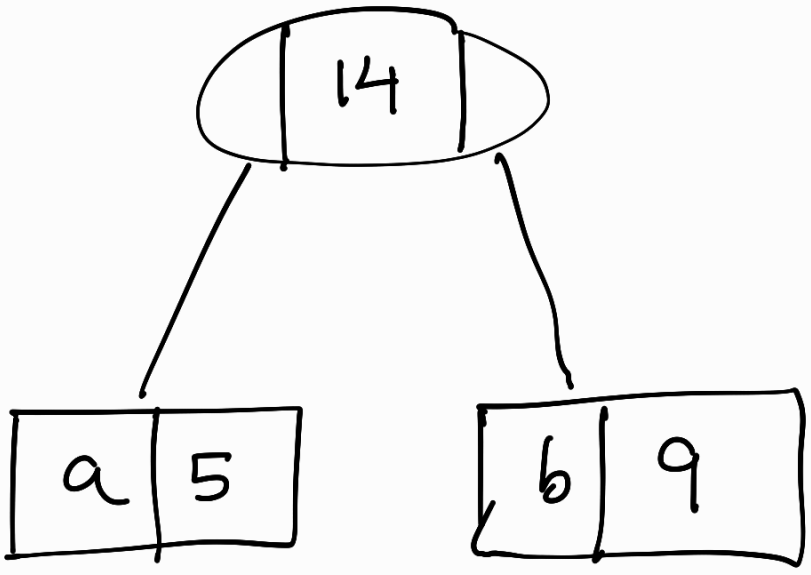
The remaining node is the root node & the tree is completed.

Example

Character	Frequency
a	5
b	9
c	12
d	13
e	16
f	45

Step 1: Build a min heap that contains 6 nodes where each node represents root of a tree with single node.

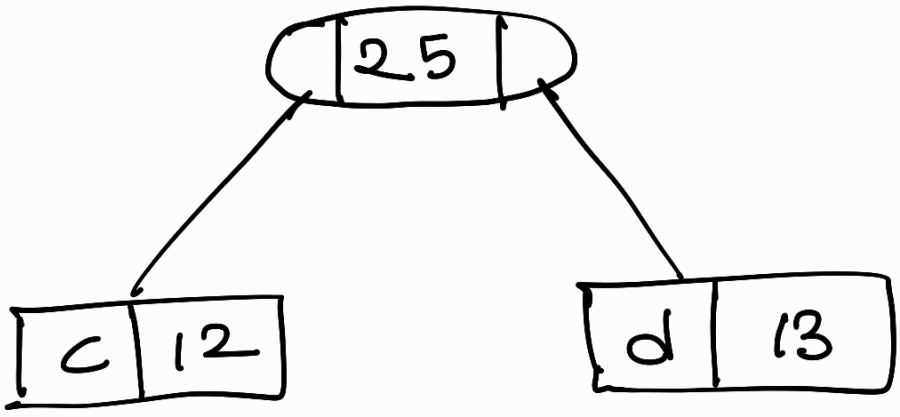
Step 2: Extract two minimum frequency nodes from min heap. Add a new internal node with frequency 5+9 = 14



Now, min heap contains 5 nodes.

Character	Frequency
c	12
d	13
a, b	14
e	16
f	45

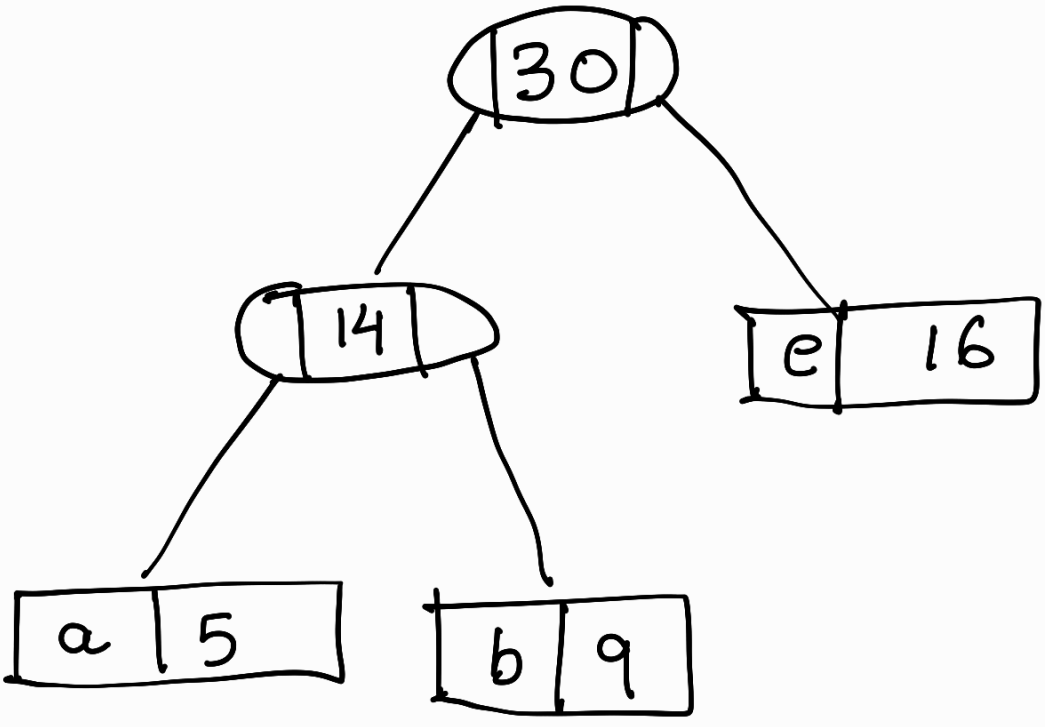
Step3 Extract two minimum frequency nodes from the heap. Add a new internal node with frequency $12 + 13 = 25$



