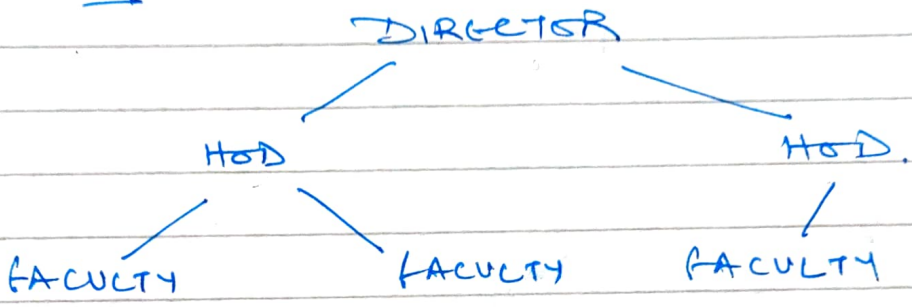
TREES

→ Non Linear Data structure
→ Used to represent hierarchical relationship between elements.

Eg



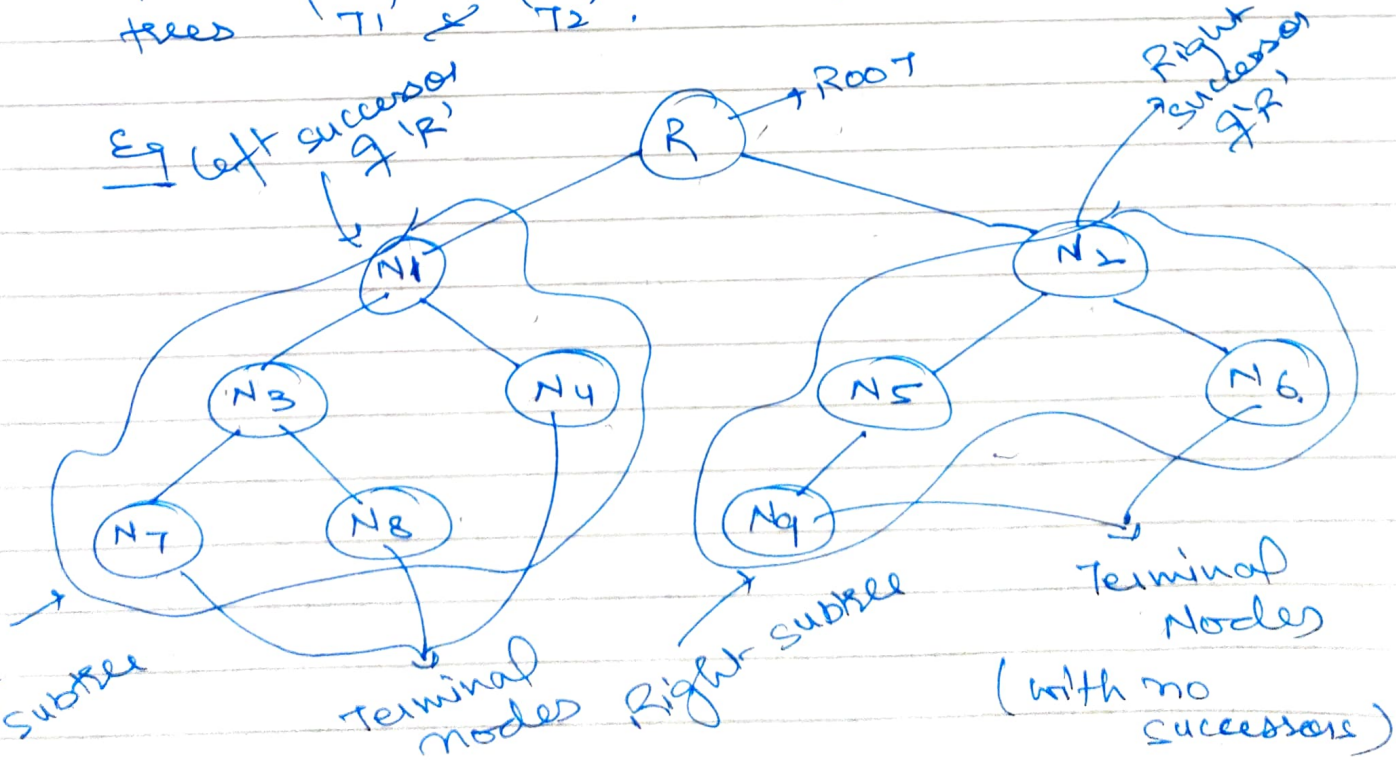
BINARY TREE

DEFINITION

Binary tree 'T' is defined as set of elements called NODES. Such that:

1) 'T' is empty (NULL Tree / EMPTY Tree)
2) 'T' contains distinguished node 'R' called ROOT of 'T' and remaining nodes of 'T' form an ordered pair of disjoint binary trees 'T1' & 'T2'.

Eg



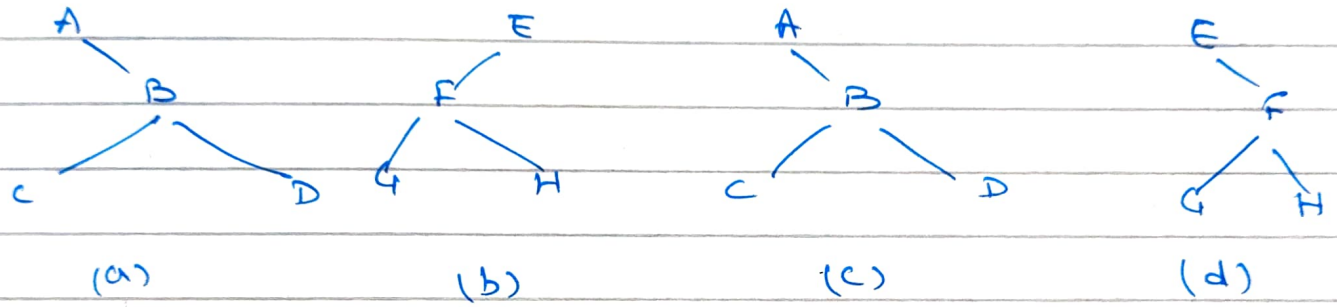
SIMILAR TREES

Two trees are similar if they have same shape or structure Eg: (a) (c) (d)

COPIES TREES

Two trees have same data and are similar

Eg: (a) (c)

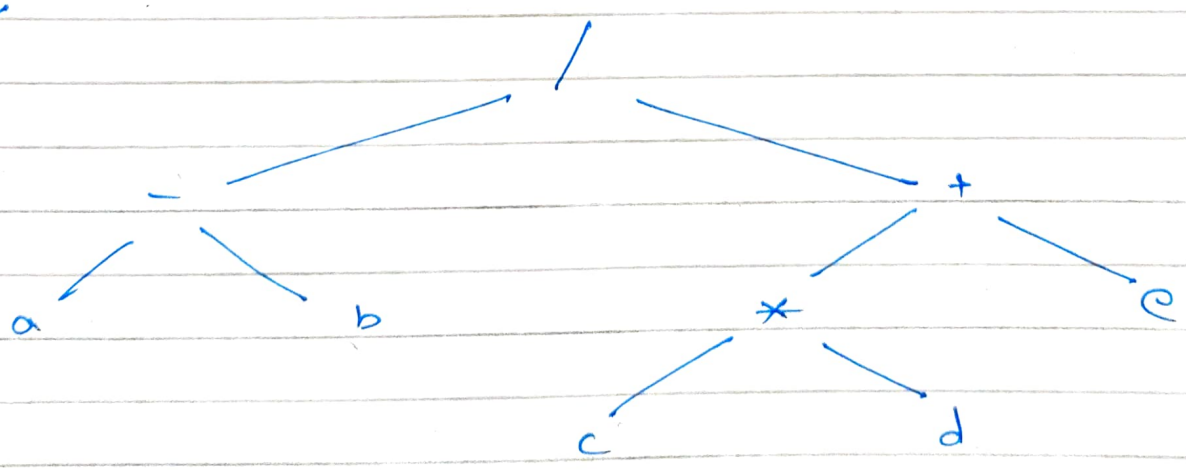


(a) (b) (c) (d)

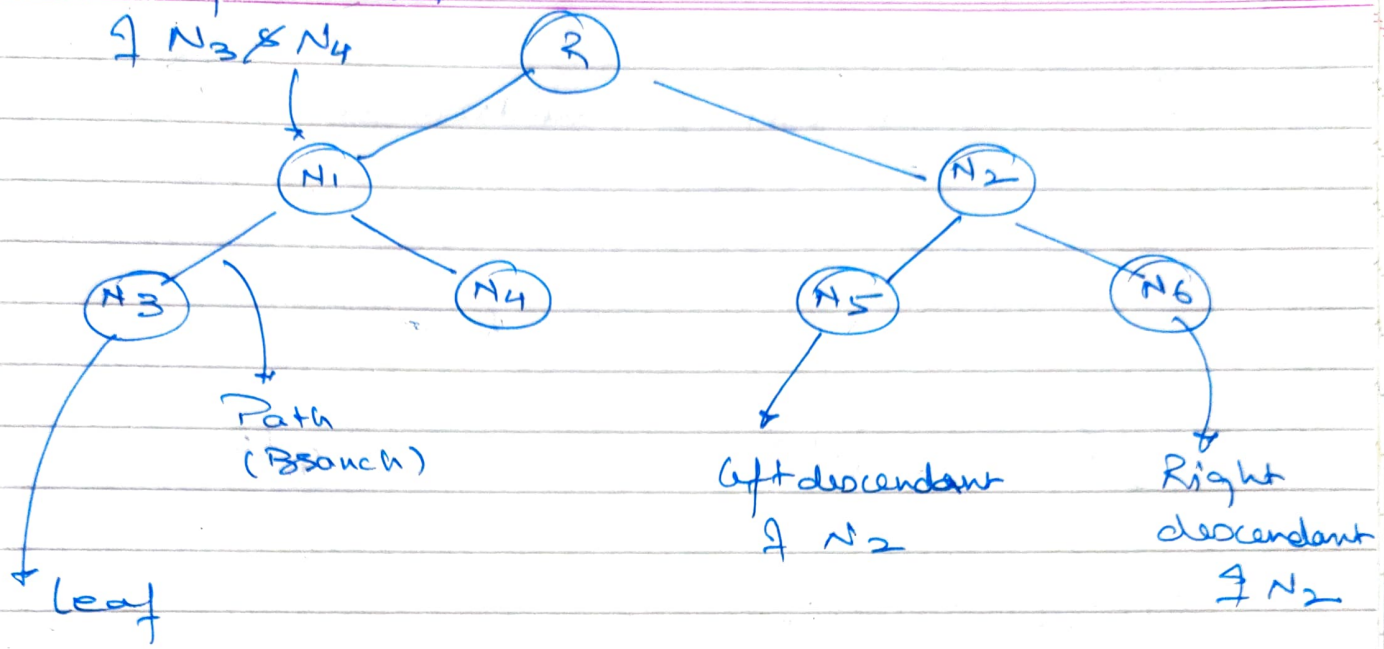
Ques Represent using binary trees.

