# AVL Tree

A → Adelson

V → Velsky

L → Landis.

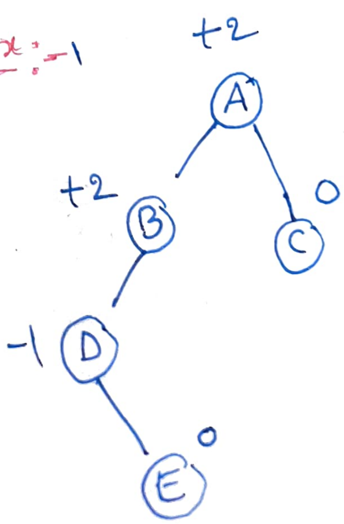
AVL Tree is a self-balancing Binary search Tree in which height difference b/w Left and right subTrees of Any node and balance factor of Each node is -1, 0, 1.

**Balance Factor:-** The balance factor for any node N is denoted as BF(N).

$$\boxed{BF(N) = h_L - h_R}$$

where
$h_L$ = height of Left subtree
$h_R$ = height of Right subtree

Ex:-1




(Binary Tree)

$BF(A) = 3 - 1 = 2$

$BF(B) = 2 - 0 = 2$

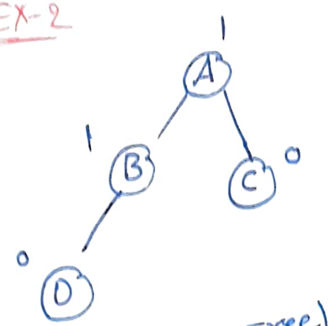
$BF(C) = 0 - 0 = 0$

$BF(D) = 0 - 1 = -1$

$BF(E) = 0 - 0 = 0$

Thus, this Tree is not an AVL Tree because left subtree is not balanced and also root node is not balanced.

EX-2



$BF(A) = 2-1 = 1$
$BF(B) = 1-0 = 1$
$BF(C) = 0-0 = 0$
$BF(D) = 0-0 = 0$

(Binary Tree)

∴ The above binary Tree is an AVL Tree

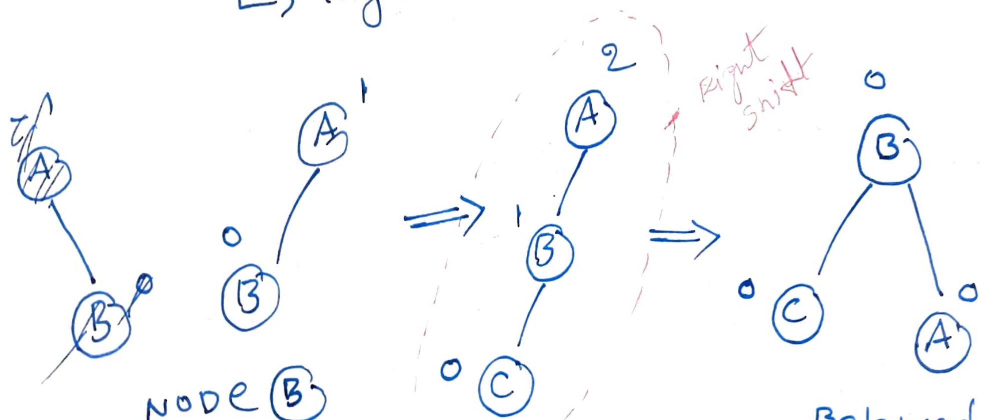
If a Binary Tree is not balanced then it can be balanced by the help of any one of the ROTATION methods. These Rotations are:

(1) LL Rotation
(2) RR Rotation
(3) LR Rotation
(4) RL Rotation.

(1) LL Rotation :-

↳ Right Rotation

Right Shift



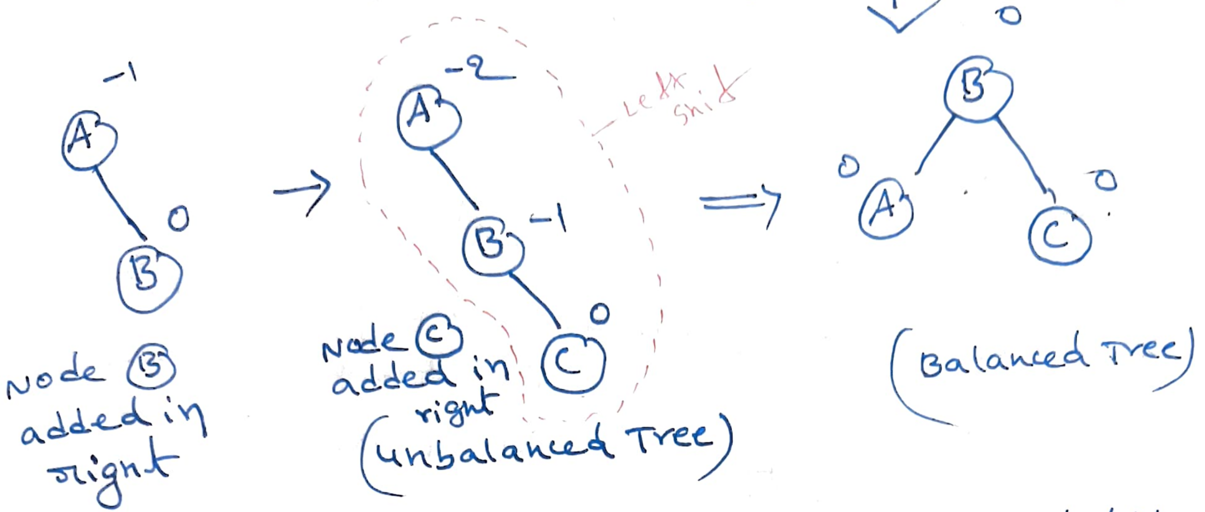

NODE B added in Left

(unbalanced Tree)

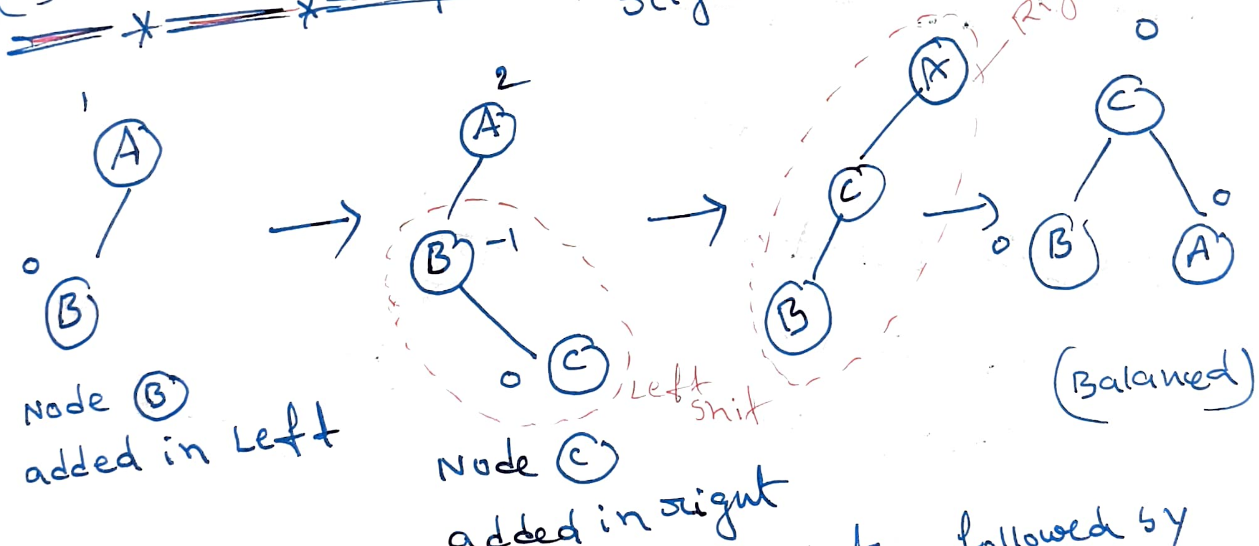
Balanced Tree

## (2) RR Rotation :-

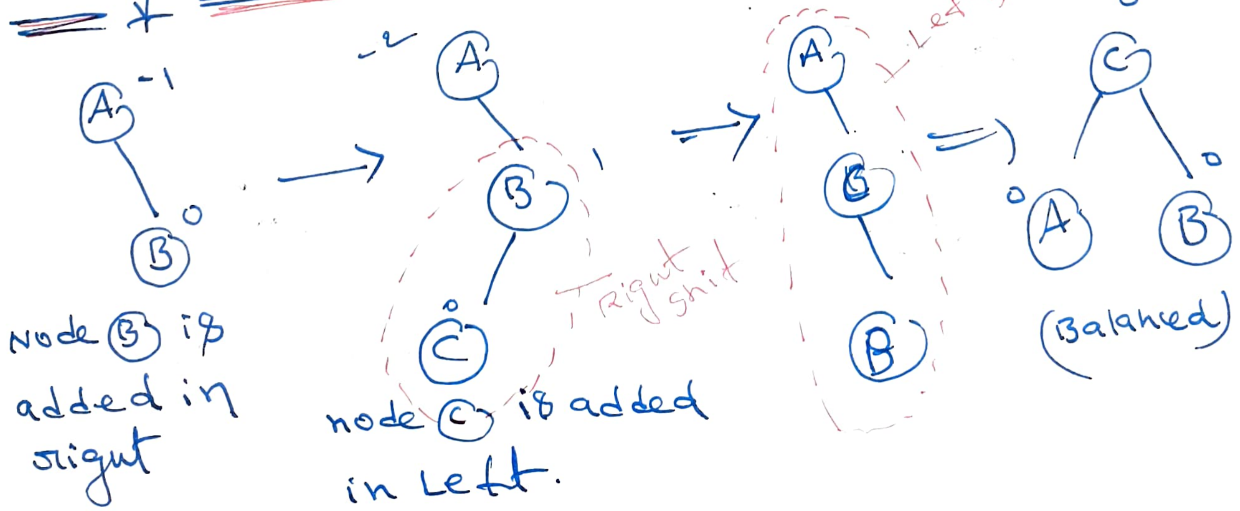
↳ Left Rotation




## (3) LR Rotation :- {Left rotation followed by right rotation}




## (4) RL Rotation :- {Right rotation followed by Left rotation}



# problem-1 AVL Tree Insertion

create an AVL Tree of the following Elements. {56, 69, 80, 18, 14, 36, 25, 38, 28, 50, 87, 98}