Now, the min heap consists of 4 nodes

Characters	Frequency
a, b	14
c, d	25
e	16
f	45

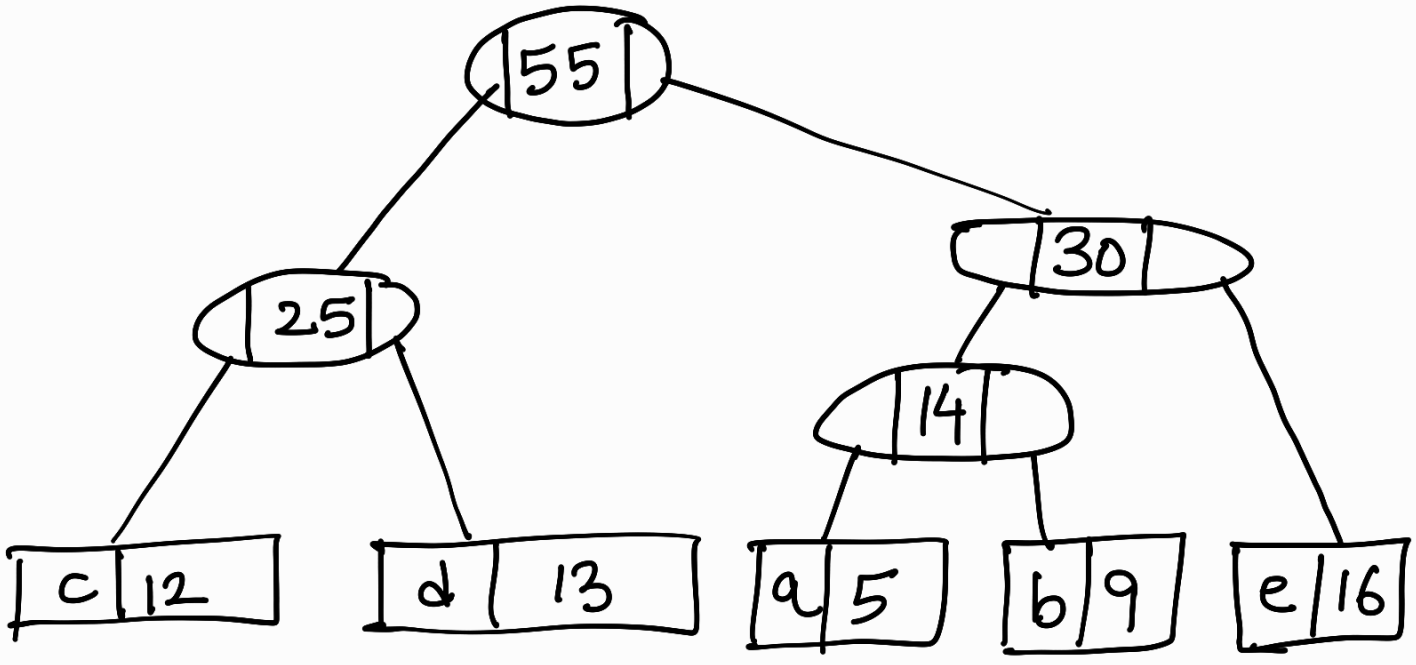
Step 4: Extract two minimum frequency nodes. Add a new Internal node with frequency 14 + 16 = 30



Now, the min heap contains 3 nodes.

Character	Frequency
c,d	25
a,b,e	30
f	45

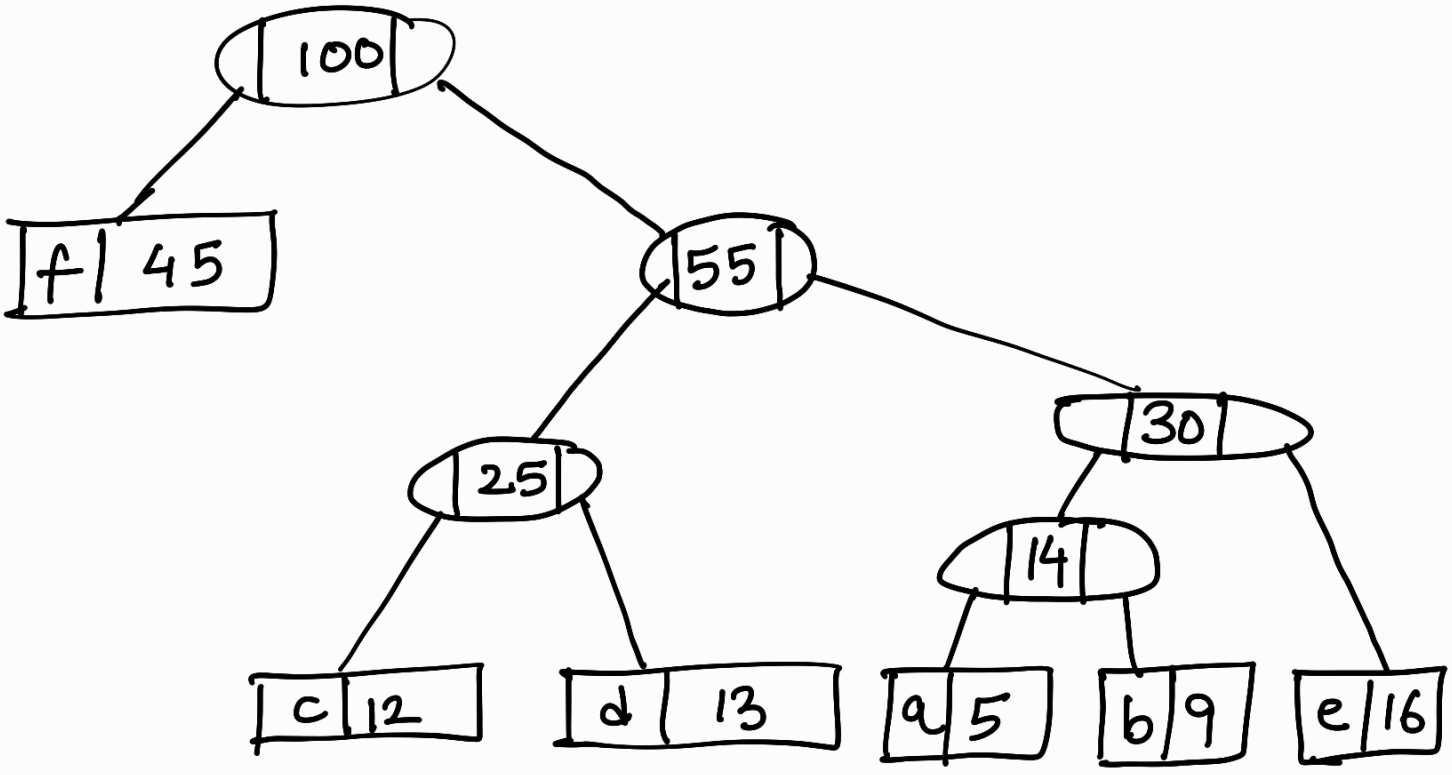
Step 5: Extract two minimum frequency nodes. Add a new internal node with frequency 25+30 = 55