E = (a-b) / ((c*d) +e)

Ans

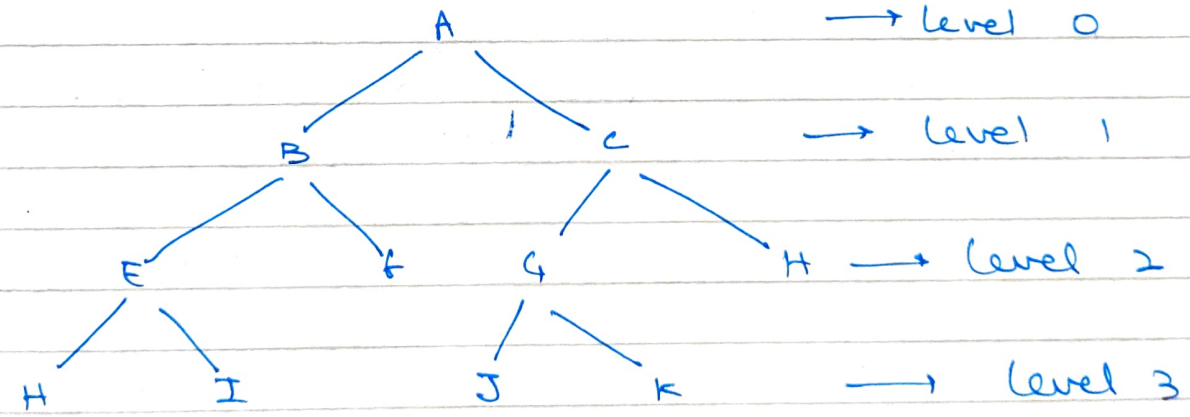


Parent | Predecessor



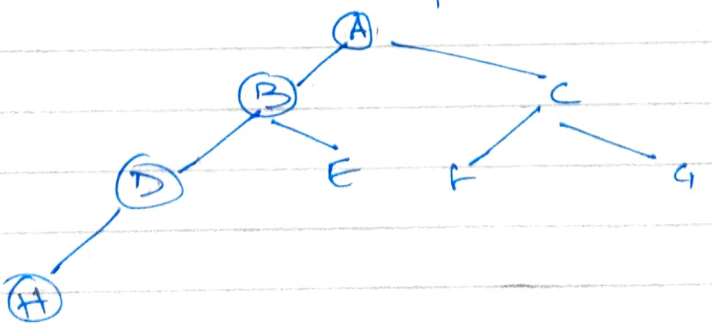
LEVEL NUMBER

The number of parent nodes a tree node has.
Root 'R' has level number 0.



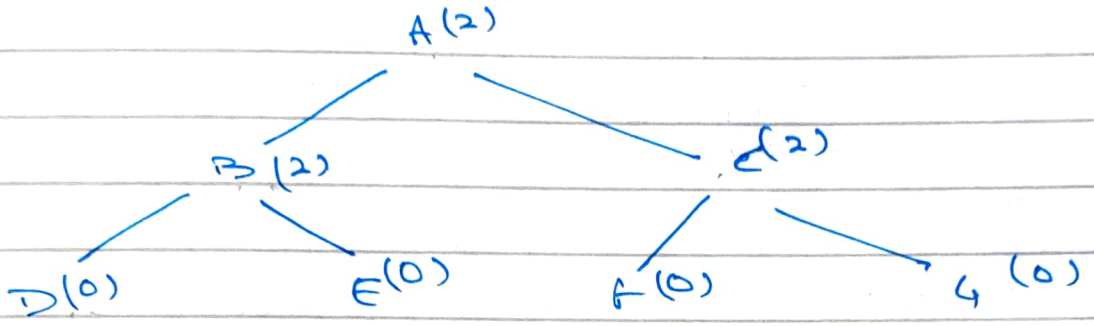
Depth of Tree

→ Maximum number of nodes in branch of 'T'



Degree of a node

→ it is number of children a node has.



Degree of a tree.

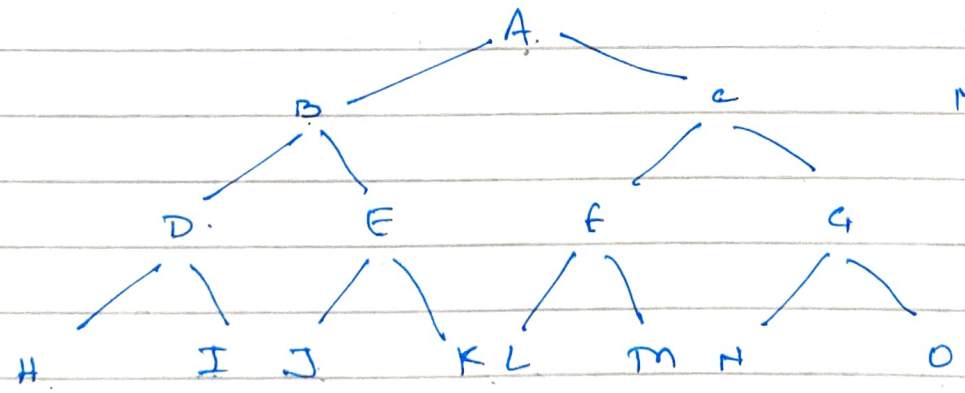
→ The maximum degree of a node.

→ In the above example maximum degree is '2' hence degree of above tree is '2'.

Complete Trees

A tree in which each node is at some distance from the root.

Let height be h, then total number of nodes are $2^{h+1}-1$.



Height → 3

Nodes → $2^{3+1}-1$

$= 16-1 = 15$

Height of a Tree

it is the number of links from the root to deepest leaf.

Traversing Binary Tree

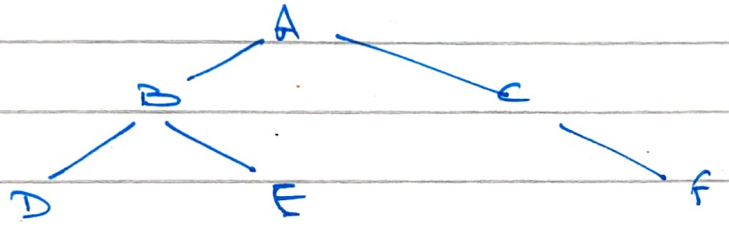
Preorder → R_o L R (N L R)
↑ Node

→ Process Root 'R'
→ Process LEFT of 'R'
→ Process RIGHT of 'R'

Inorder → L R_o R (LNR)

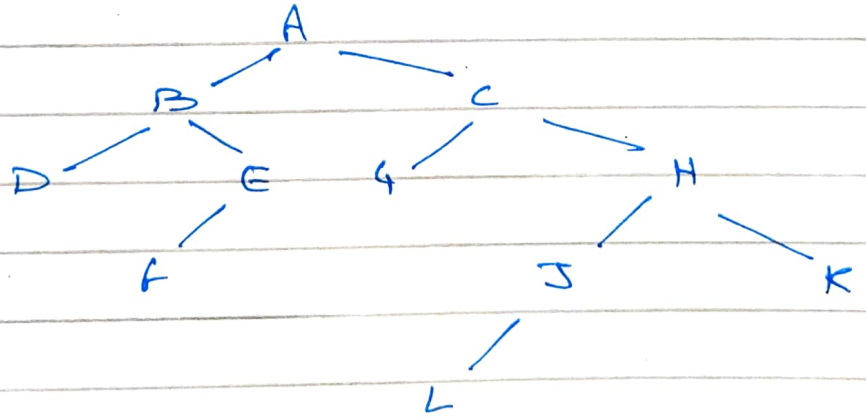
Postorder → L R R_o (LRN)

Eg:



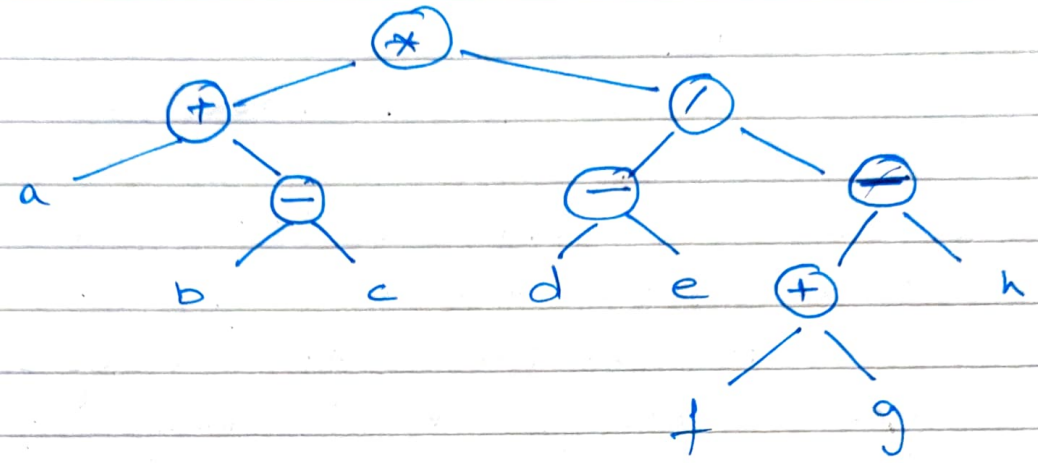
Preorder → A B D E C F
Postorder → D E B F C A
Inorder → D B E A C F

Eg



Inorder – D B F E A G C L J H K
Postorder – D F E B G L J K H C A
Preorder – A B D E F C G H J L K

Eg



Preorder

* + a - b c / - d e - + f g h

Postorder

a b c - + d e - f g + h - / *

Ques. A binary tree T has 9 nodes. The inorder and preorder traversals of 'T' yield the following sequences of nodes:

INORDER : E A C K F H D B G

PREORDER : F A E K C D H G B

Draw the tree.

Ans → First node of preorder is the root 'F'

→ To find left childs of 'f' use inorder, all the entries to the left of 'f' will be nodes of left subtree.

E, A, C, K

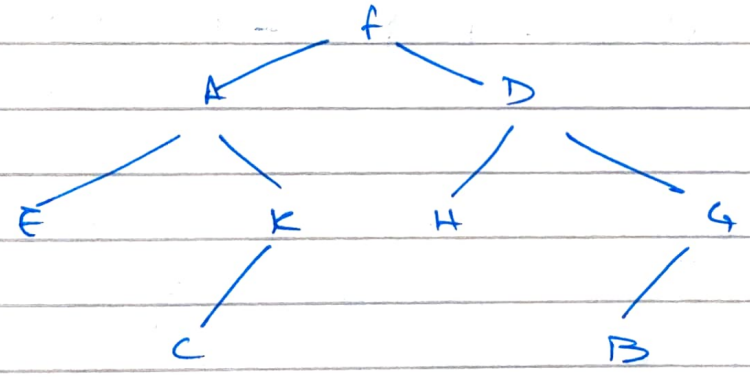
Now left child of 'F' is used by choosing entry next of 'F' in preorder ie 'A'

→ The right subtree of 'f' has H, D, B, G in the preorder while moving towards right remove all the entries in the left subtree E, K, C in preorder and then 'D' comes which is right

child of 'F'

→ Now taking 'A' as root. Read all the entries to the left of 'A' in inorder ie 'E', since there is only one entry so 'E' is on the left.

→ While moving towards right in preorder checkout all the left entries of 'A' and after checking out the first 'non left' (E) entry, node 'K' is the right node of 'A'

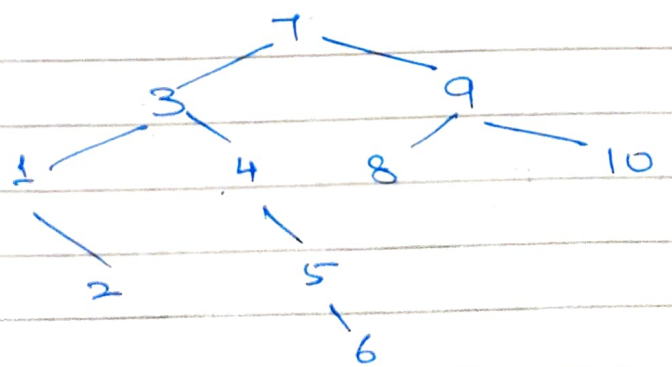


Binary Search Tree.

