ADITYA ENGINEERING COLLEGE (A)

ADVANCED DATA STRUCTURES

AVL Trees

Before discussing about AVL Tree, we need to discuss about drawbacks of BST.

If BST is not balanced, then the time complexity of all operation of BST is O(n) in worst case

AVL Trees Introduction

- AVL tree is named after its inventors G M Adelson, Velsky and E M Landis in 1962.
- AVL tree is a self-balancing binary search tree in which the heights of the two sub-trees of a node may differ by at most one. Because of this property, AVL tree is also known as a height-balanced tree.
- The advantage of using an AVL tree is that it takes O(log n) time to perform search, insertion and deletion operations in average case as well as worst case i.e O(log n).
- The structure of an AVL tree is same as that of a binary search tree but with a little difference. In its structure, it stores an additional variable called the *BalanceFactor*.

AVL Trees Balance Factor

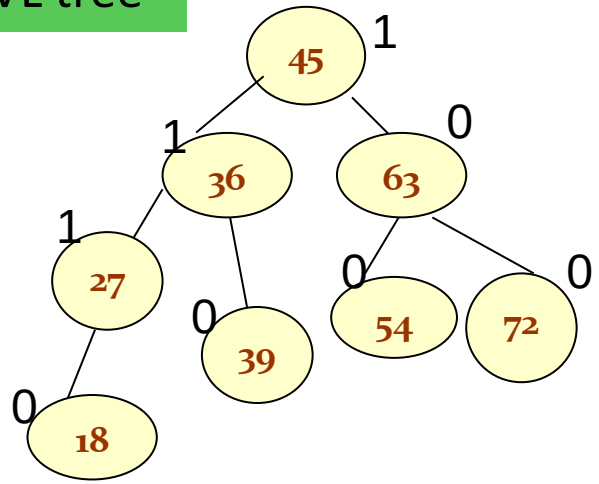
- The balance factor of a node is calculated by subtracting the height of its right sub-tree from the height of its left sub-tree.

  Balance factor = Height (left sub-tree) – Height (right sub-tree)

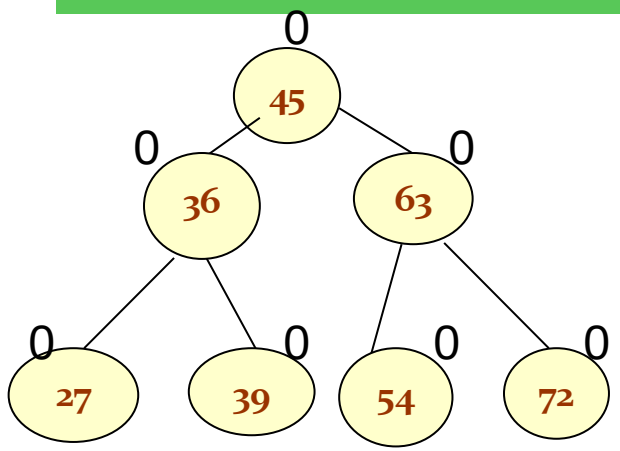
- A binary search tree in which every node has a balance factor of -1, 0 or 1 is said to be height balanced. A node with any other balance factor is considered to be unbalanced and requires rebalancing.
- If the balance factor of a node is 1, then it means that the left sub-tree of the tree is one level higher than that of the right sub-tree. Such a tree is called *Left-heavy tree*.
- If the balance factor of a node is 0, then it means that the height of the left sub-tree is equal to the height of its right sub-tree.
- If the balance factor of a node is -1, then it means that the left sub-tree of the tree is one level lower than that of the right sub-tree. Such a tree is called Right-*heavy tree*.