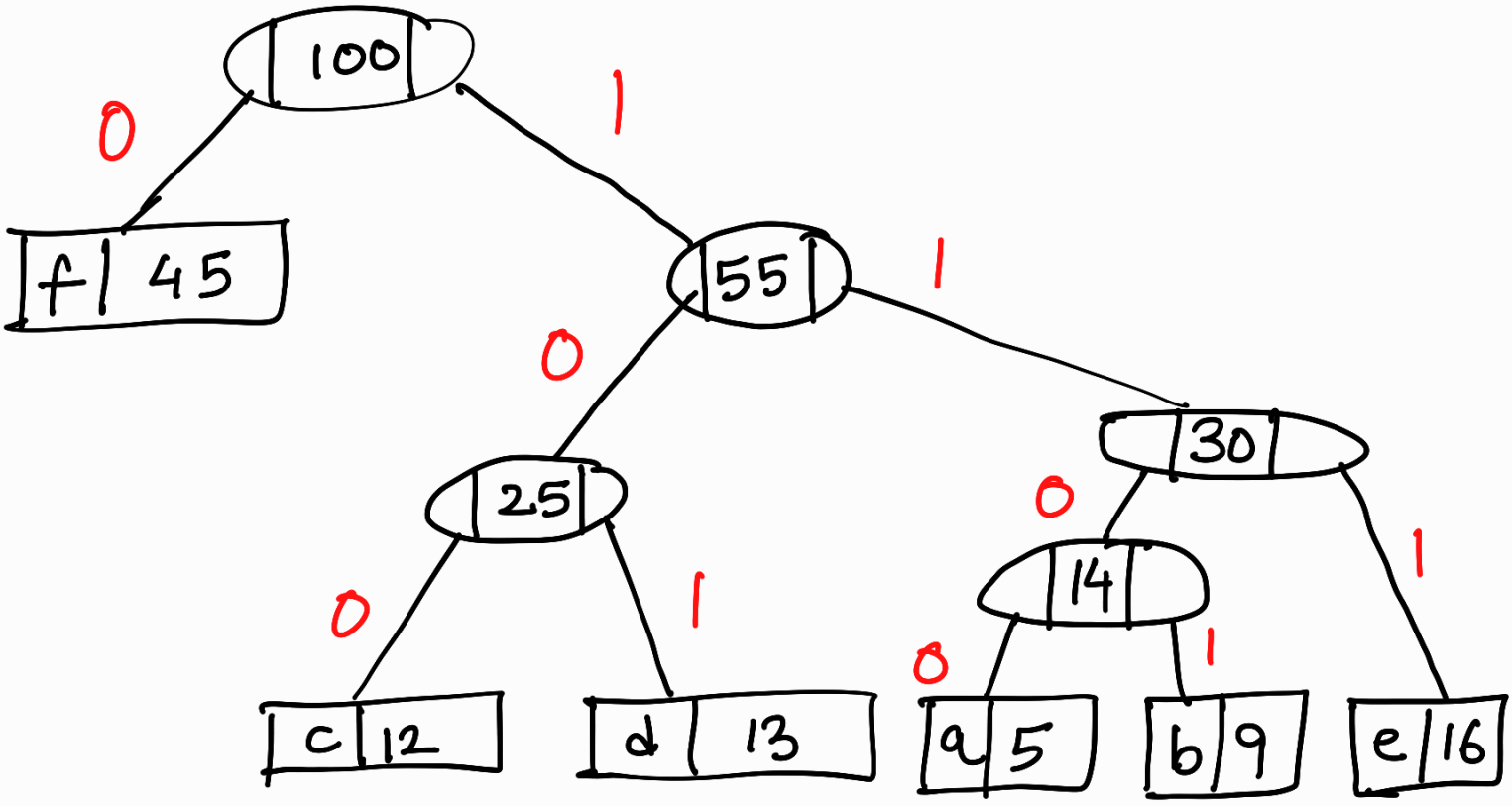
Characters	frequency
f	45
a, b, c, d, e	55

Step6: Extract two minimum frequency nodes. Add a new internal node with frequency

45 + 55 = 100



Now Code the nodes



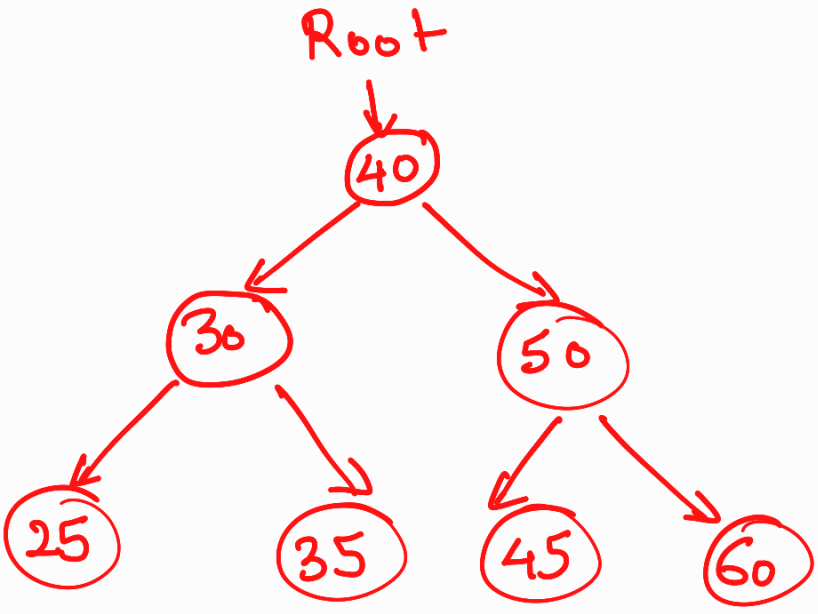
Characters	Code Word
a	1 1 0 0
b	1 1 0 1
c	1 0 0
d	1 0 1
e	1 1 1
f	0