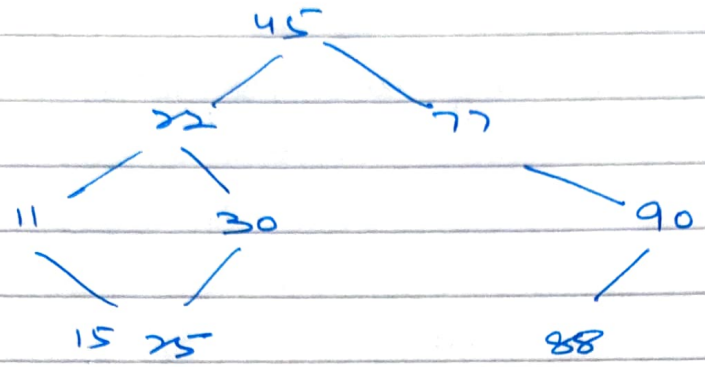
The tree which satisfies the following rule:

→ The value of key in the left child or left subtree is less than that of value of the root.

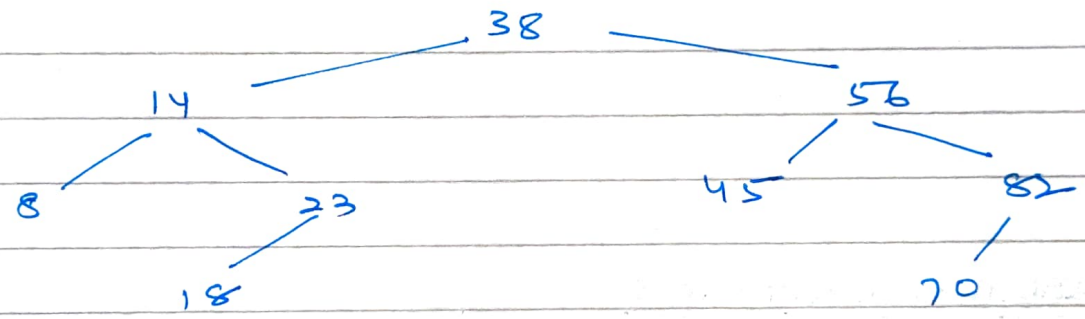
→ The value of child in the right / right subtree is more than the value of root.



Eg



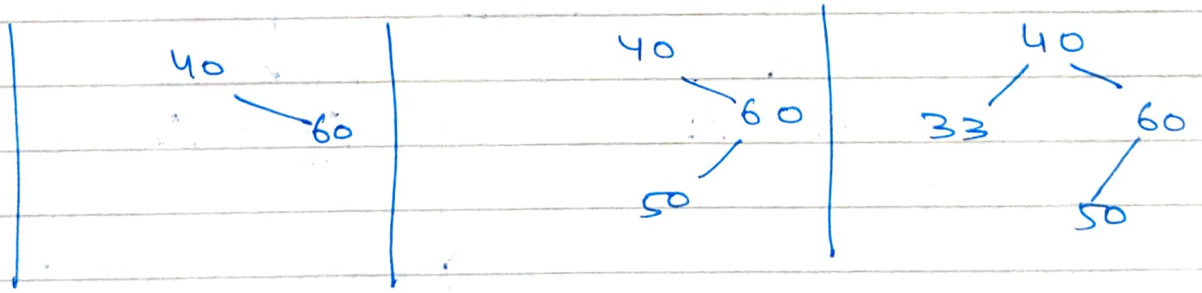
Eg

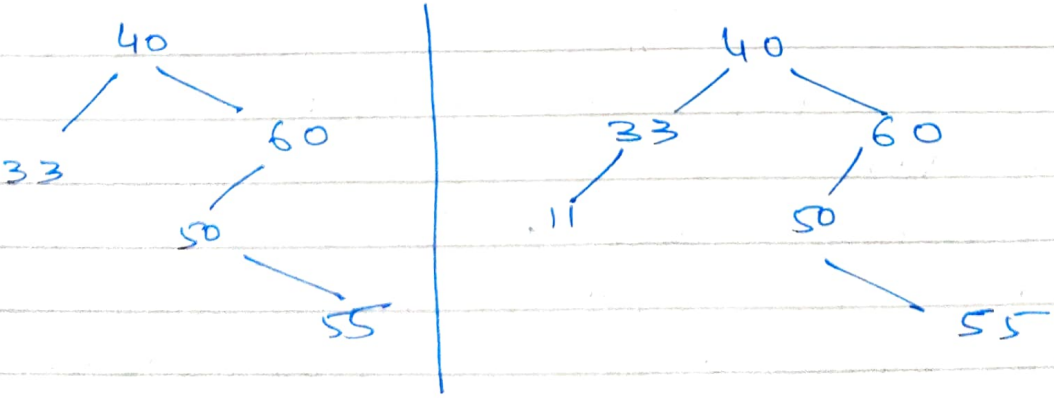


Ques Create a binary search tree:

40, 60, 50, 33, 55, 11

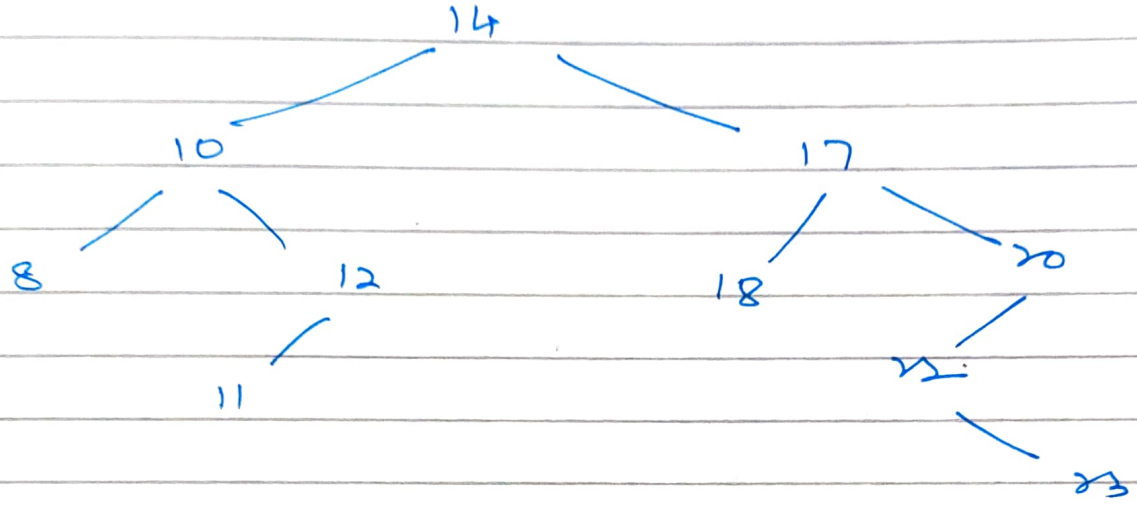
Ans





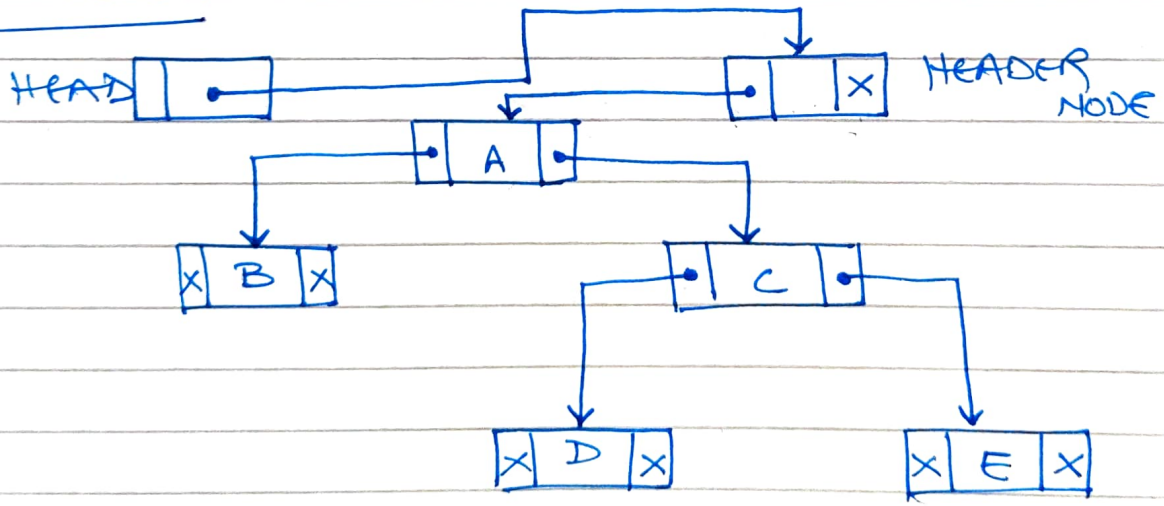
Ques Create a tree.

14, 10, 17, 12, 10, 11, 20, 12, 18, 25, 20, 8, 22, 11, 23



THREADED BINARY TREE

HEADER NODES.



→ Suppose a binary tree 'T' is implemented using linked list implementation

→ An extra, special node called header node is added to the begning of 'T'.

→ When this node is added the tree pointer is called HEAD (not ROOT), it will point to header node, and left pointer of header node will point

to the root of 'T'.

→ Suppose if 'T' is empty, then 'T' will still contain header node but left of header node will be NULL.

LEFT [HEAD] = NULL

→ Another way to indicate empty tree is

LEFT [HEAD] = HEAD.

THREADED BINARY TREES

THREADS: INORDER THREADING

→ In linked representation of trees, appox half entries of left and right will contain NULL elements.

→ This space may be efficiently used by replacing NULL pointer entries by some other type of information.

→ We replace certain NULL entries by special pointers which points to nodes higher than the tree.

→ These special pointers are called THREADS.

→ These binary trees are called THREADED TREES.

→ The THREADS in the trees must be distinguished from ordinary pointers.

→ In memory A 'tag' field is used to distinguish (threads may have 'tag' as negative number)

ONE-WAY INORDER THREADING (only Right pointers)

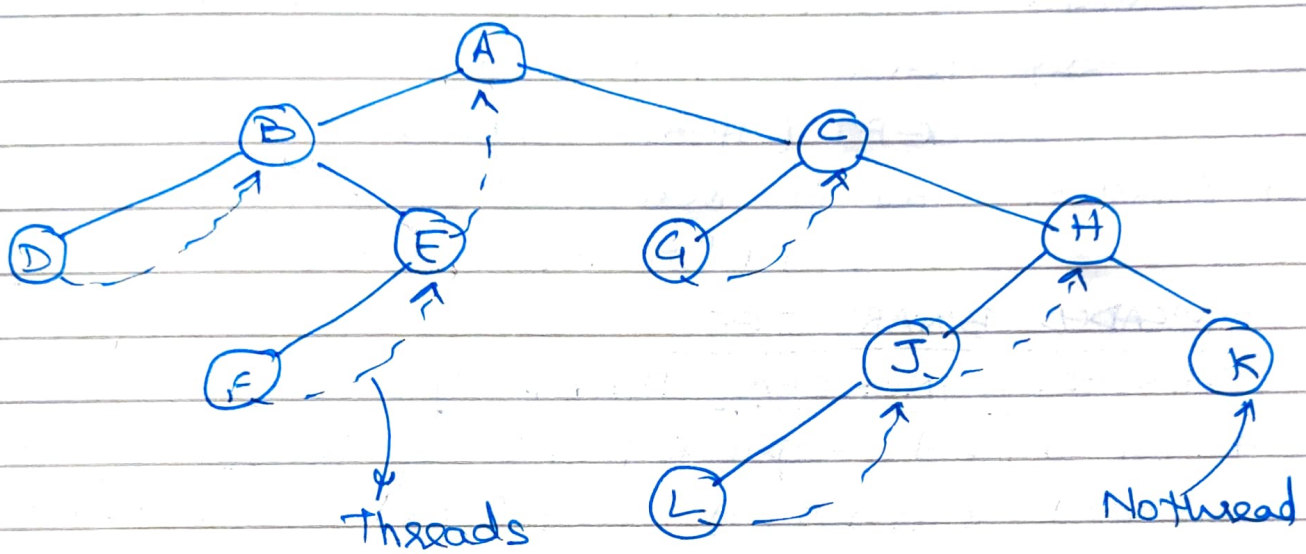
The diagram of TREE is given on next page.