**solution:-**

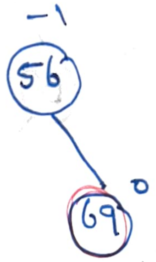
Insertion of data items is an AVL Tree is same as insertion in BST. Here only we have to check the balanced Factor and balance the Tree After insertion of one item, then another item and so on.

**step-1:-** insert node 56


**step-2:-** insert 69

69 > 56 (yes)

56[right] = 69




(balanced)

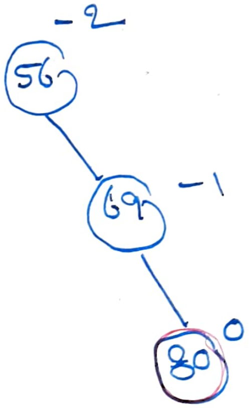
**step-3** insert 80

80 > 56 (yes)

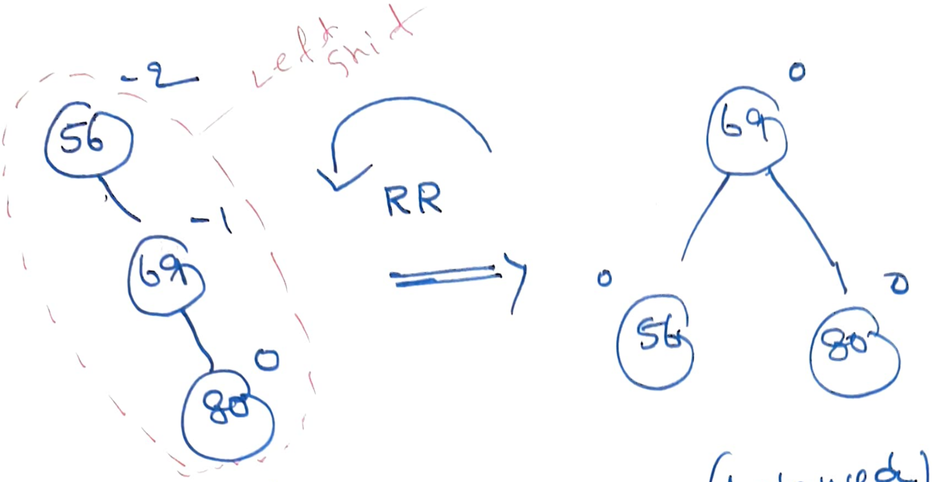
56[right] = 80

80 > 69 (yes)

69[right] = 80

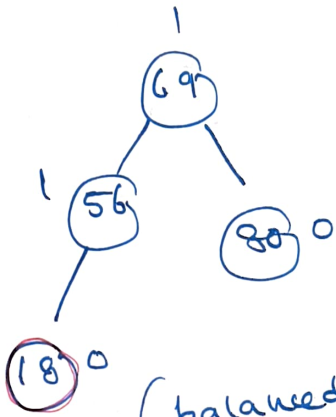



(unbalanced)




(unbalanced) (balanced)

## Step-4 insert 18




18 > 69 (F)

69[Left] = 18

18 > 56 (F)

56[Left] = 18

(balanced).

## Step-5 insert 14

14 > 69 (F)

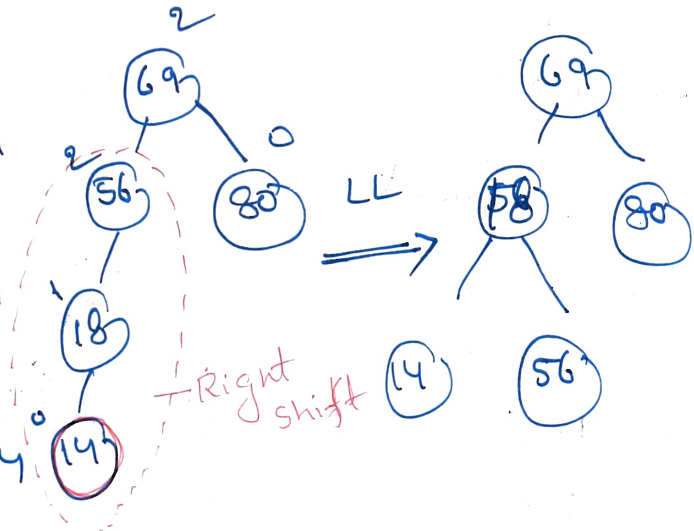
69[Left] = 14

14 > 56 (F)

56[Left] = 14

14 > 18 (F)

18[Left] = 14




Here (69) (56) both are unbalanced then go for balancing Lower node, where the (14) is inserted.

## step-6:- insert 36

36 > 69 (F)

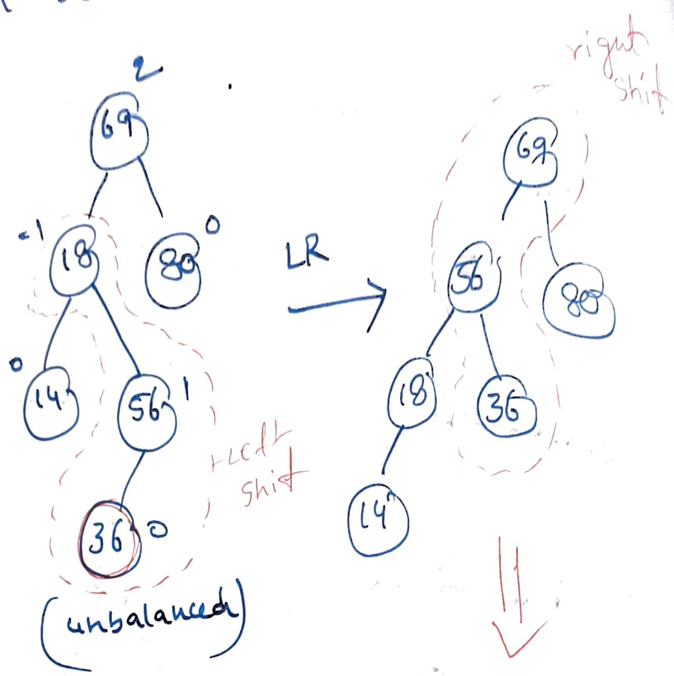
69[Left] = 36

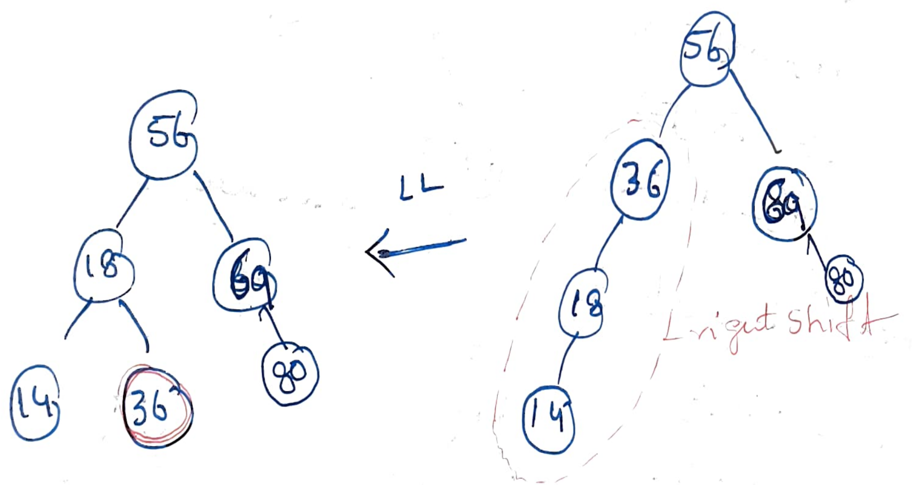
36 > 18 (T)

18[right] = 36

36 > 56 (F)

56[Left] = 36




## step-7 insert 25

25 > 56 (F)