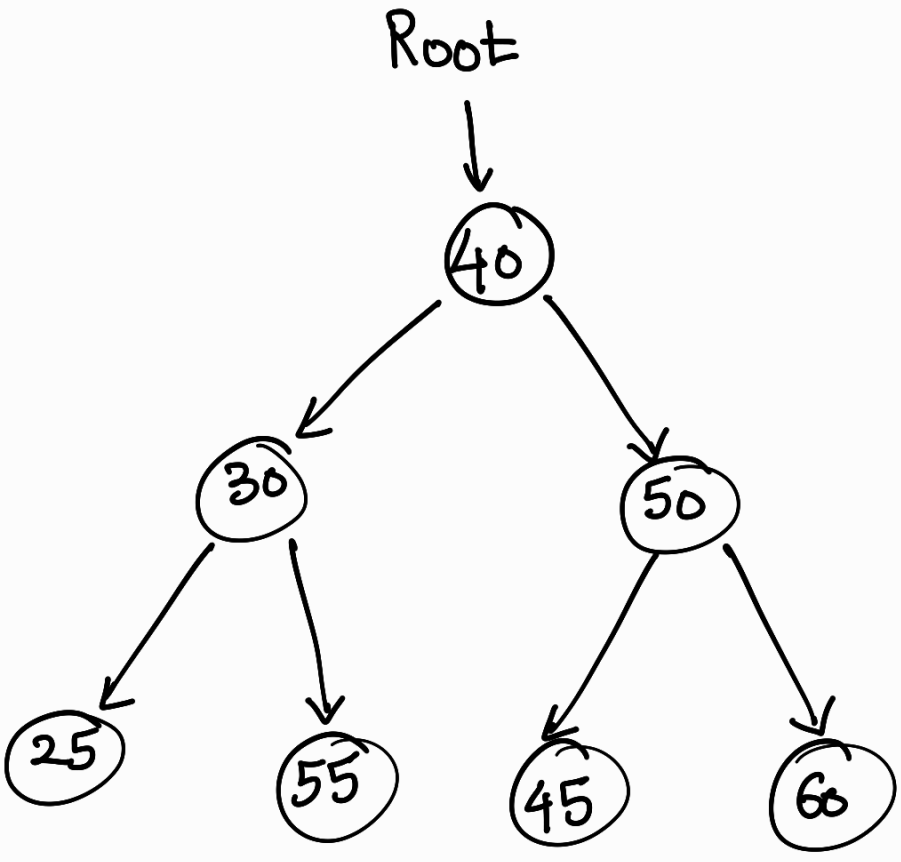
Binary Search Tree

A Binary search tree follows some order to arrange the elements.

In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node.

This rule is applied recursively to the left & right subtree of the root.





Is it a Binary Search Tree ??

Advantages :

* Searching an element in the Binary search tree is easy as we always have a hint that which subtree has the desired element
* As compared to array & linked lists, insertion & deletion operations are faster.

Example

Suppose the data elements are :-

45, 15, 79, 90, 10, 55, 12, 20, 50

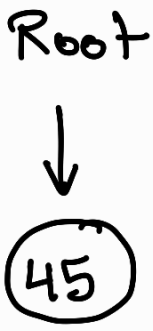
Sol:

1. First we have to insert 45 into the tree as the root of the tree.

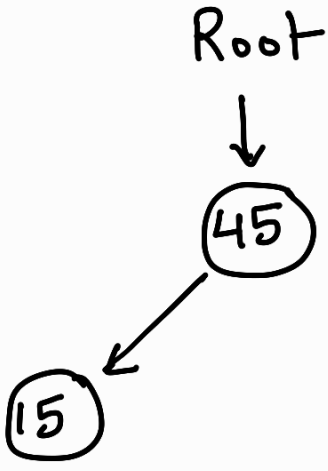
2. Then, read the next element; if its smaller than the root node, then insert it as the root of the left subtree, and move to the next element.

3. Otherwise, if the element is larger than the root node, then insert it as the root of the right subtree.

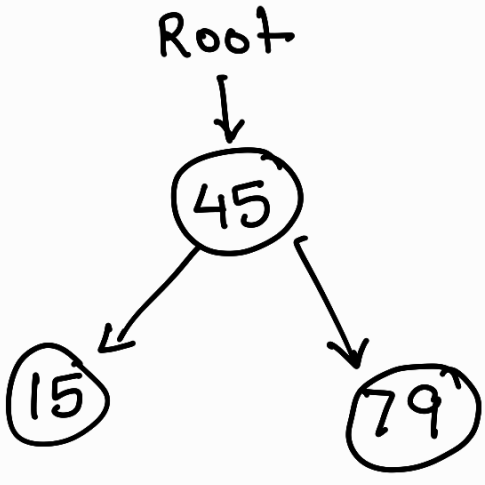
Step 1 . Insert 45.