INORDER of TREE (LNR)

D B F E A G C L J H K

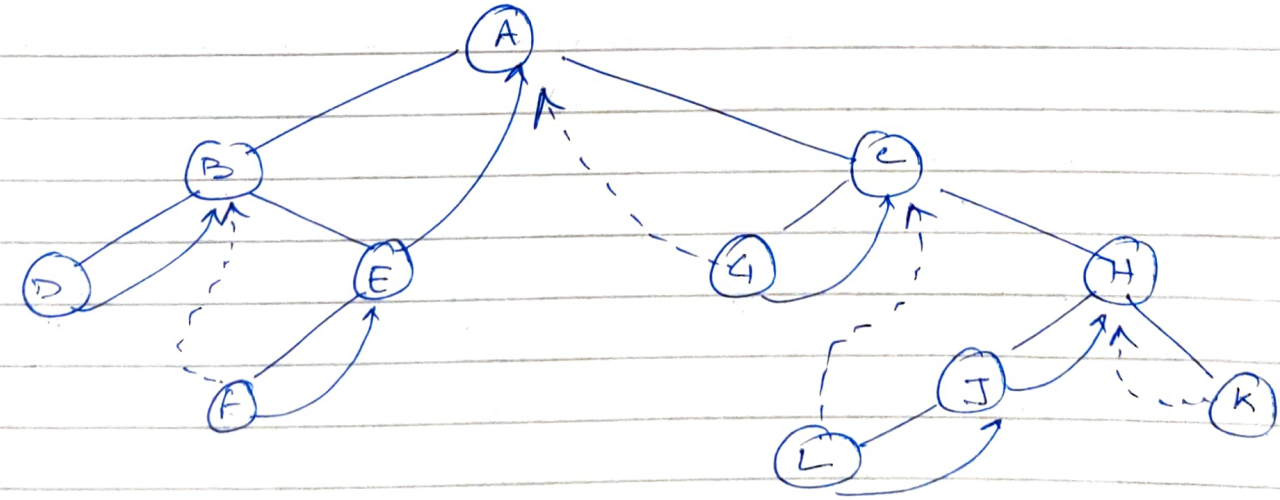
→ on observing INORDER traversal since 'A' is

accessed after 'E' so there is a thread from 'E' to 'A'



→ Also in one way threading every right pointer has been replaced by a thread except for node 'K' which is last node.

TWO WAY INORDER THREADING



INORDER TRAVERSAL (LNR)

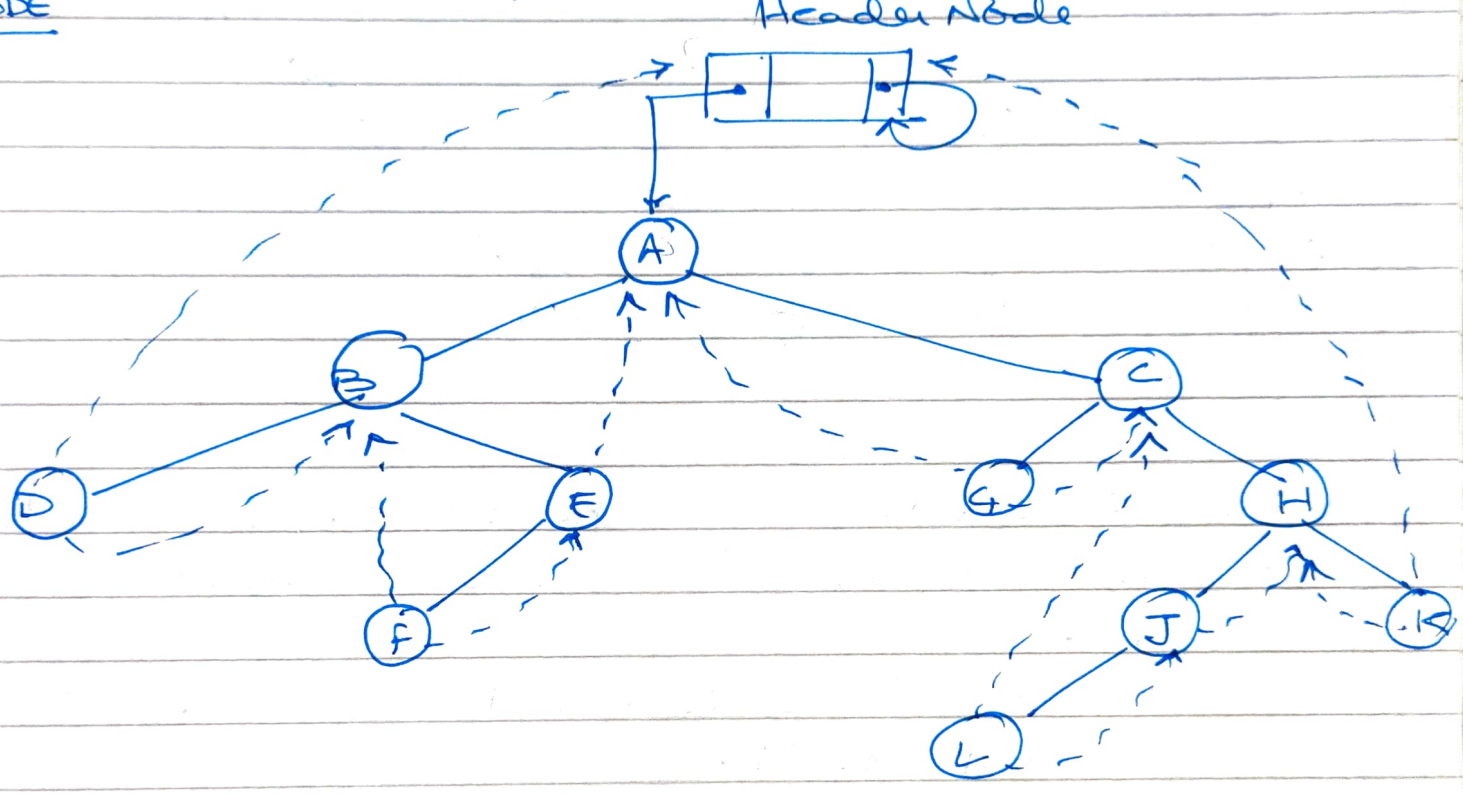
D B F E A G C L J H K

→ In two way inorder we have left thread from node 'L' to 'C'. Since 'L' is accessed after 'C'.

→ Here every NULL left pointer has been replaced

by an thread except for node D, which is the first node for inorder traversal.

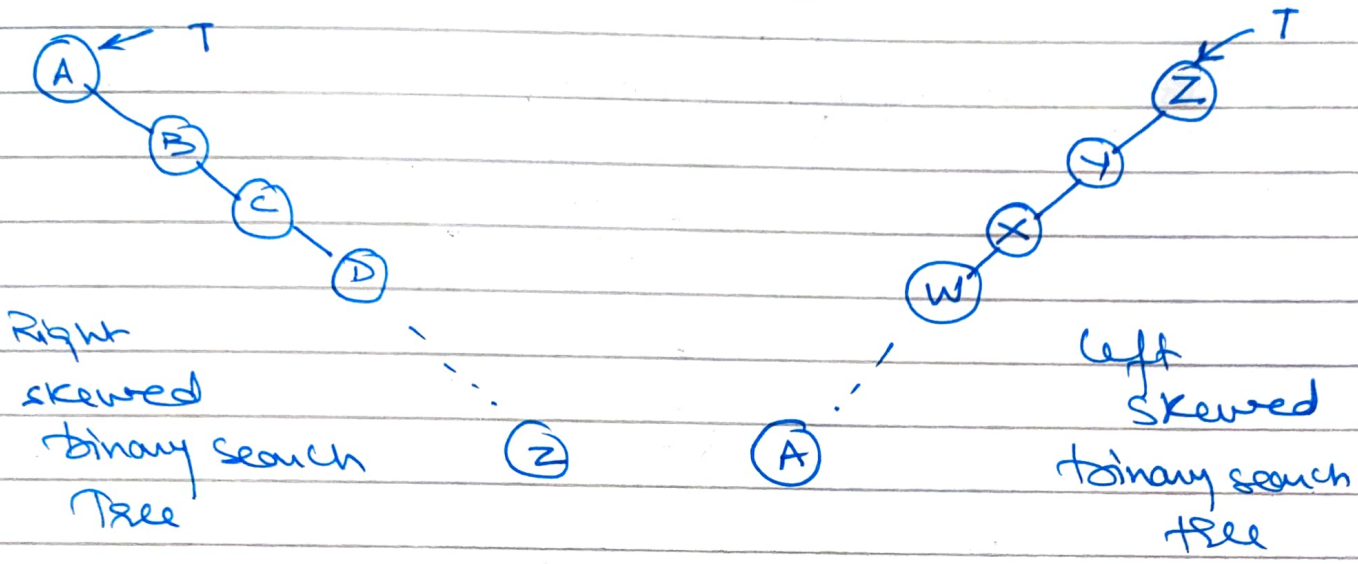
TWO WAY INORDER THREADING WITH HEADER NODE



Here left thread of 'D' and right thread of 'K' point towards header node.

AVL SEARCH TREES

(ADELSON - VELSKI - LANDIS)



→ The disadvantage of skewed binary search tree is that worst case time complexity is $O(n)$.

→ Therefore there is a need to obtain a tree for balanced height.

DEFINITION OF AVL

→ An empty binary tree is an AVL TREE.

→ for an non-empty binary tree to be AVL, it is required

$$\text{iff} \quad \left| h(T^L) - h(T^R) \right| <= 1$$

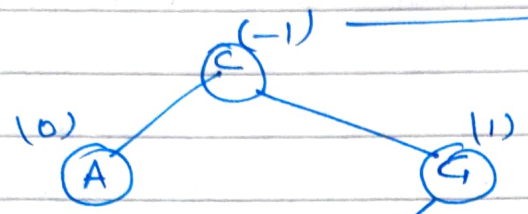
- $h(T^L)$ → height of left subtree
- $h(T^R)$ → height of right subtree
- Balance factor (0, 1, -1)

Height Left = L
Height Right = R

Date :
Page No.