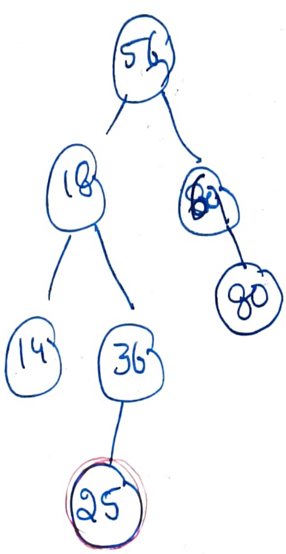
56[Left] = 25

25 > 18 (F)

18[right] = 25

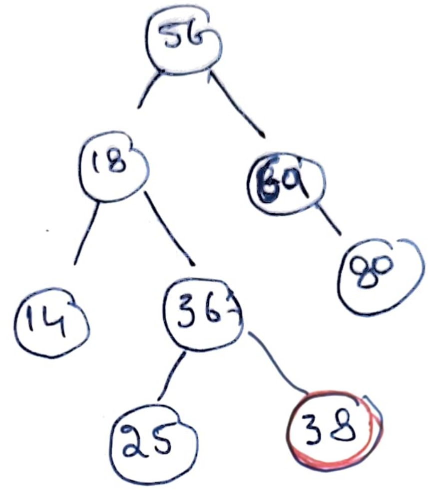
25 > 36 (F)

36[Left] = 25



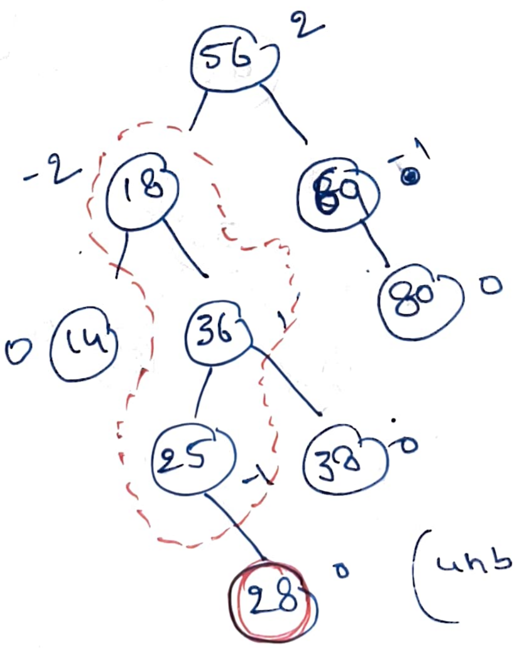
step-8 :- insert 38

38 > 56 (F)
56[left] = 38
38 > 18 (T)
18[right] = 38
38 > 36 [T]
36[right] = 38



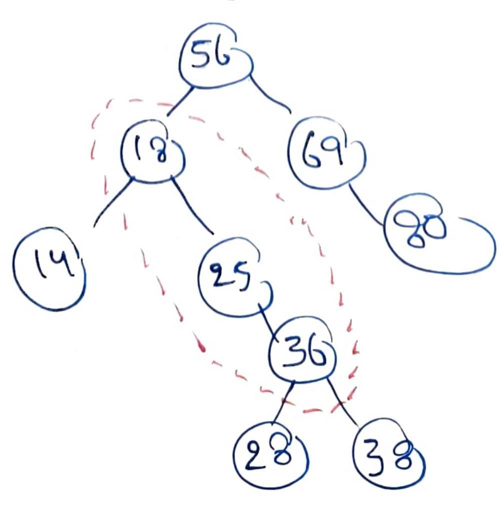
step-9 :- insert 28

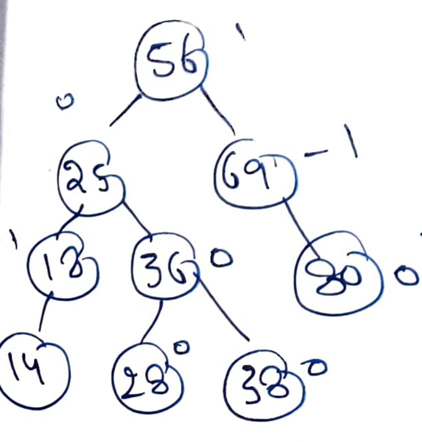
28 > 56 (F)
56[Left] = 28
28 > 18 [T]
18[right] = 28
28 > 36 [F]
36[Left] = 28
28 > 25 [T]
25[right] = 28



(unbalanced)

RL





(balanced)

Step-10 :- insert 50

50 > 56 [F]

56[Left] = 50

50 > 25 (T)

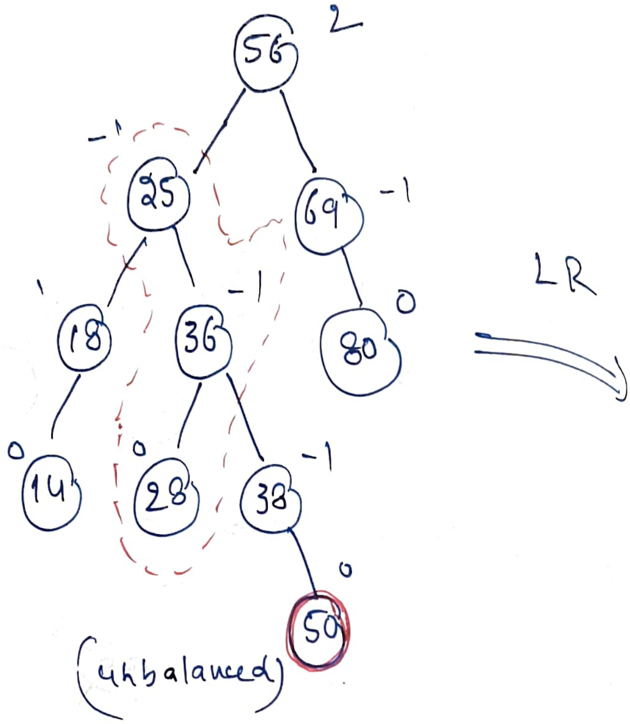
25[right] = 50

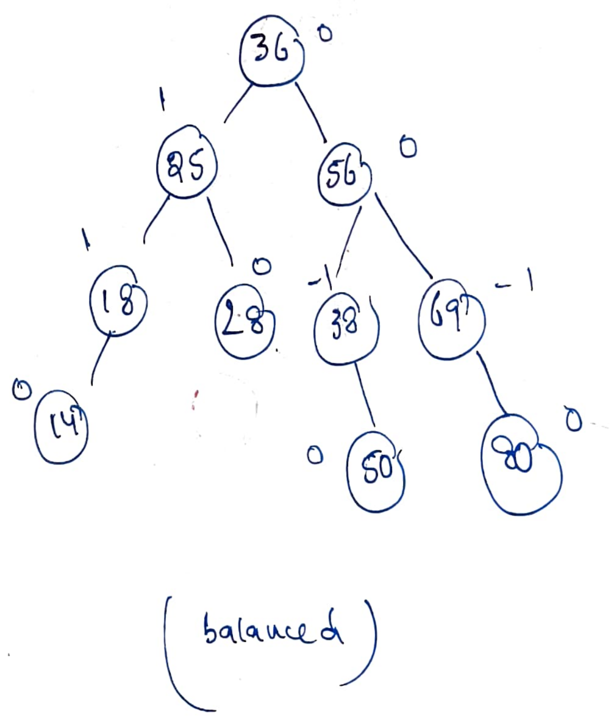
50 > 36 [T]

36[right] = 50

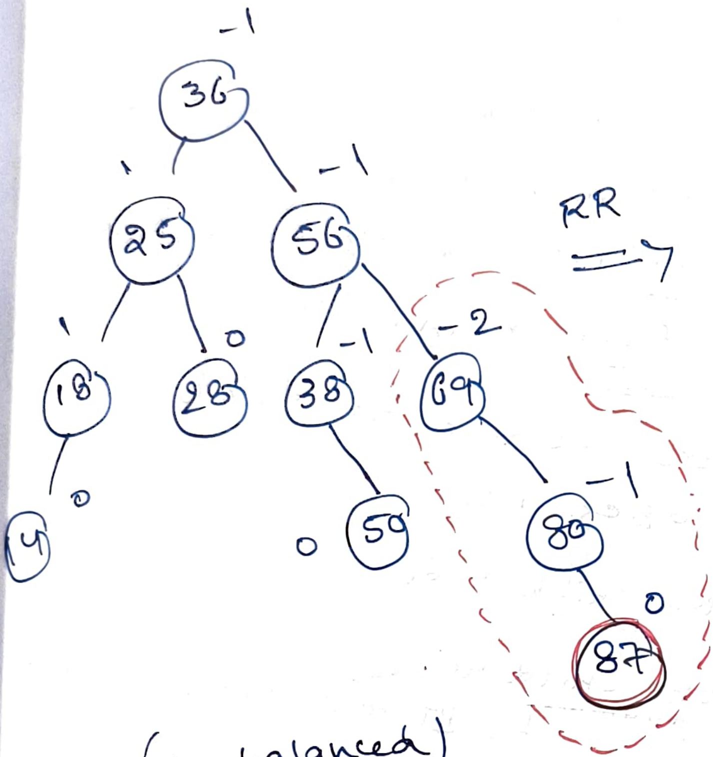
50 > 38 [T]