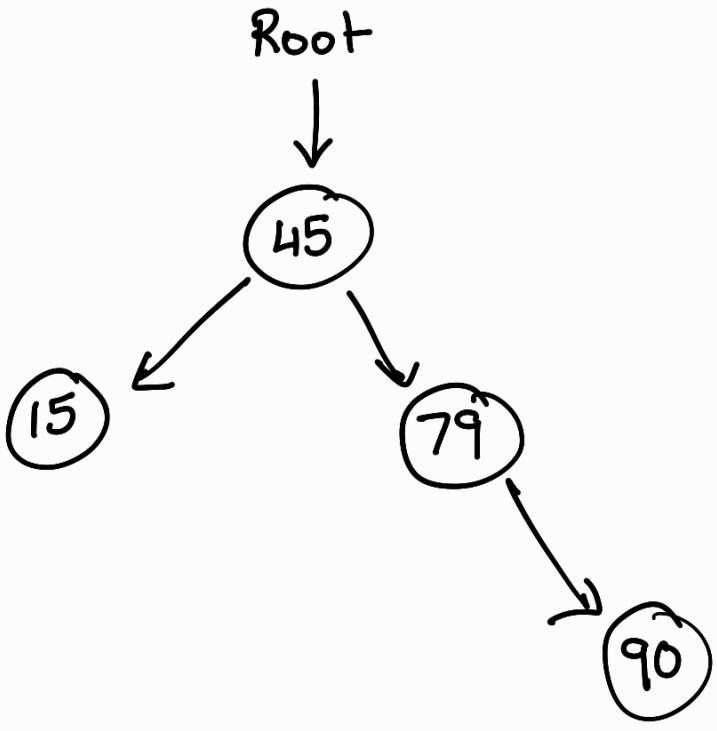
Step 2 Insert 15



Step3 Insert 79

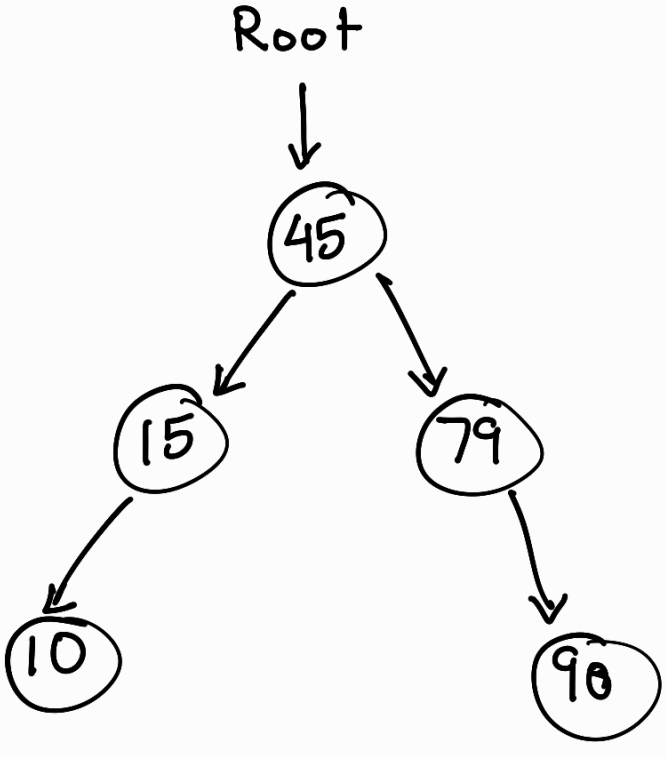


Step 4



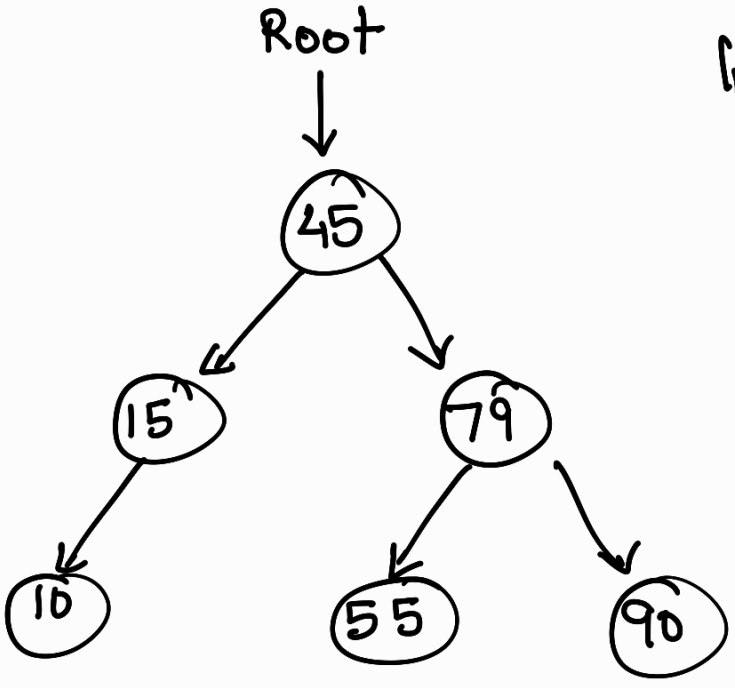
Insert 90

Step 5