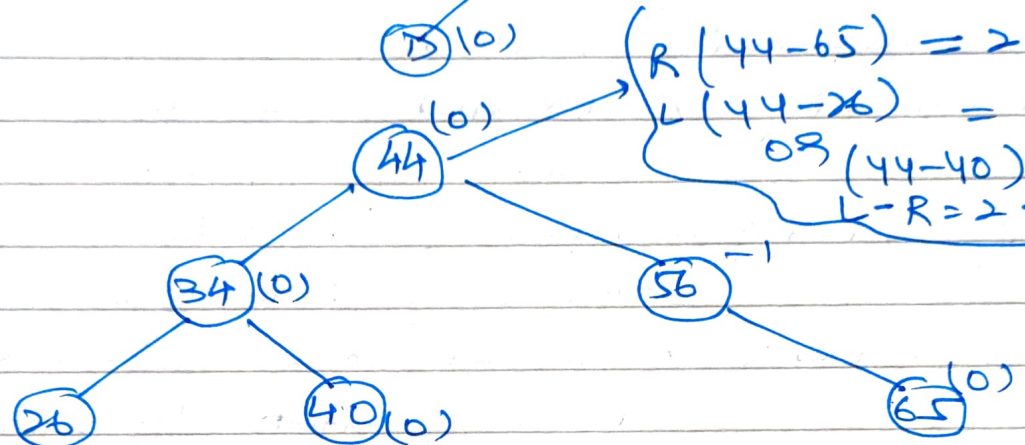
Representation of a AVL Tree

(a)



R(C to D) = 2
L(C to A) = 1
L - R = 1 - 2 = -1

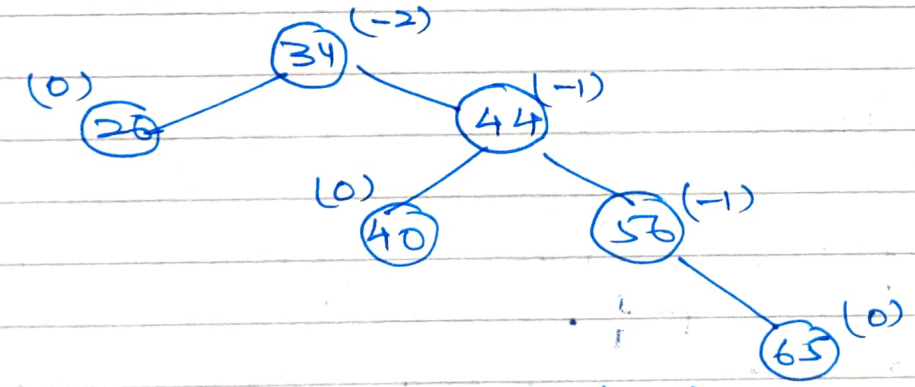
(b)



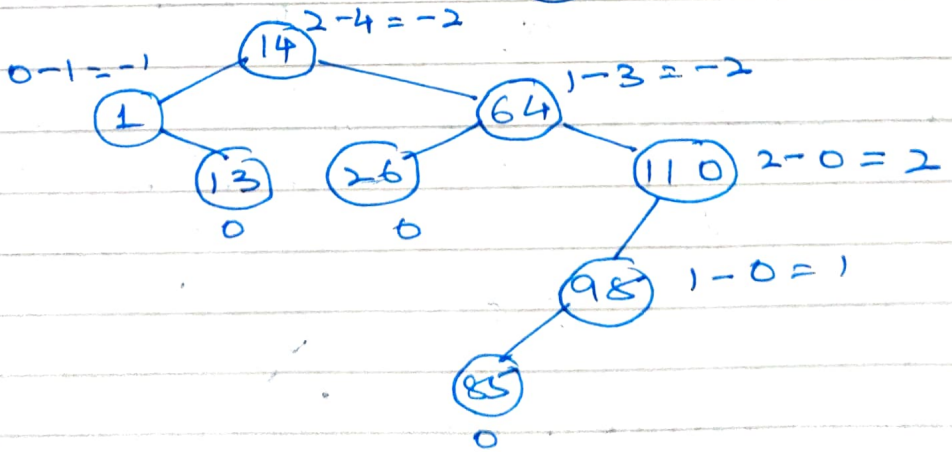
R(44-65) = 2
L(44-26) = 2
OR (44-40)
L - R = 2 - 2 = 0

NON AVL EXAMPLE

(a)



(b)



INSERTION IN AVL SEARCH TREE

→ Insertion into AVL tree is similar to binary search tree

→ if after insertion, the balance factor of node is affected then we have to adjust this unbalanced tree by using ROTATIONS.

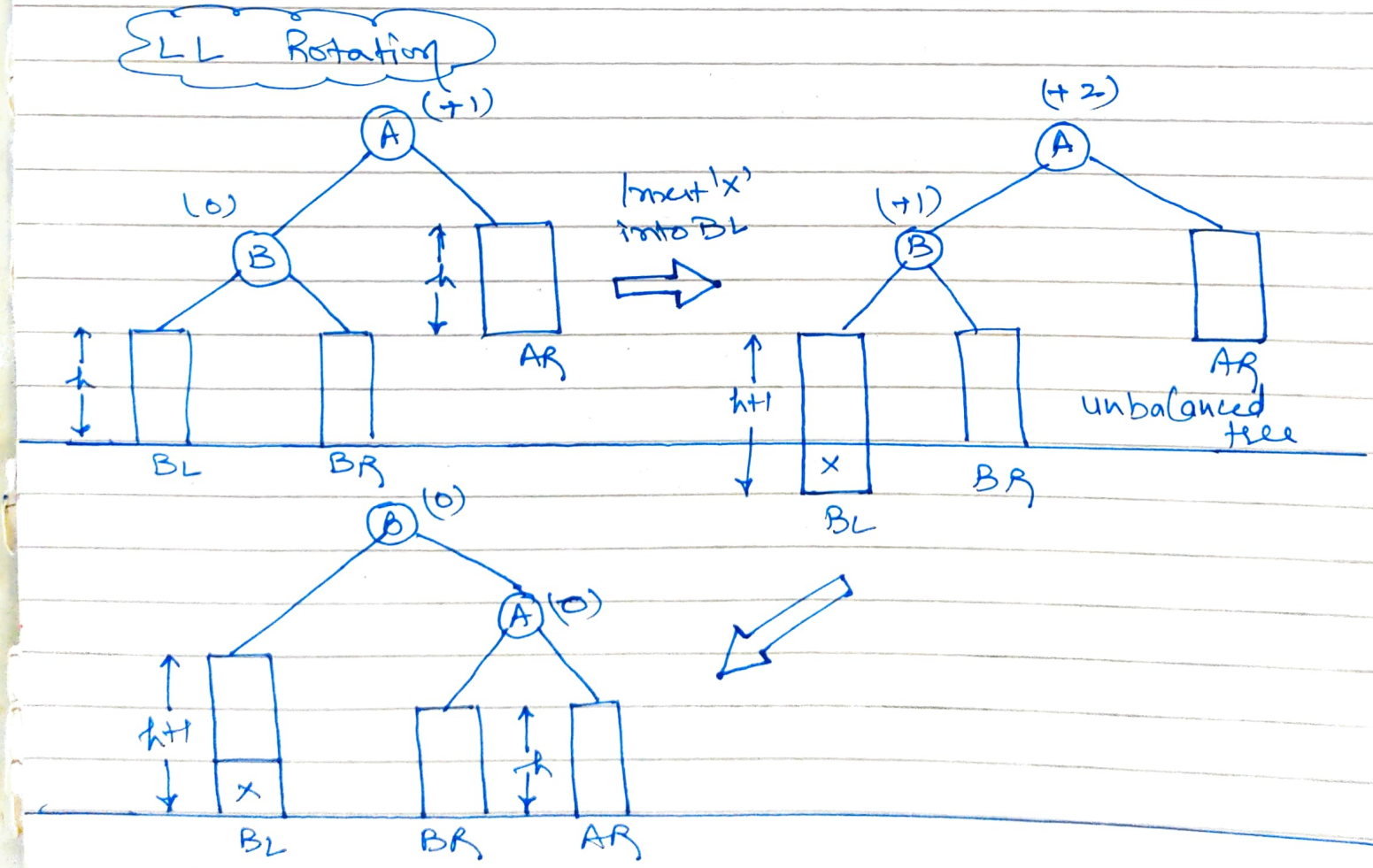
→ The rebalancing rotations are classified as:

LL Rotation → Inserted node is in the left subtree of left subtree of node 'A'

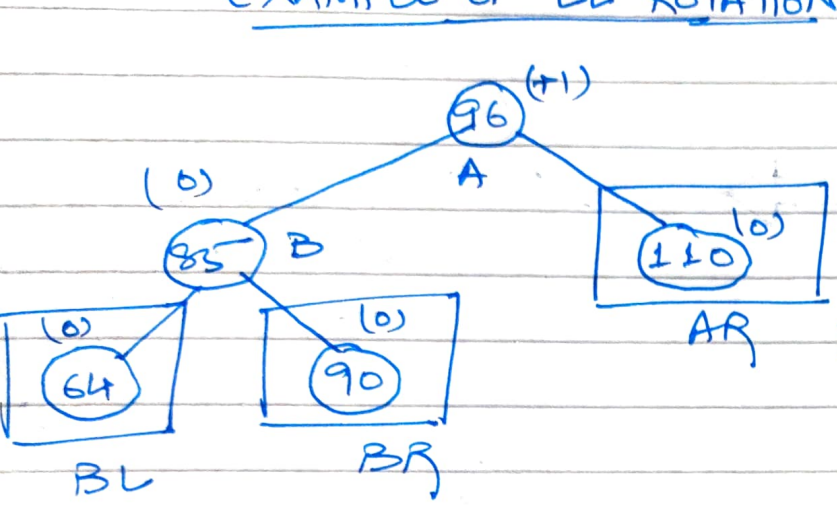
RR Rotation → Inserted node is in the right subtree of right subtree of node 'A'.

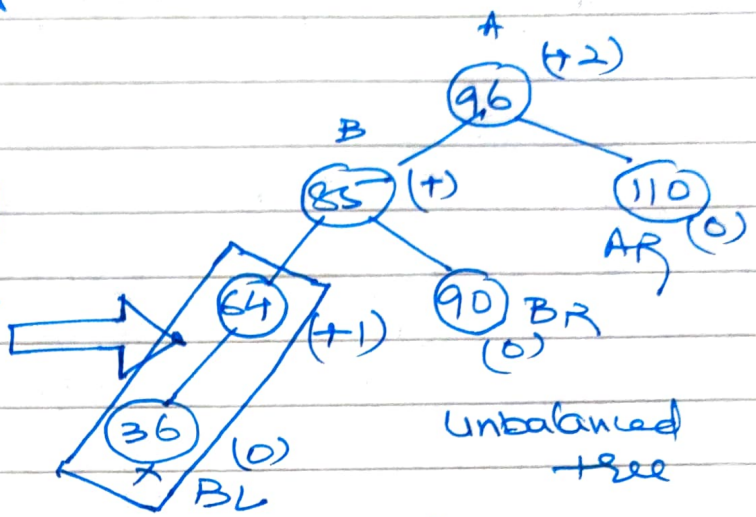
LR Rotation → Inserted node is in the right subtree of ~~right~~ left subtree of node 'A'

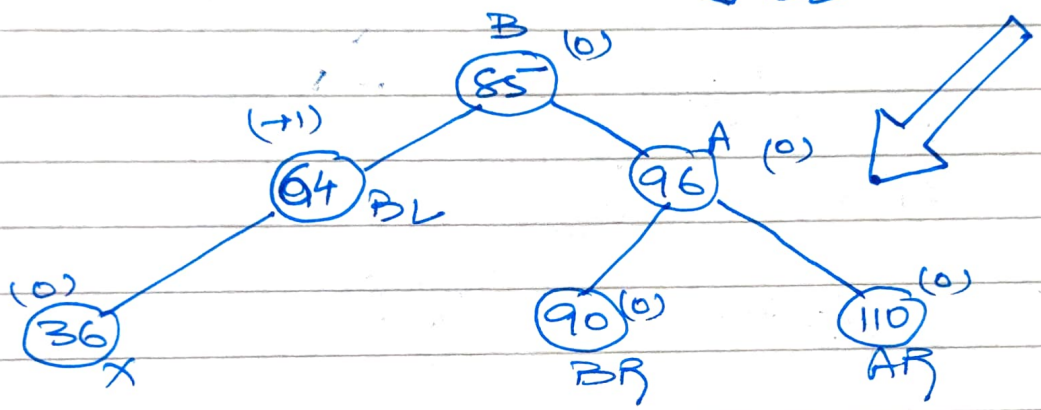
RL Rotation → Inserted node is in the left subtree of right subtree of node 'A'.