38[right] = 50

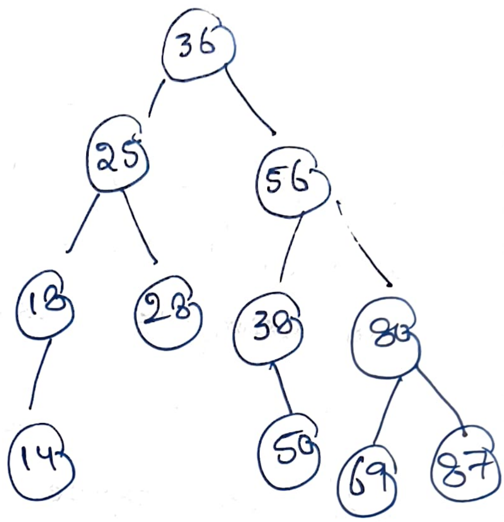


step-11 :- insert 87



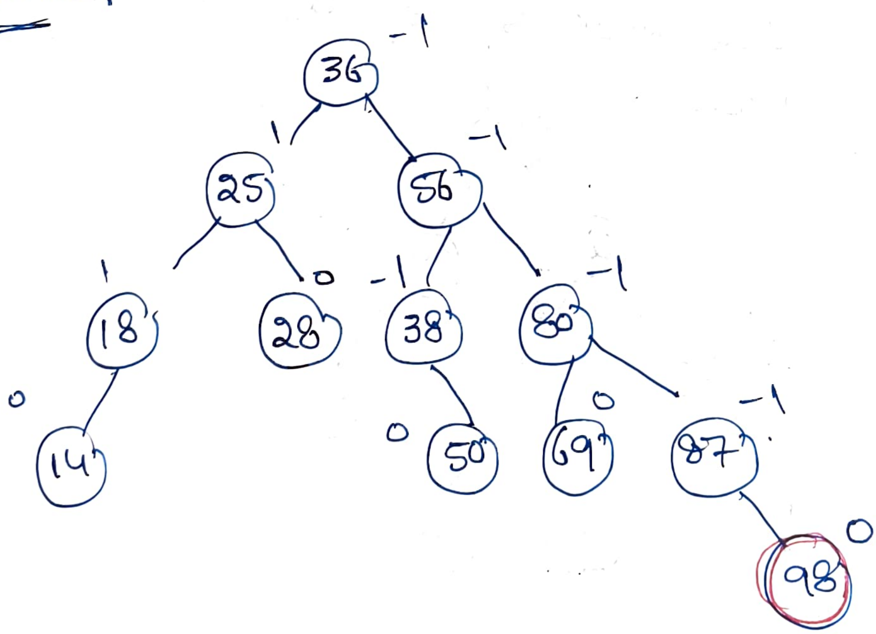

RR ⇒




(unbalanced)

(balanced)

step-12 := insert 98




(balanced).

# AVL Tree Deletion

Deletion in AVL Tree is same as deletion in BST. But here After deleting one node we have to balance the Tree using LL, RR, LR & RL Rotation to make balanced AVL Tree.

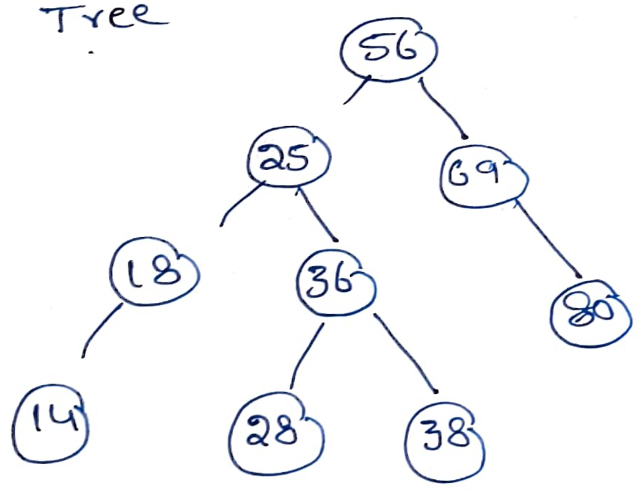
Here we have to delete the node in three cases i.e

case 1: No child node

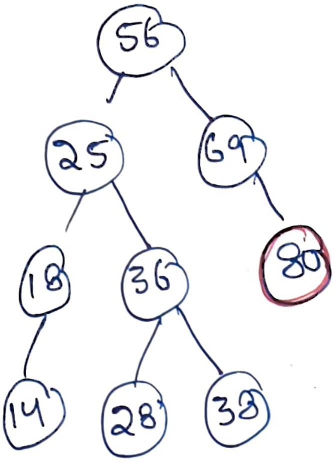
case 2: one child node
↳ LT may Left or right

case 3: Two child node

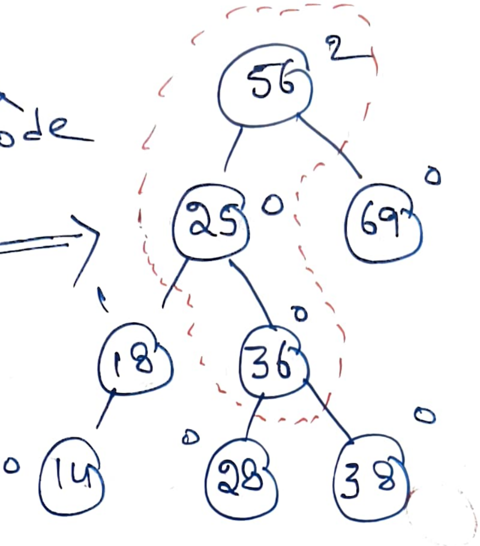
**Ex:-1** Delete NODE 80 of the following AVL Tree




**solution:-** Here Delete node 80 have No chil node so it's under case 1.



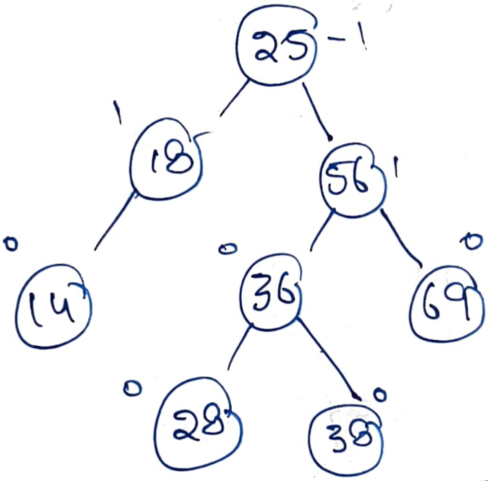
Deleted node 80



(balanced)

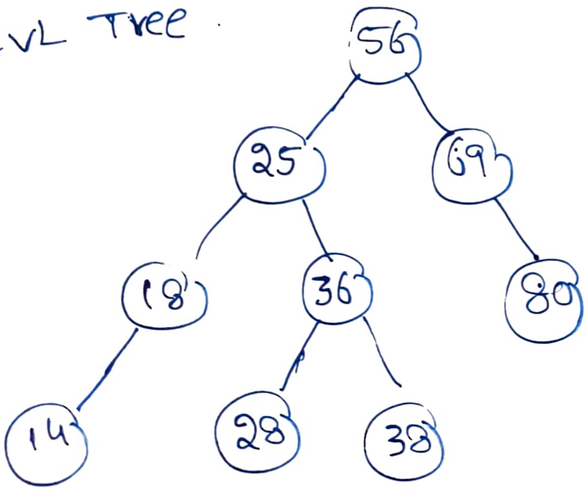
(unbalanced)