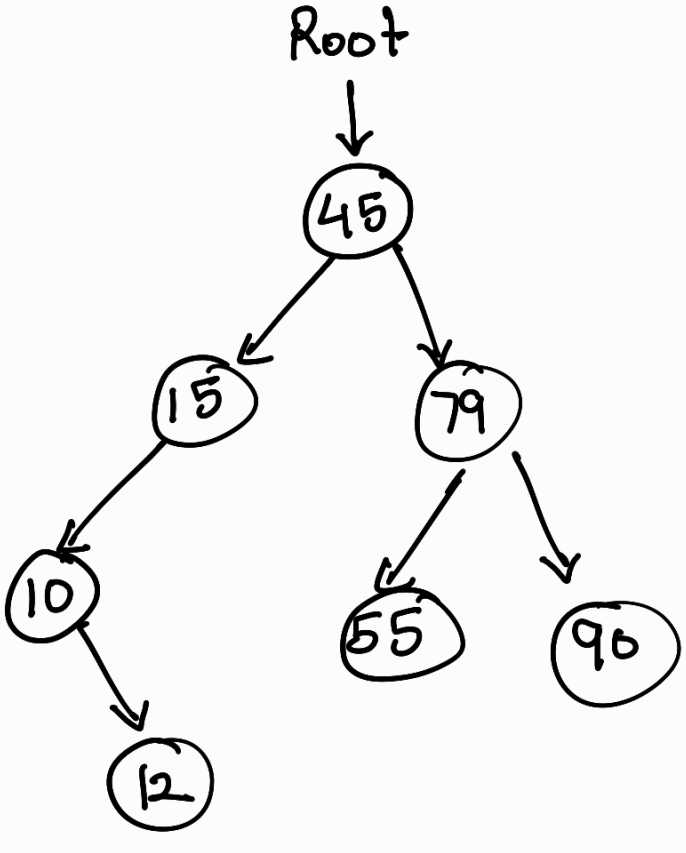
Insert 10

Step 6



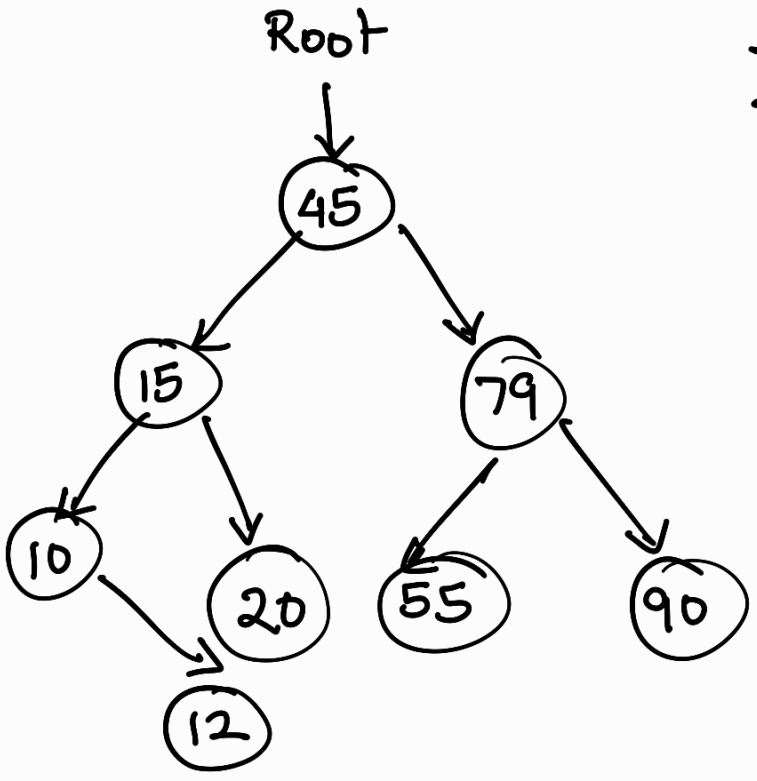
Insert 55

Step 7



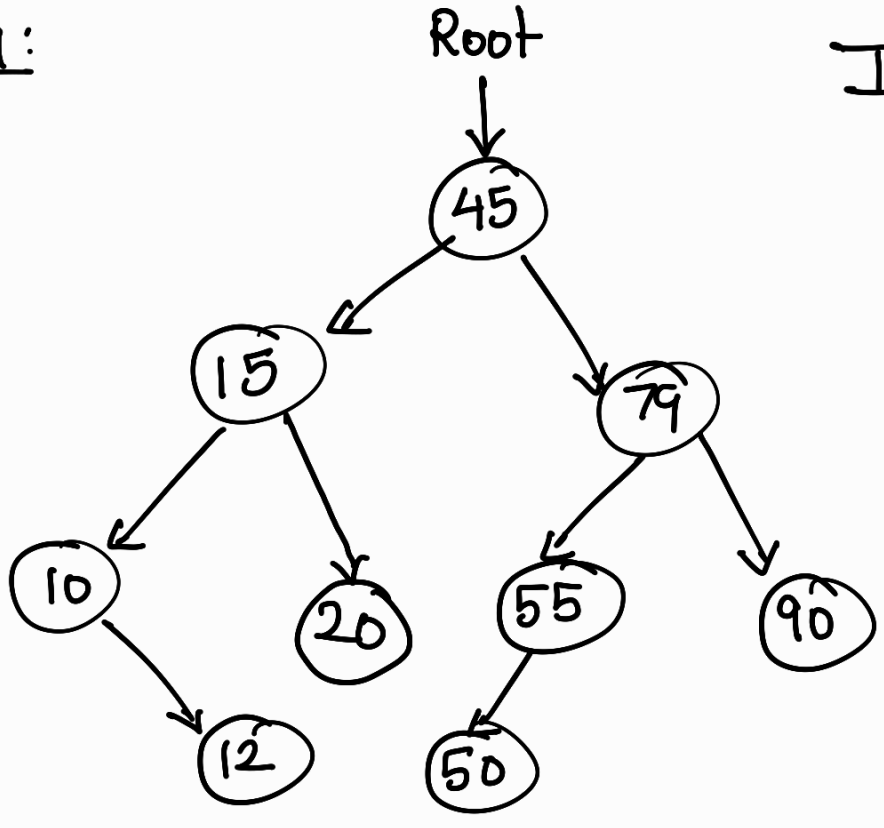
Insert 12

Step 8:



Insert 20

Step 9:



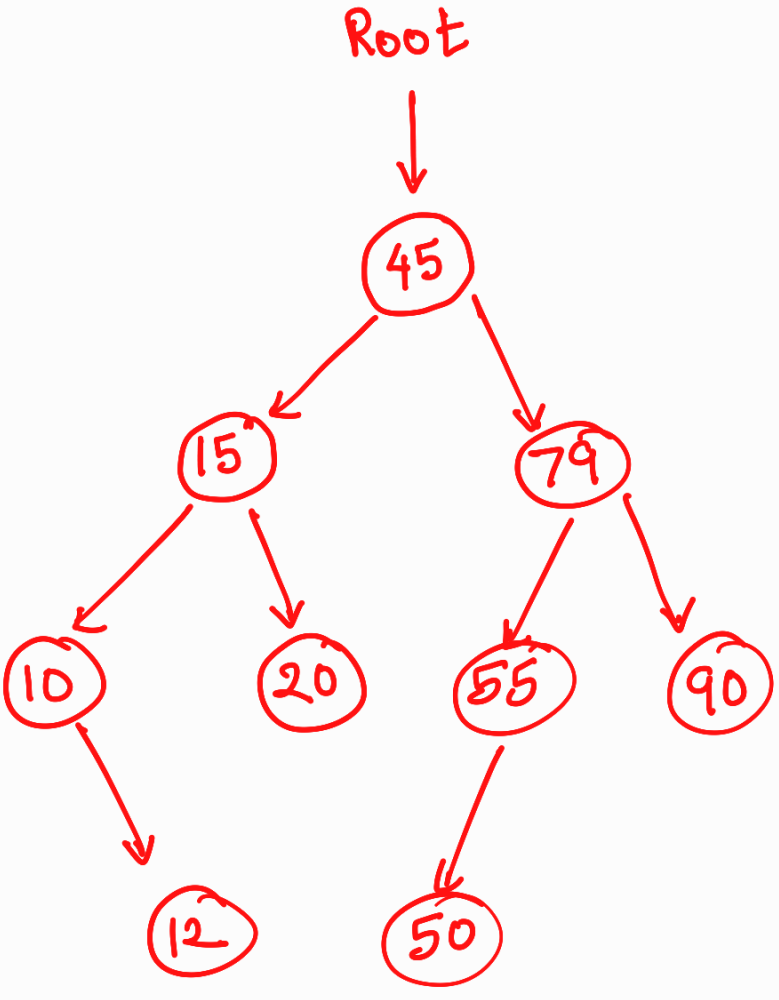
Insert 50

Searching in a Binary Search Tree

1. First, compare the element to be searched with the root element of the tree.
2. If root is matched with the target element, then return the node's location.
3. If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.
4. If it is larger than the root element, then move to the right subtree.

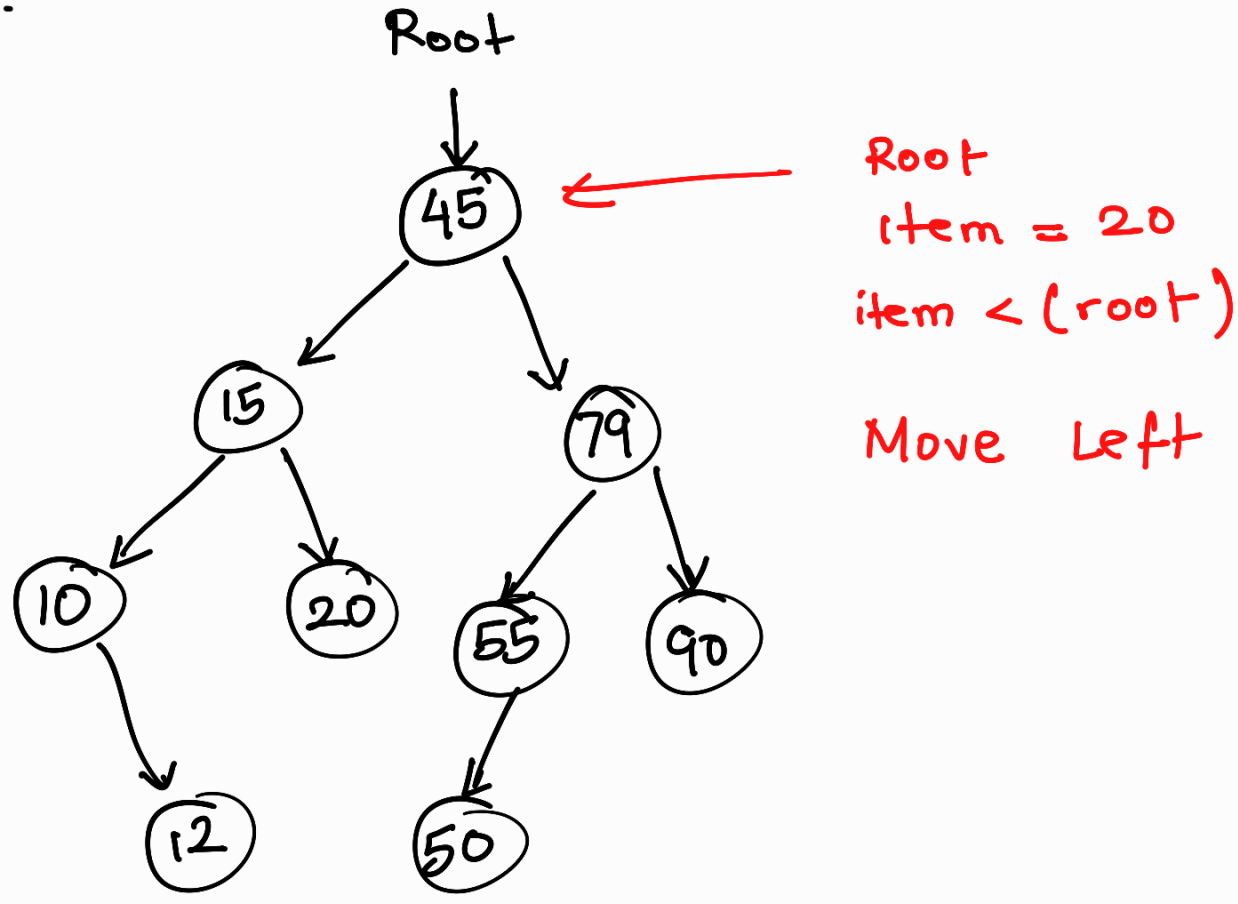
5. Repeat the above procedure recursively until the match is found.