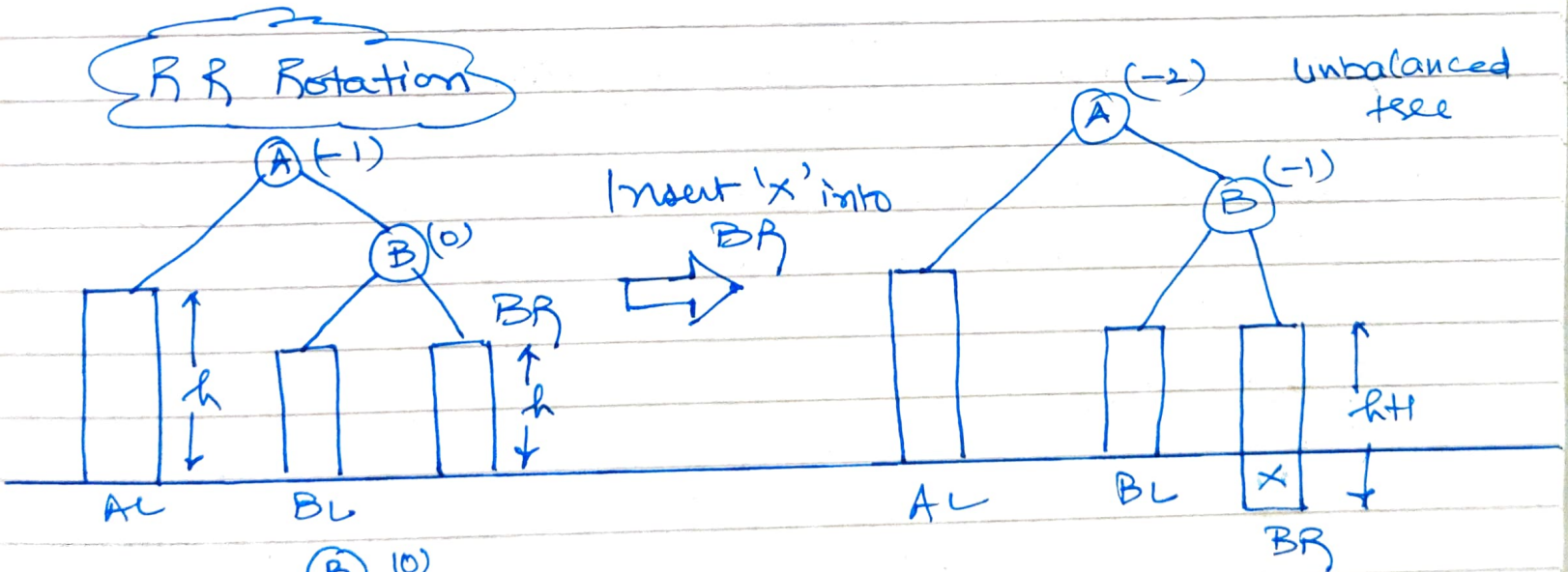


EXAMPLE OF LL ROTATION

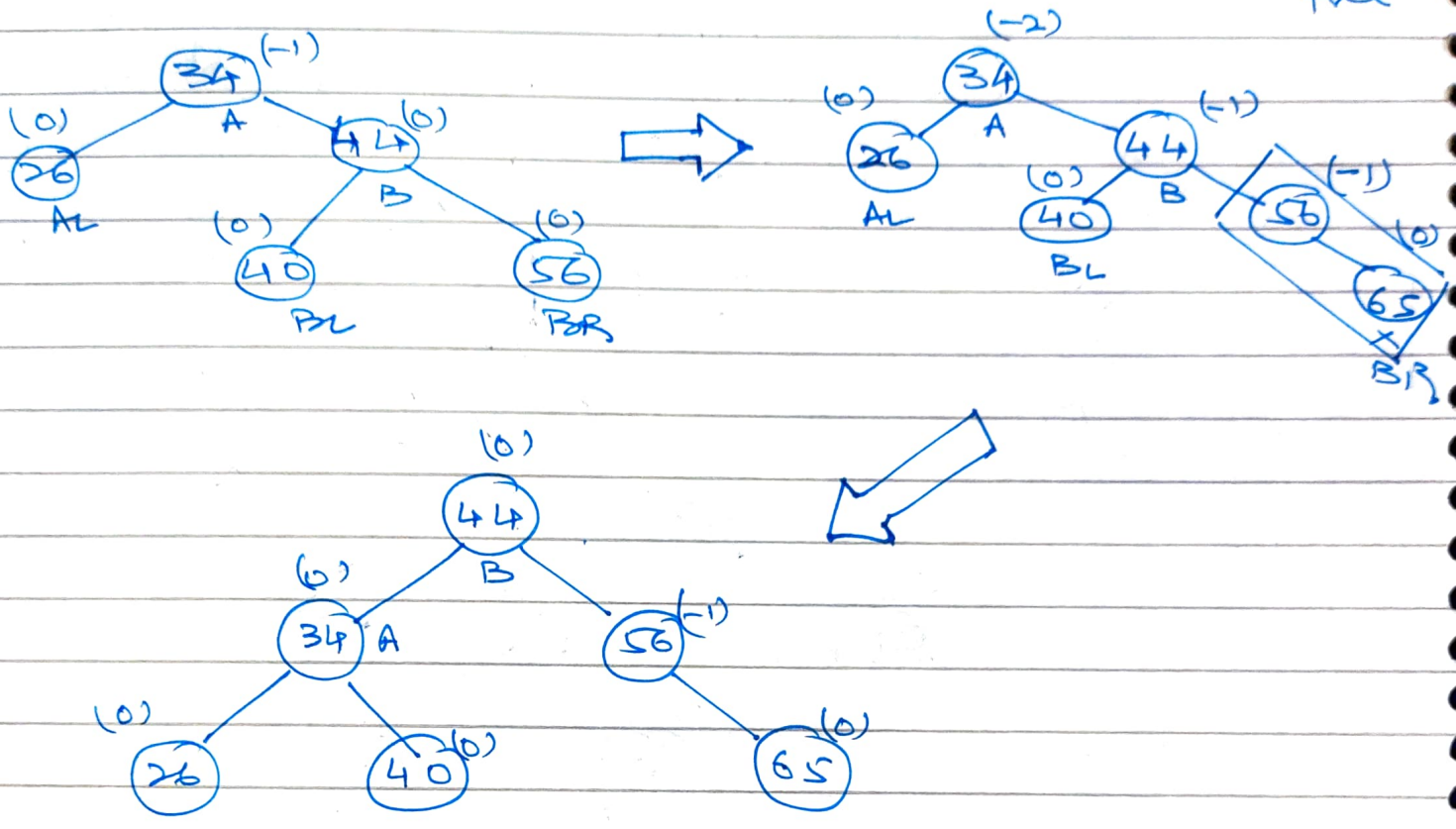








EXAMPLE OF RR ROTATION



LR & RL ROTATIONS — DOUBLE ROTATION

→ They are similar in nature but are mirror images of one another.

LR Rotation

In this case the balance factor (BF) values of nodes 'A' and 'B' are dependent up BF & value of node 'C' after insertion.

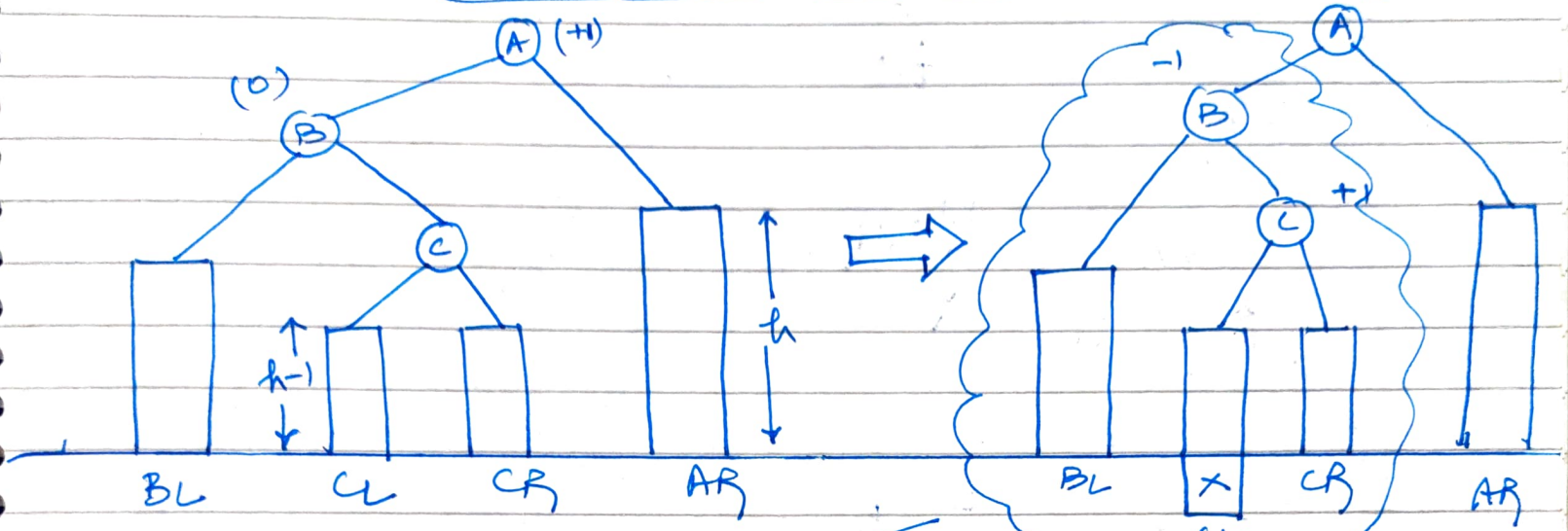
(1) if $Bf(C) = 0$ after insertion then $Bf(A) = Bf(B) = 0$ after Rotation.

(2) if $Bf(C) = -1$ after insertion then $Bf(A) = 0$, $Bf(B) = 1$

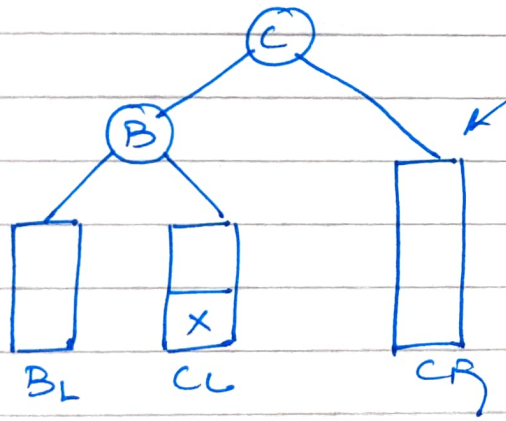
(3) if $Bf(C) = 1$ after insertion then $BF(A) = -1$, $Bf(B) = 0$

after Rotation.