LL



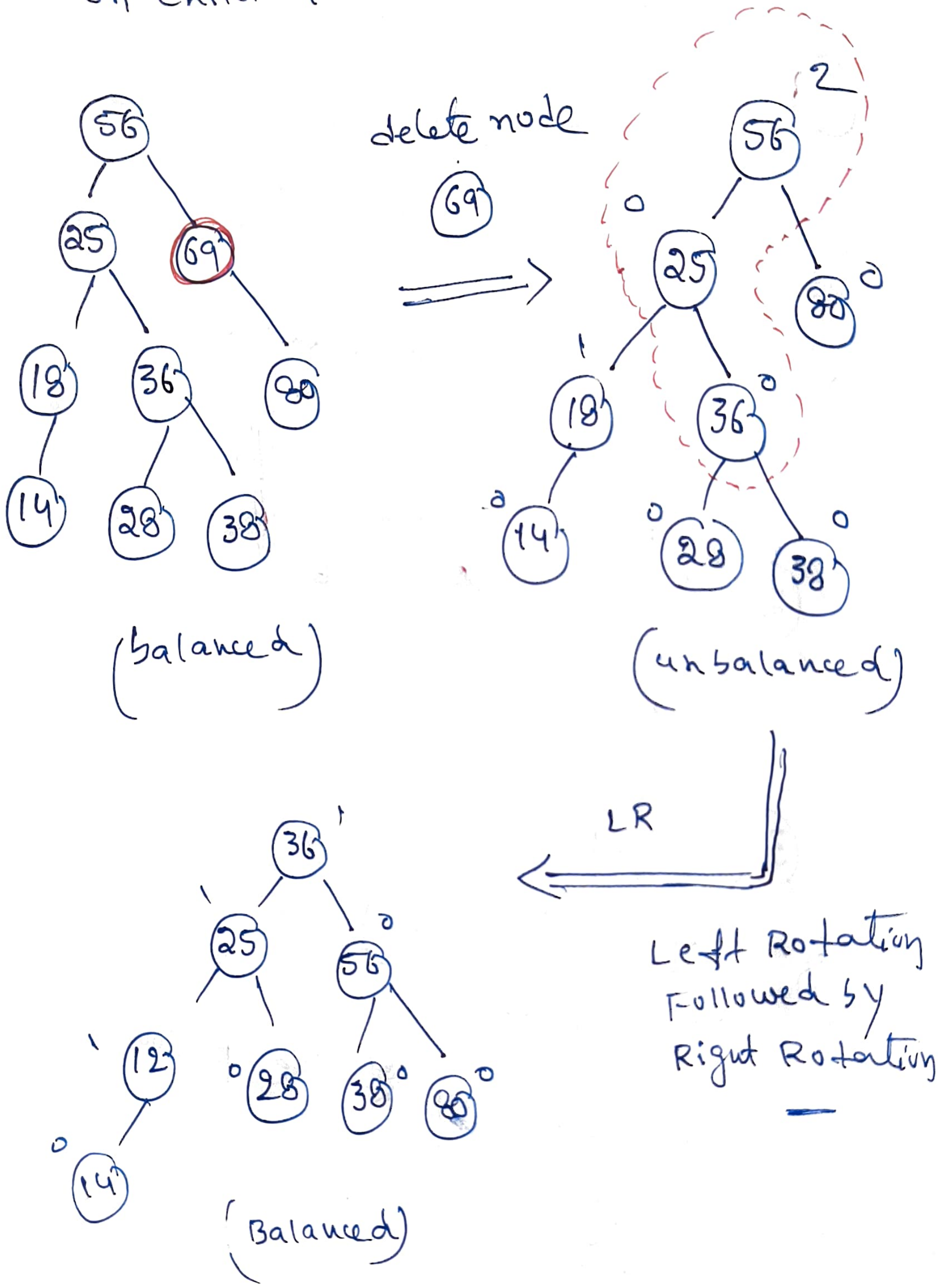
(Balanced)

**Ex-2** Delete node 69 of the following AVL Tree.

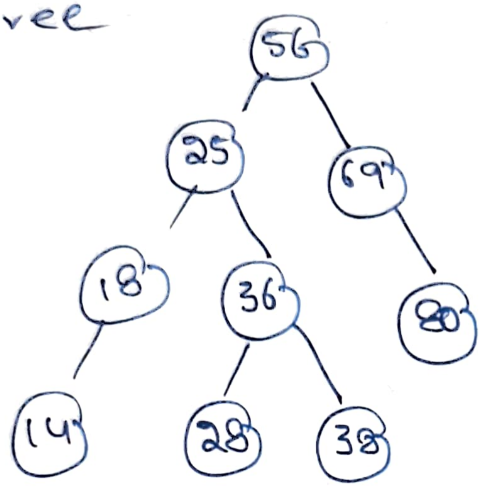


Solution :-

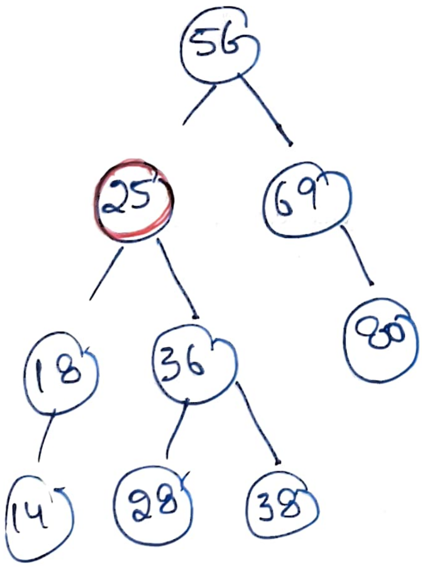
Here delete node 69 have on child node so It's under case 2.



Ex-3:— Delete node 25 of the following AVL Tree

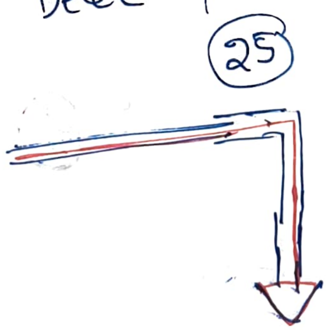



**Solution:—** Delete node 25 have two child node so It's under case 3.

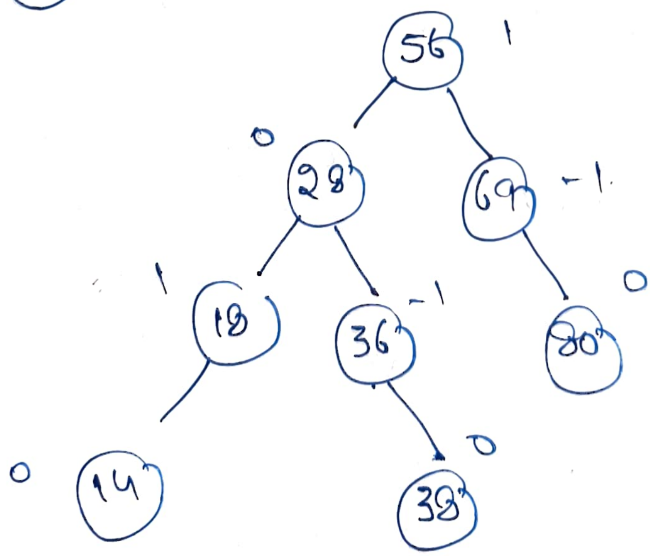



(balanced)

Delete node




25 has two child node.




(balanced)

problem :- construct AVL Tree of the following Elements {10, 15, 9, 12, 13, 79, 45, 36, 22}.

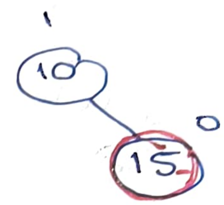
solution :-

step-1 insert 10

step-2 insert 15

15 > 10 (T)

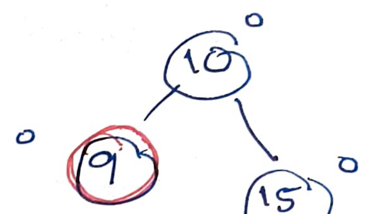
10{right} = 15



step-3 insert 9

9 > 10 (F)

10{Left} = 9



(balanced)

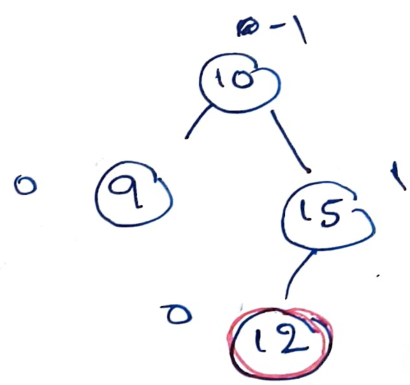
step-4 :- insert 12

12 > 10 (T)

10{right} = 12

12 > 15 (F)

15{Left} = 12



(balanced)

step-5 insert 13

13 > 10 (T)

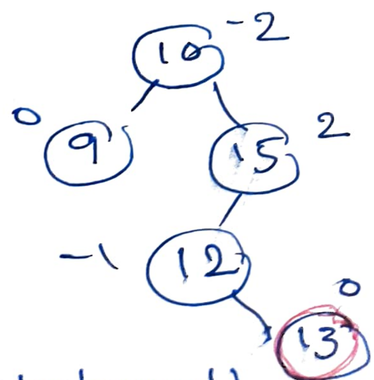
10[right] = 13

13 > 15 (F)

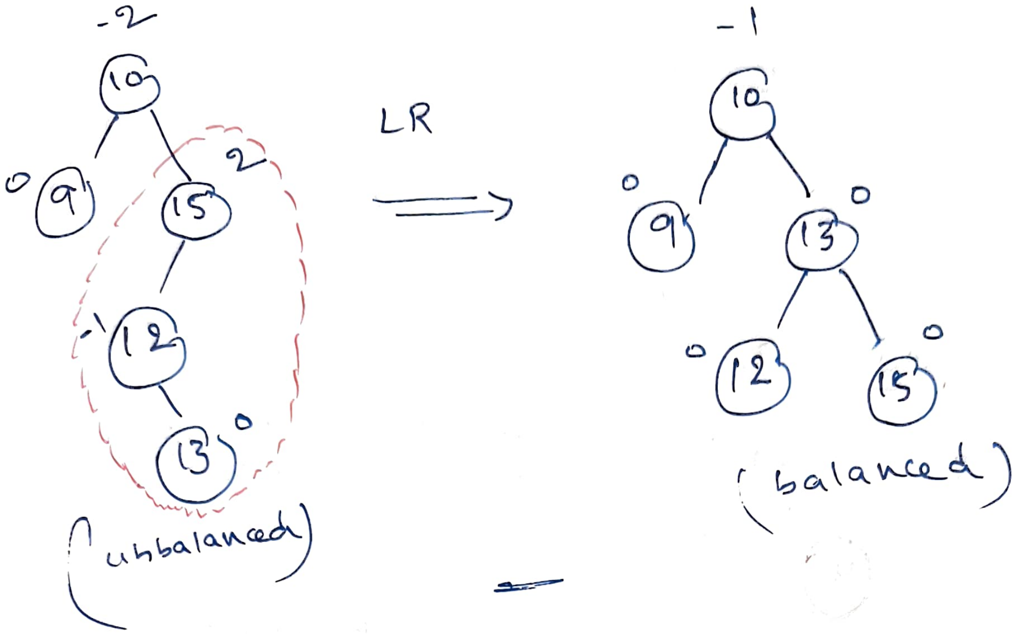
15(Left) = 13

13 > 12 (T)