6. If the element is not found or not present in the tree, then return NULL.

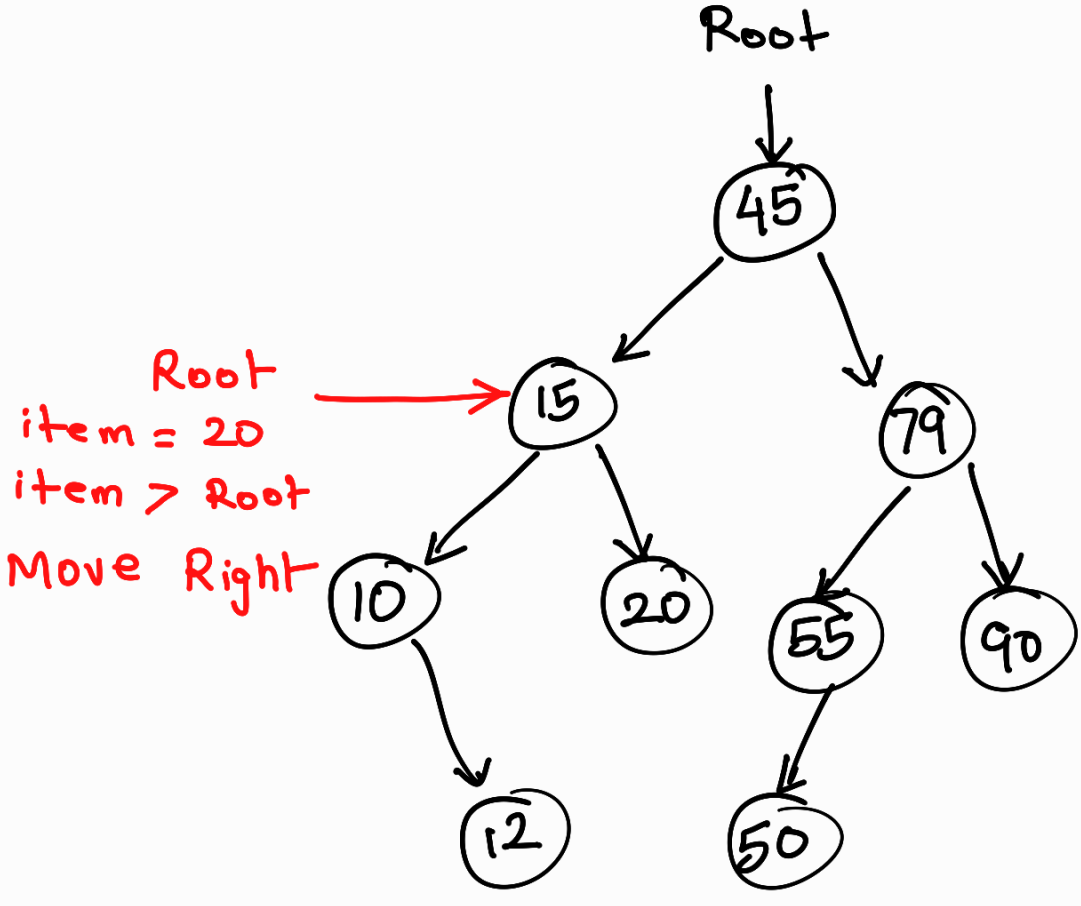


Find 20

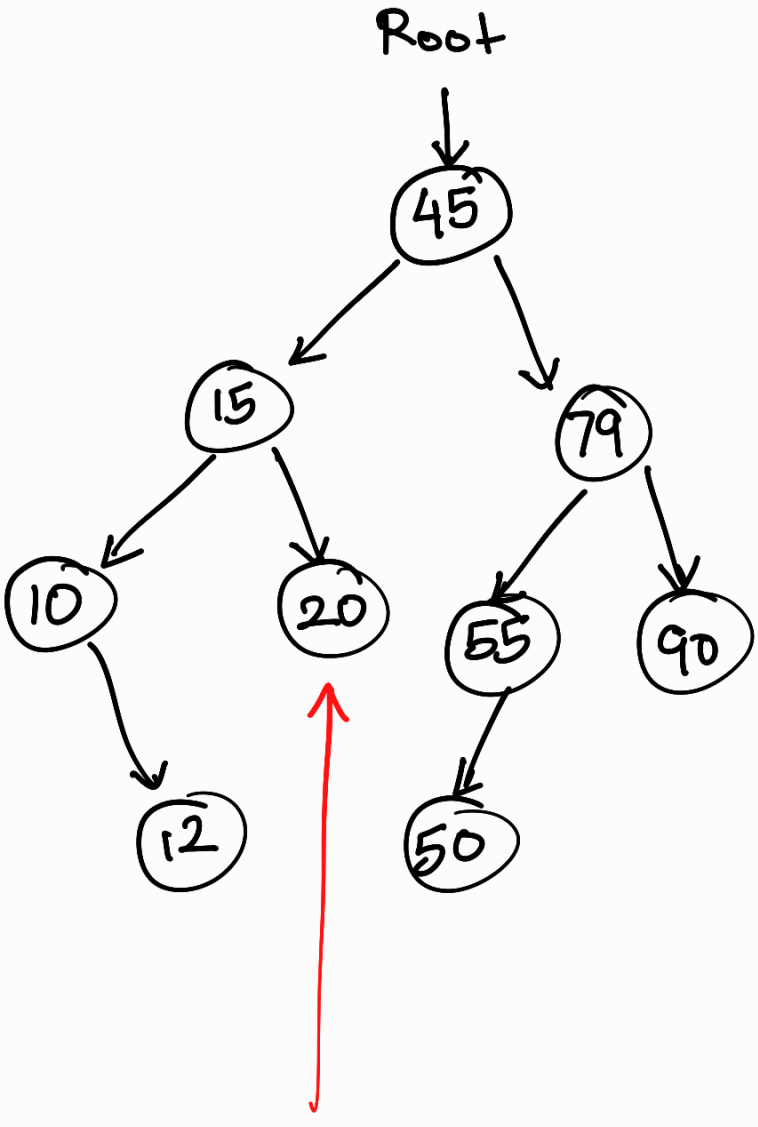
Step 1:


Root
45
15
79
10
20
55
90
12
50
Root
item = 20
item < (root)
Move Left

Step 2:


Root
45
Root
item = 20
item > Root
Move Right
15
79
10
20
55
90
12
50

Step 3 :



Root

item = Root

Return Root

Match Found