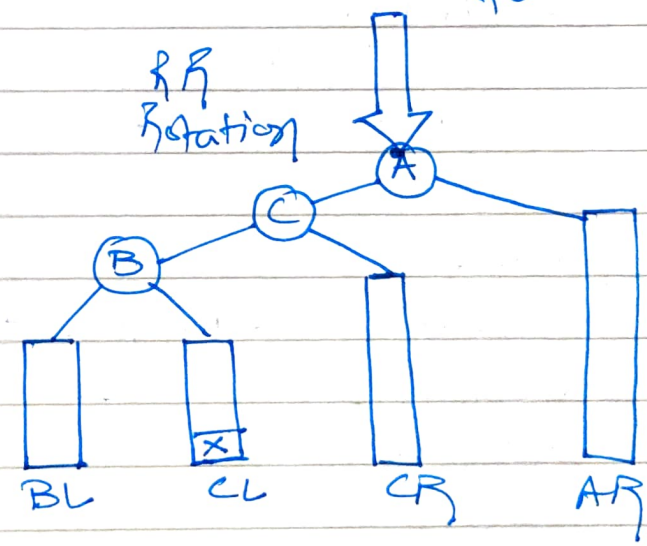
LR ⇒ RR + LL

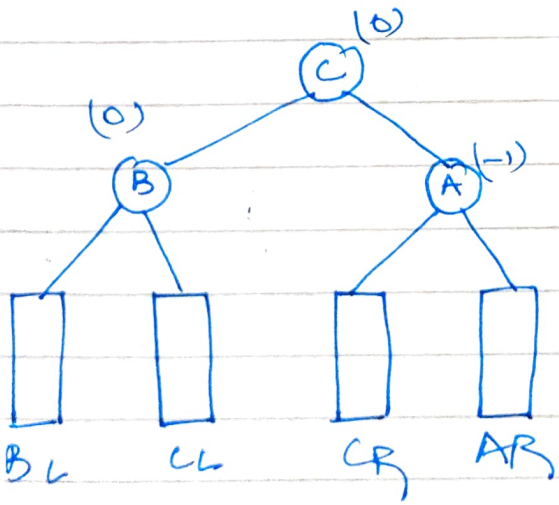


give RR rotation to subtree

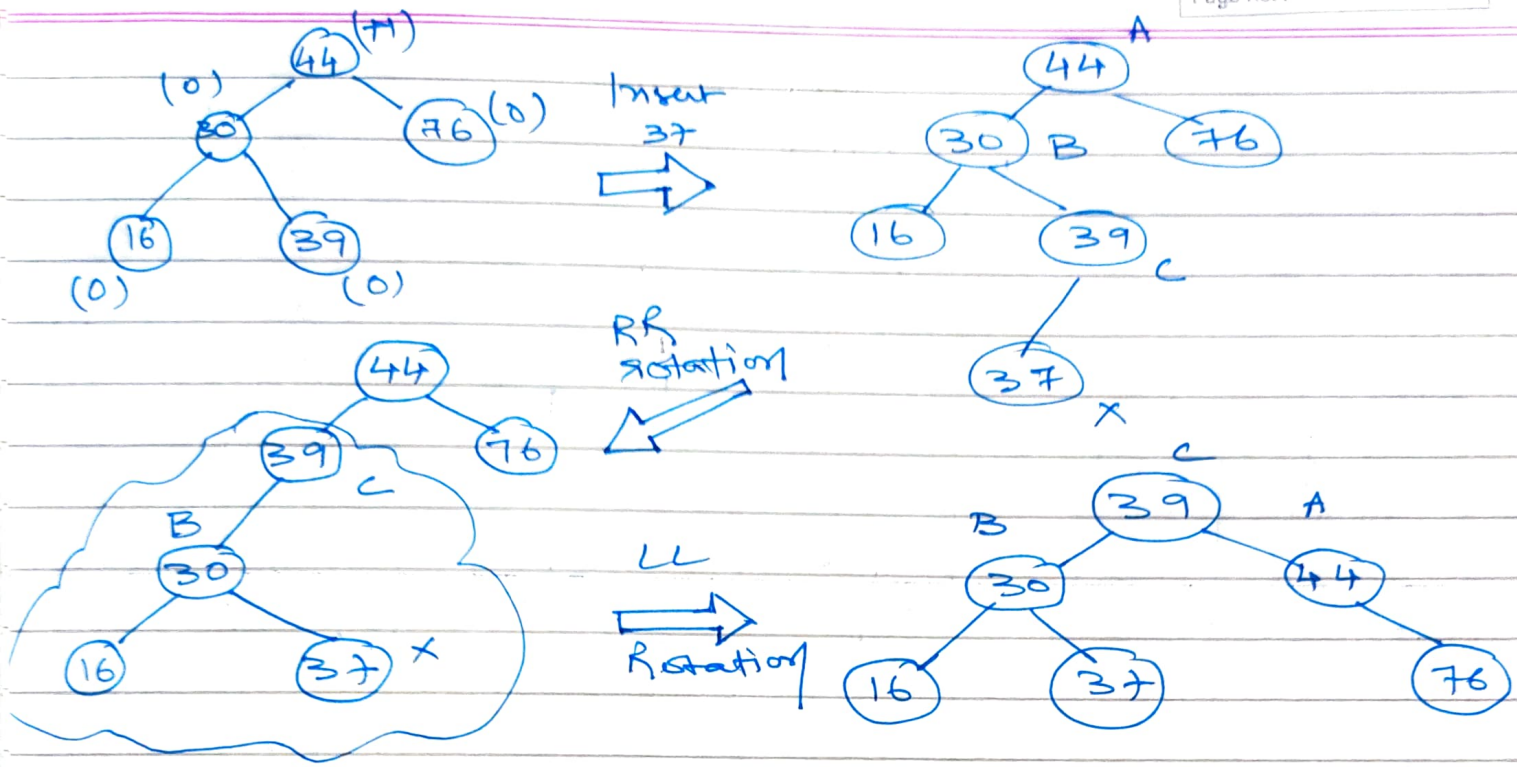


RR Rotation





LL Rotation



RL Rotation

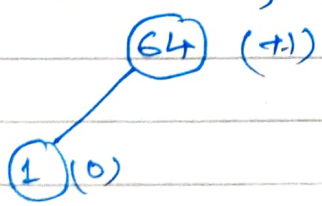
RL → LL + RR

Example

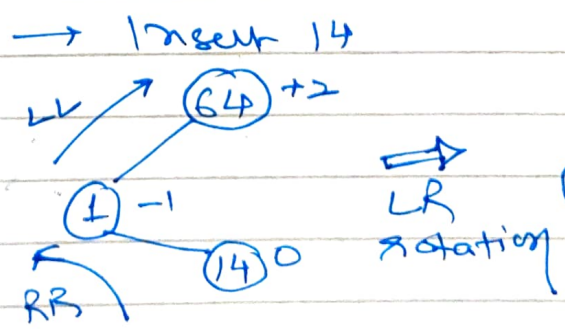
Construct a AVL tree by inserting the foll elements in order of their occurance.