12{right} = 13



(unbalanced)




Step-6 : 79

79 > 10(T)

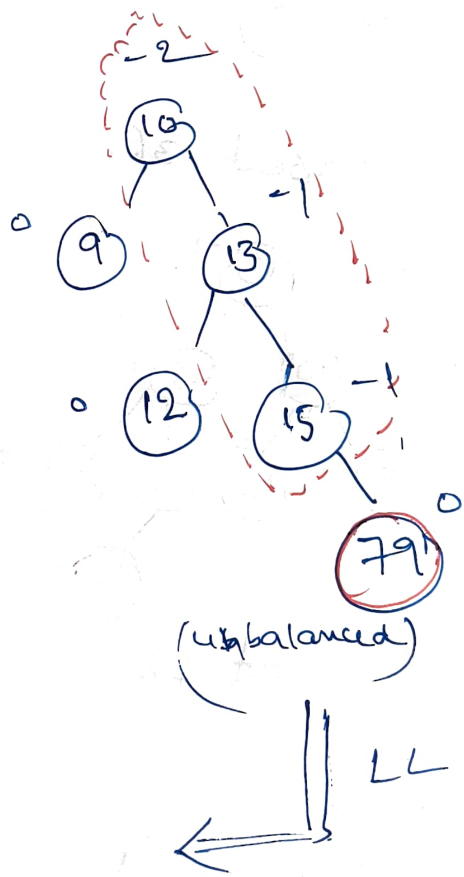
10[right] – 79

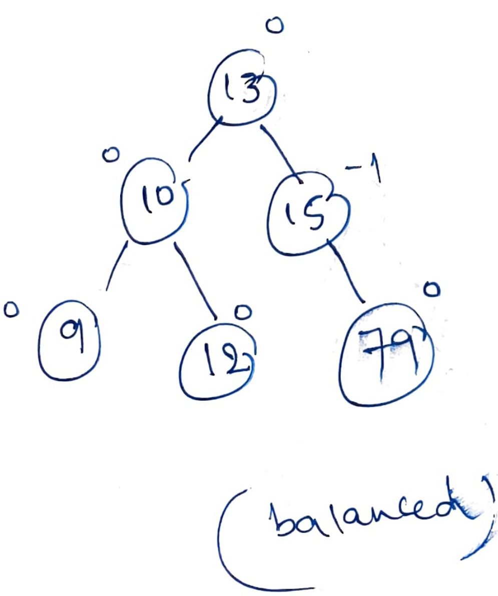
79 > 13[T)

13[right] – 79

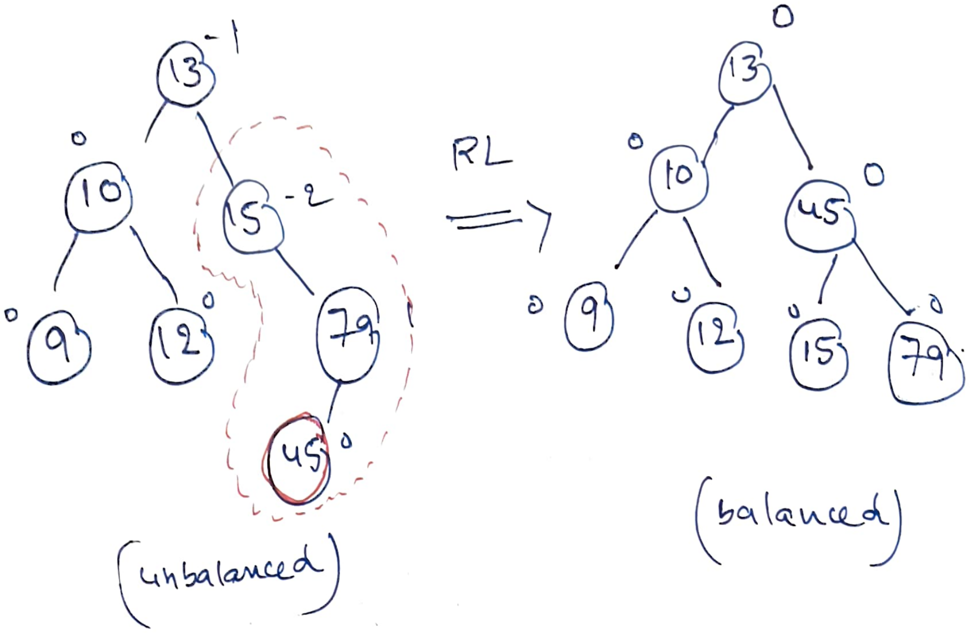
79 > 15{T)

15{right] – 79

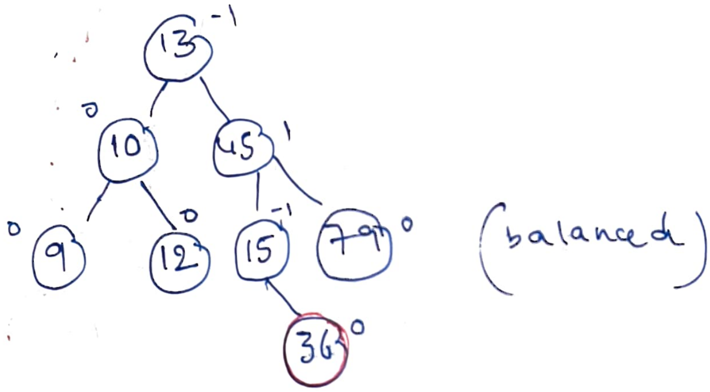


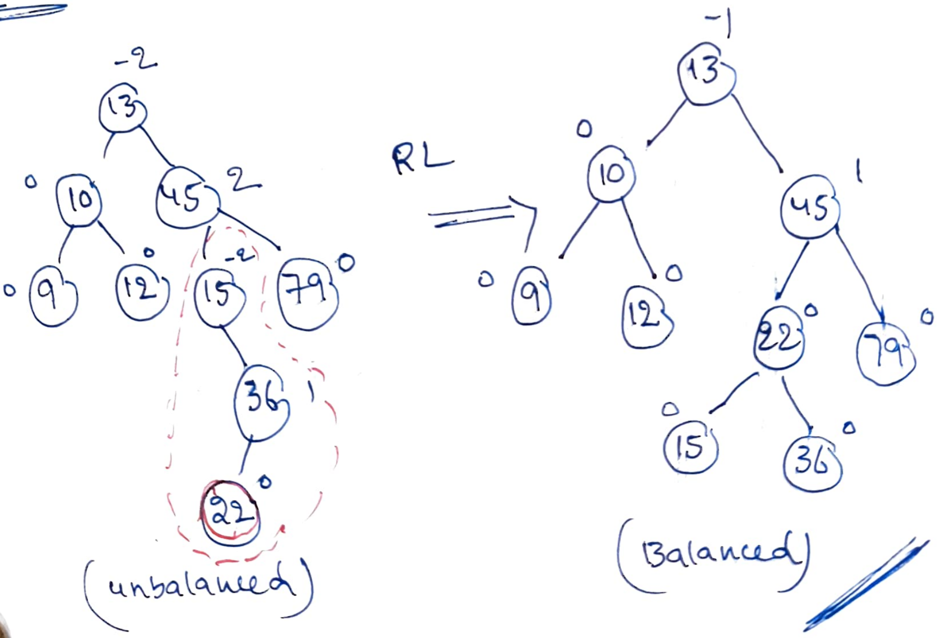
Step 7:- insert 45




Step 8:- insert 36.




Step 9:- insert 22



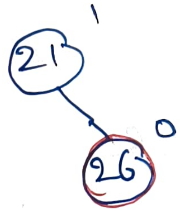
**problem:-** construct AVL Tree of the following

Data {21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7}

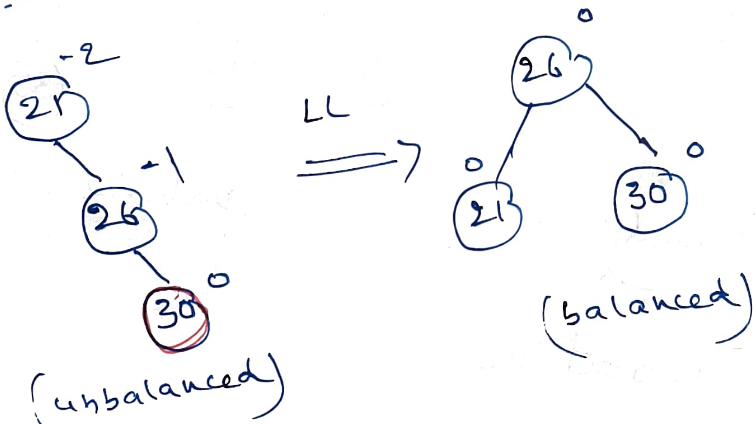
**solution:-**

**step-1:-** insert 21

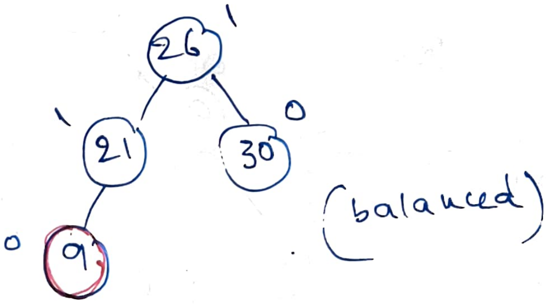

**step-2** insert 26



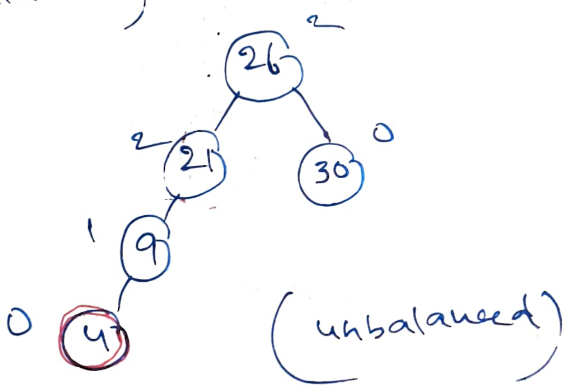

**step-3:-** insert 30

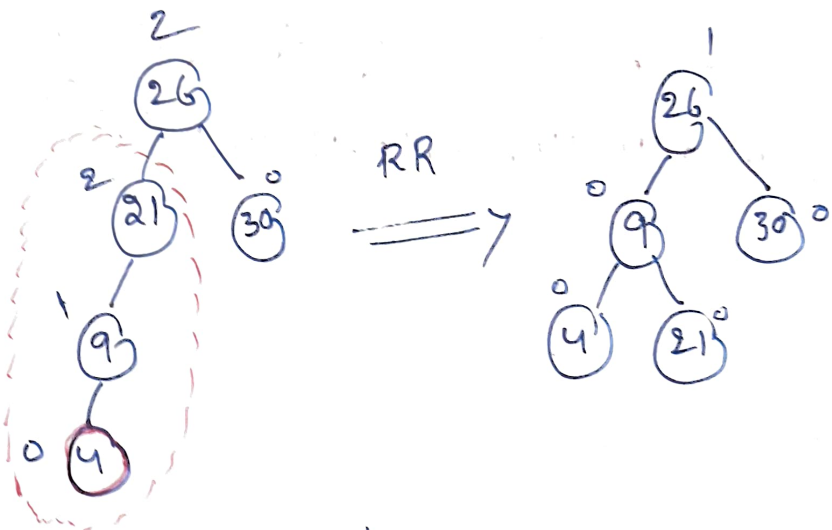



**step-4:-** insert 9

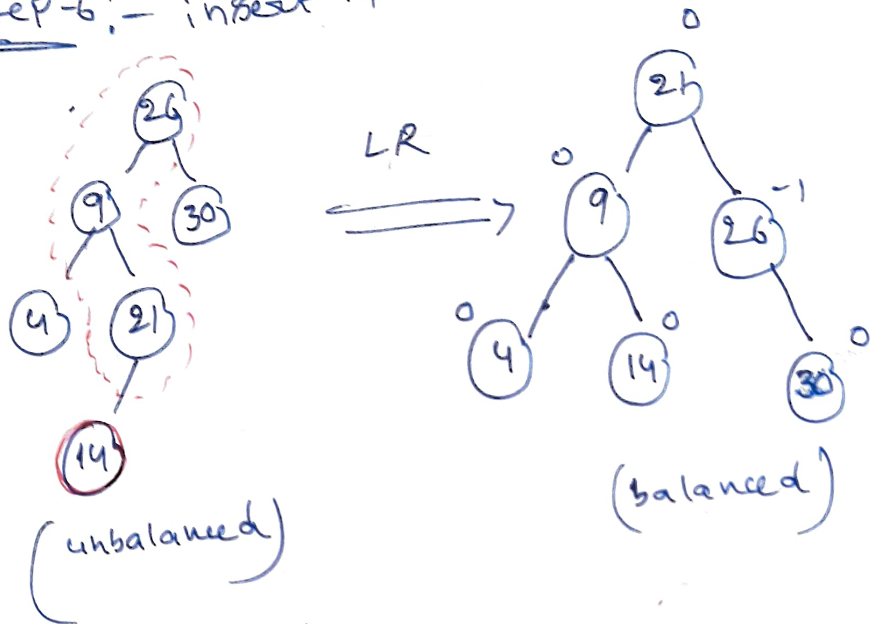