64, 1, 14, 26, 13, 110, 98, 85

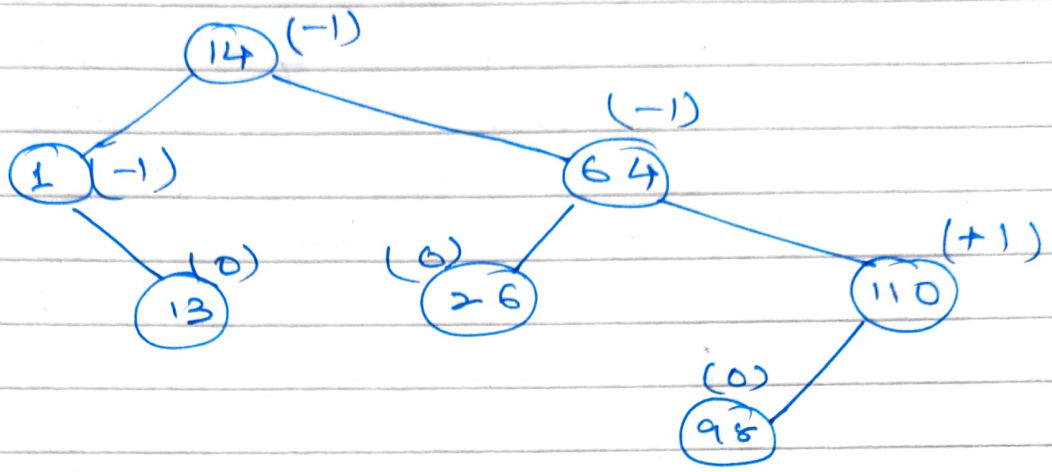
→ Insert 64, 1



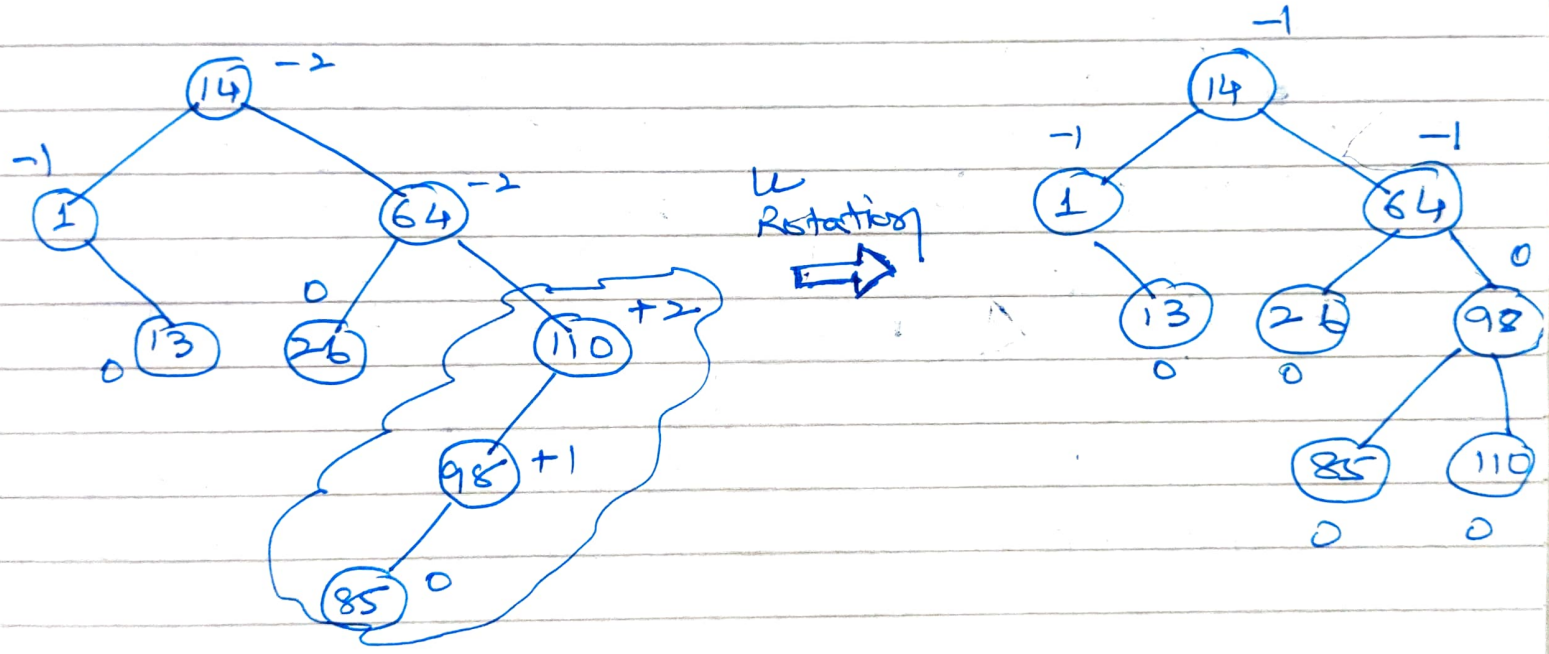
→ Insert 14



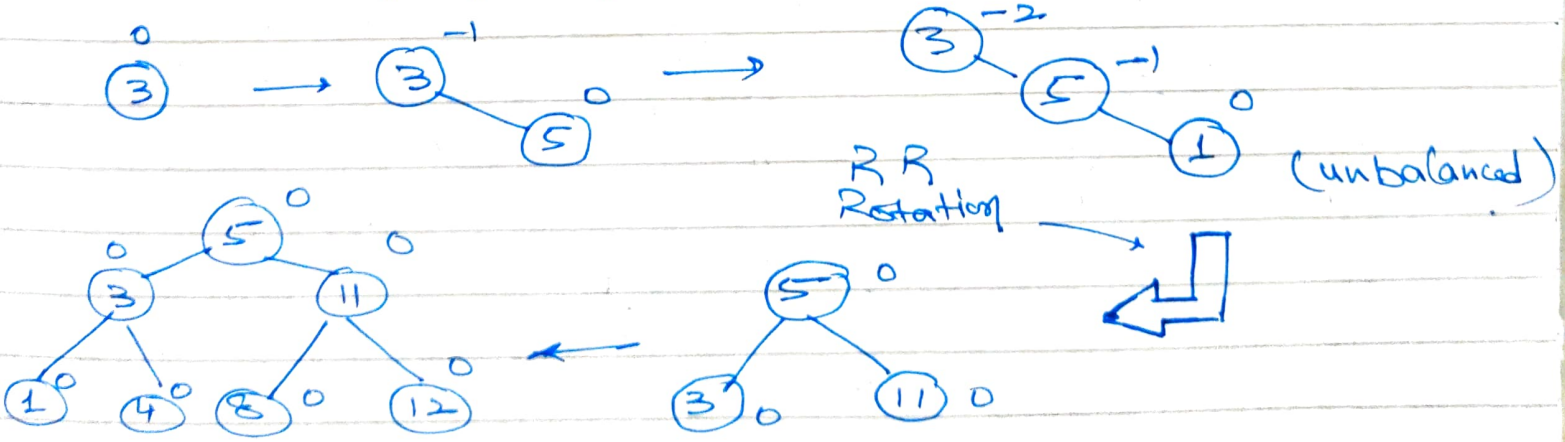
→ Insert 26, 13, 110, 98



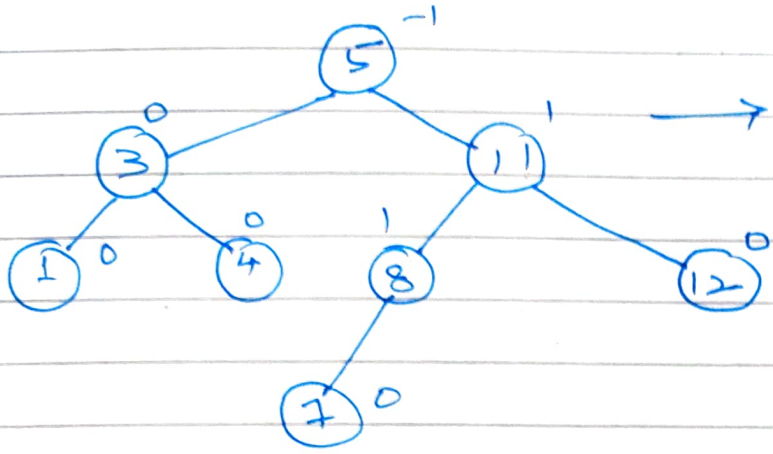
→ Insert 85



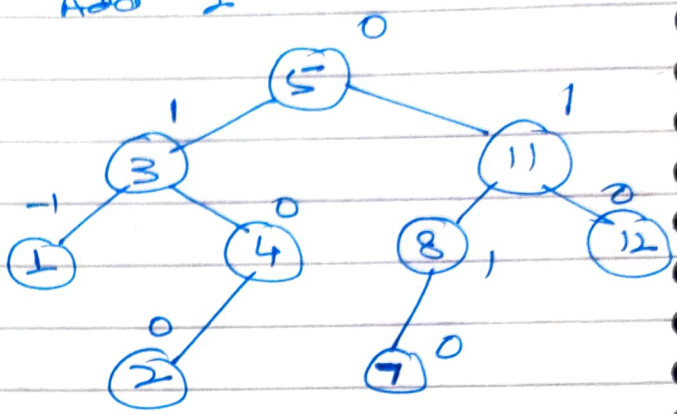
Example 3, 5, 11, 8, 4, 1, 12, 7, 2, 6, 10



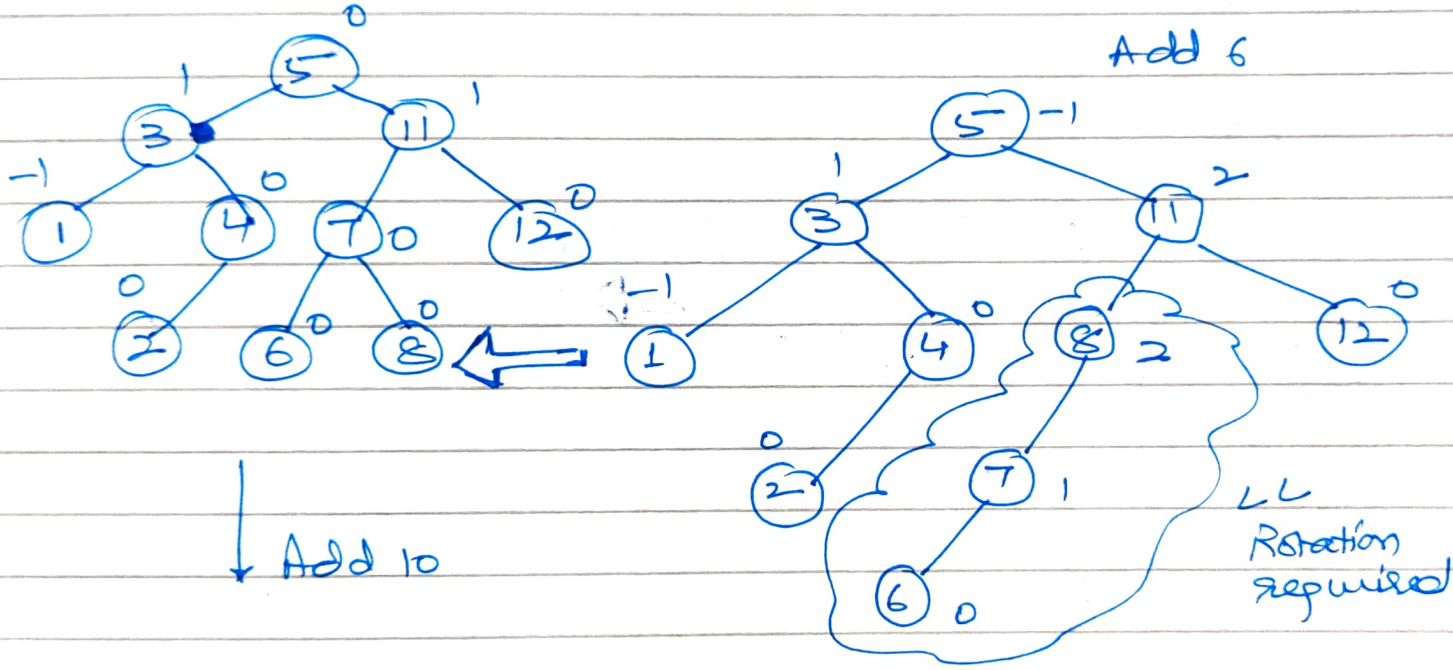
Add 7



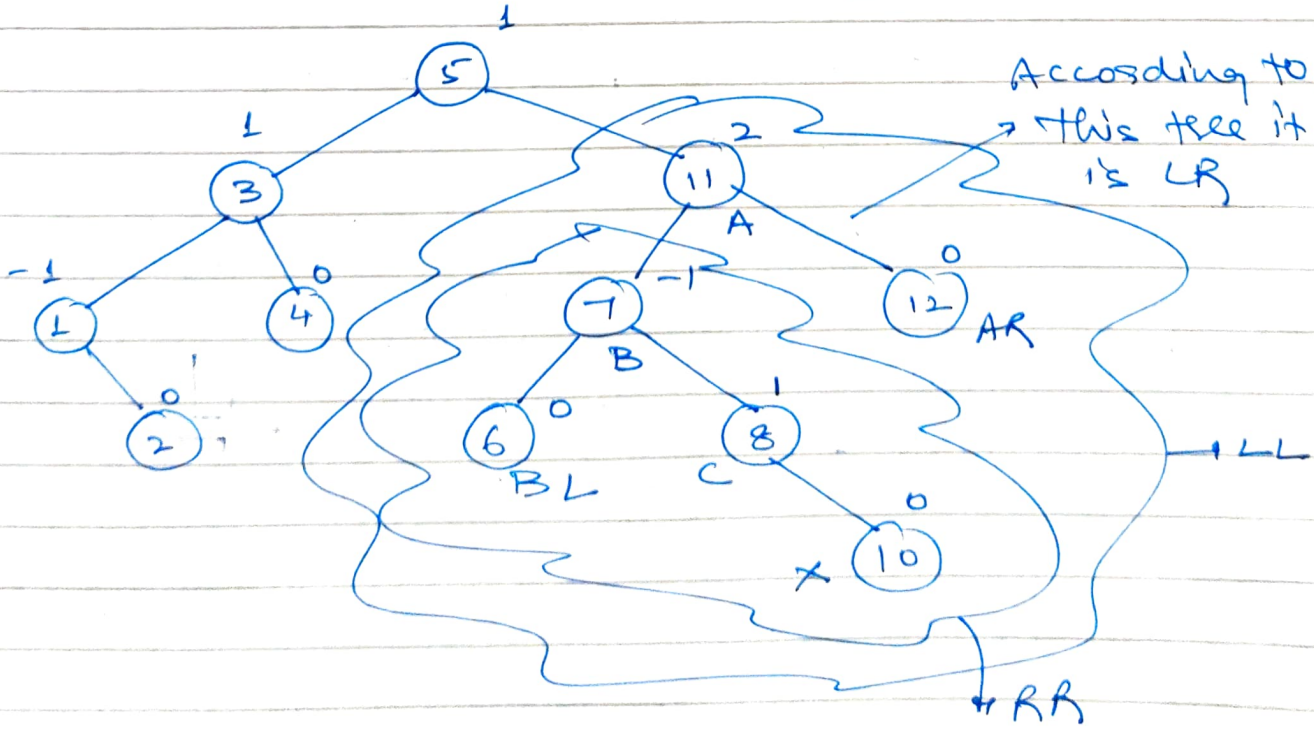
Add 2



Add 6



Add 10



In an inorder traversal since the closest parent to 10 is 11, we rotate tree in clock wise (L) direction to first obtain balance then we give another anti clock wise rotation.