**step-5:-** insert 4

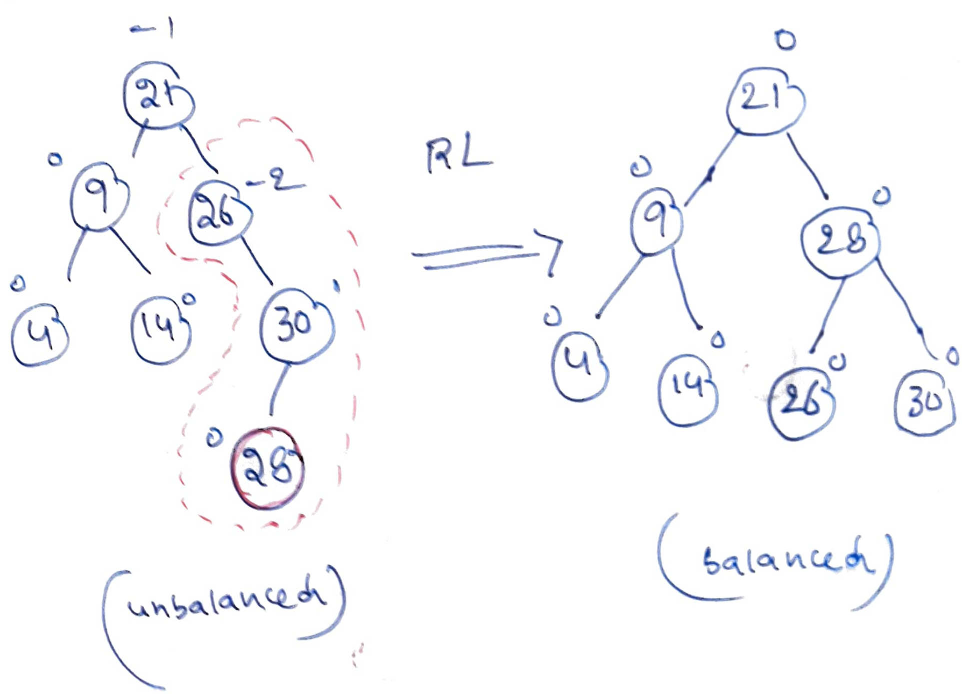




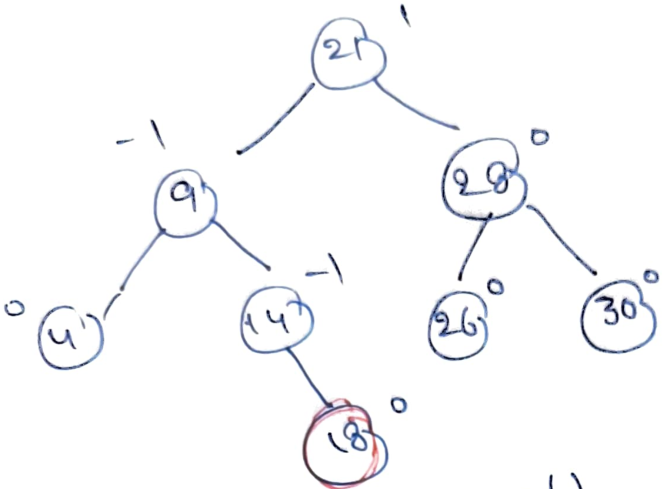

Step-6:- insert 14




step 7:- insert 28

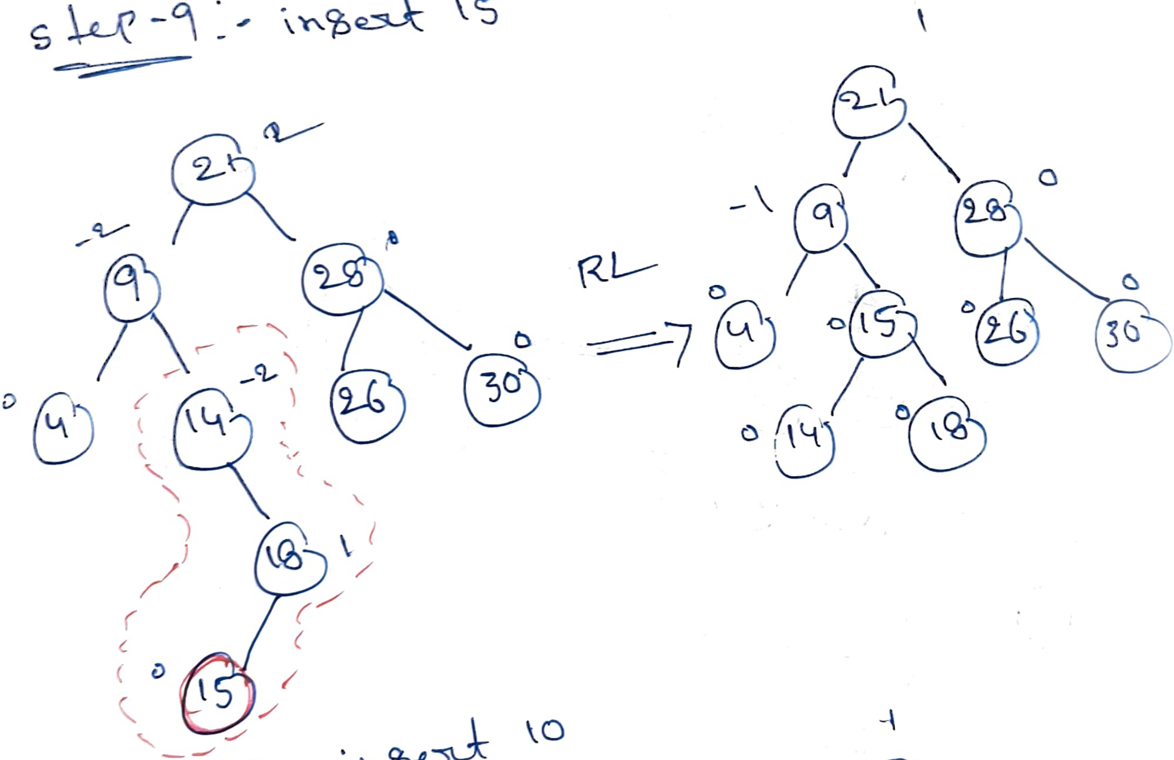


## step-8 :- insert 18

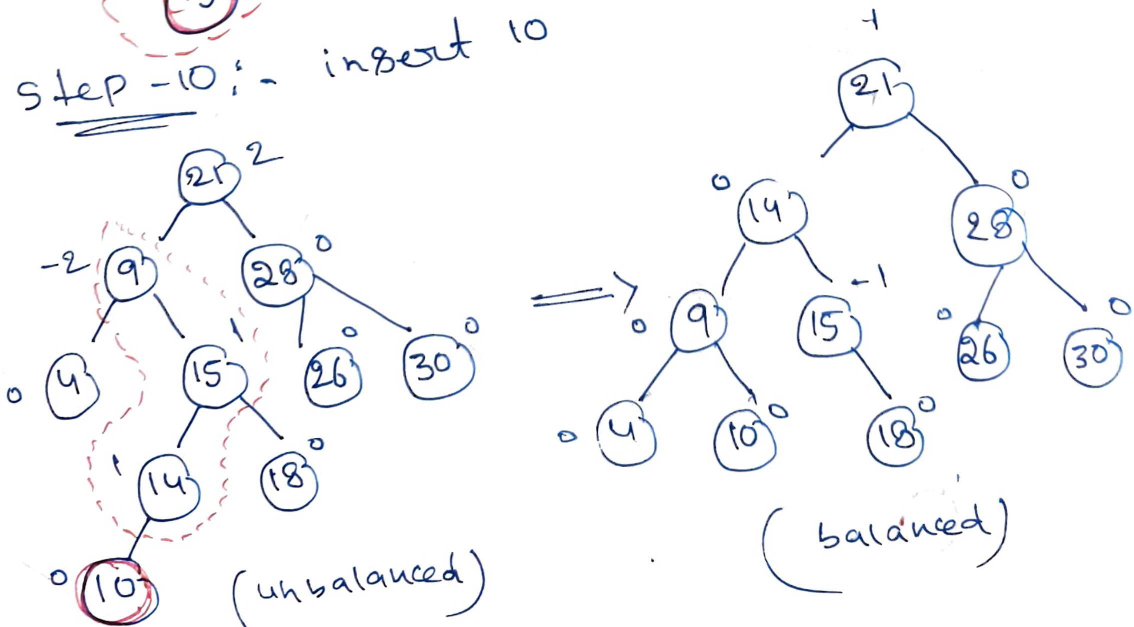


(balanced).

## step-9 :- insert 15



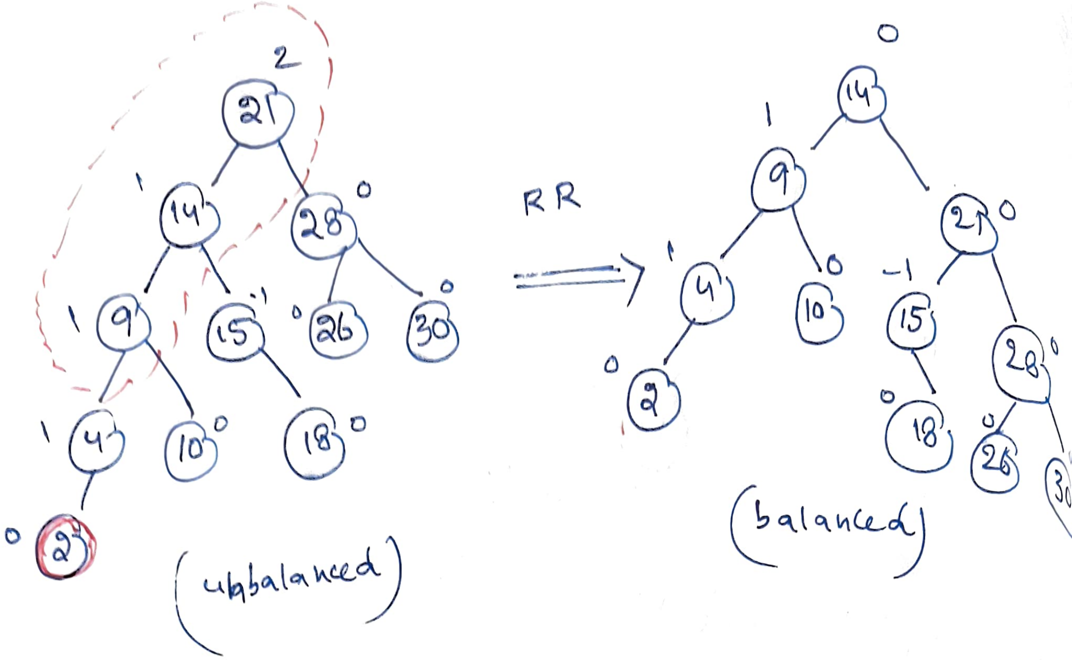

## step-10 :- insert 10



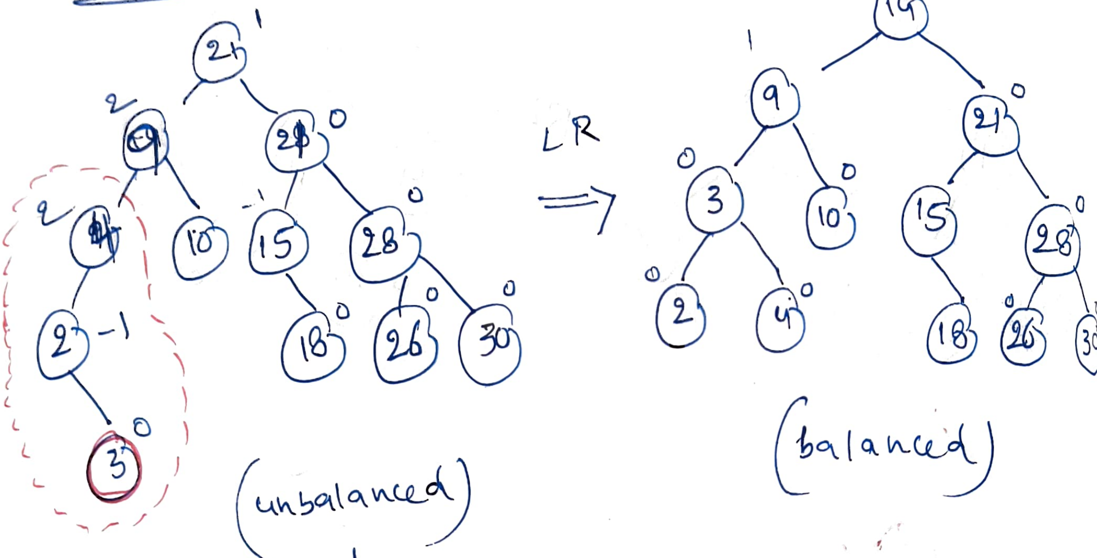
(unbalanced)

(balanced)

## step-11 :- insert 2




## step-12 :- insert 3




## step-13 insert 7