AVL Trees Balance Factor

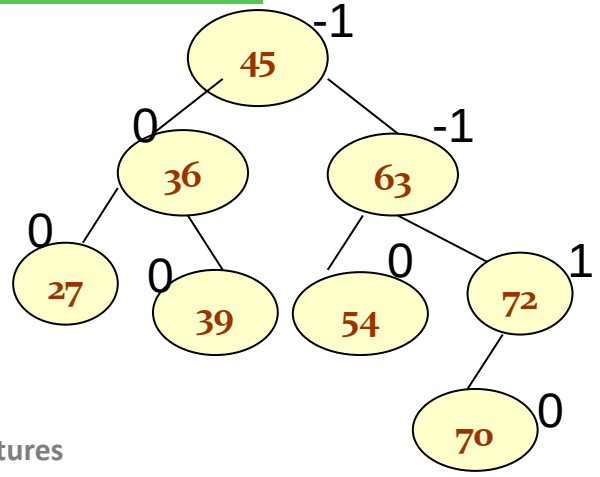
Left heavy AVL tree



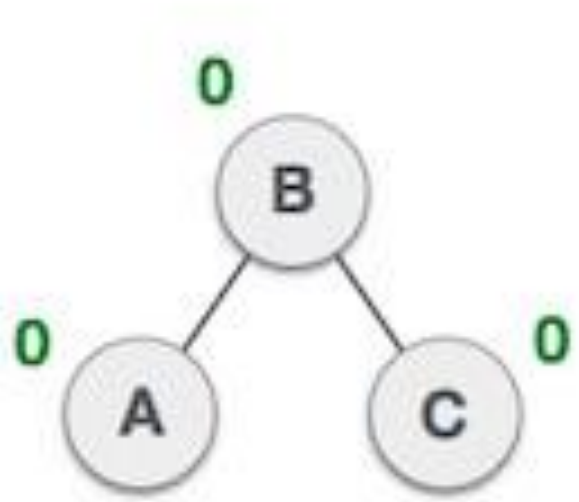
Perfect Balanced AVL tree



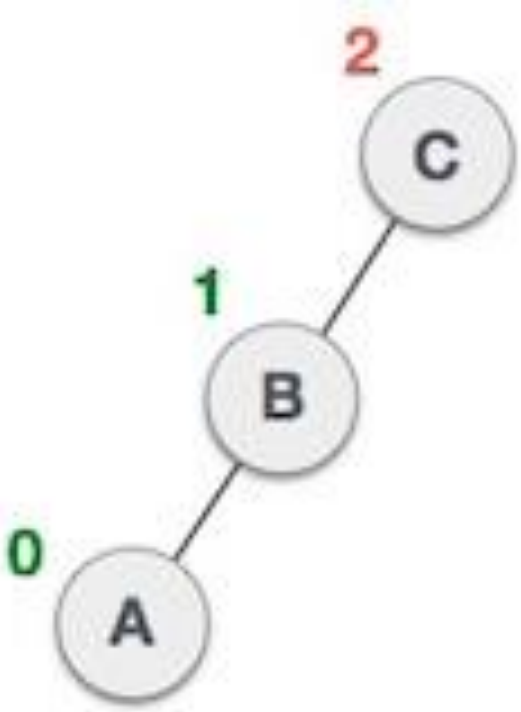
Right heavy AVL tree



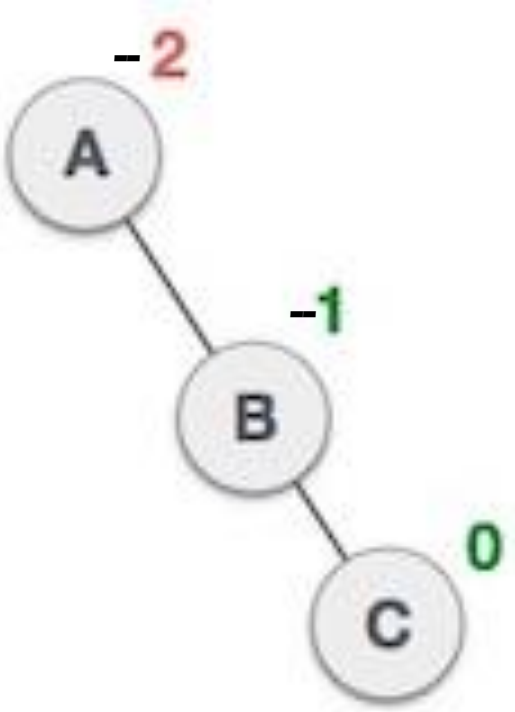
AVL Trees Balance Factor



Balanced



Not balanced



Not balanced

Searching for a Node in an AVL Tree

- Searching in an AVL tree is performed exactly the same way as it is performed in a binary search tree.
- Because of the height-balancing of the tree, the search operation takes O(log *n*) time to complete.
- Since the operation does not modify the structure of the tree, no special provisions need to be taken.

Inserting a Node in an AVL Tree

- Since an AVL tree is also a variant of binary search tree, insertion is also done in the same way as it is done in case of a binary search tree.
- Like in binary search tree, the new node is always inserted as the leaf node. But the step of insertion is usually followed by an additional step of rotation.
- Rotation is done to restore the balance of the tree. However, if insertion of the new node does not disturb the balance factor, that is, if the balance factor of every node is still -1, 0 or 1, then rotations are not needed.

Inserting a Node in an AVL Tree

- During insertion, the new node is inserted as the leaf node, so it will always have balance factor equal to zero.
- The nodes whose balance factors will change are those which lie on the path between the root of the tree and the newly inserted node.
- The possible changes which may take place in any node on the path are as follows:
  - Initially the node was either left or right heavy and after insertion has become balanced.
  - Initially the node was balanced and after insertion has become either left or right heavy.
  - Initially the node was heavy (either left or right) and the new node has been inserted in the heavy sub-tree thereby creating an unbalanced sub-tree. Such a node is said to be a critical node.

Rotations to Balance AVL Trees

- To perform rotation, our first work is to find the *critical node*. Critical node is the nearest ancestor node on the path from the root to the inserted node whose balance factor is neither -1, 0 nor 1.
- The second task is to determine which type of rotation has to be done.
- There are four types of rebalancing rotations and their application depends on the position of the inserted node with reference to the critical node.

➢ ***LL rotation:*** the new node is inserted in the left sub-tree of the left sub-tree of the critical node

➢ ***RR rotation:*** the new node is inserted in the right sub-tree of the right sub-tree of the critical node

➢ ***LR rotation:*** the new node is inserted in the right sub-tree of the left sub-tree of the critical node

➢ ***RL rotation:*** the new node is inserted in the left sub-tree of the right sub-tree of the critical node

Rotations to Balance AVL Trees

- **LL & RR rotations are called single rotations**
- **LR & RL rotations are called double rotations**

- **LR Rotation includes 2 steps**

1)Perform RR on child subtree

2)Perform LL on the entire tree

- **RL Rotation includes 2 steps**

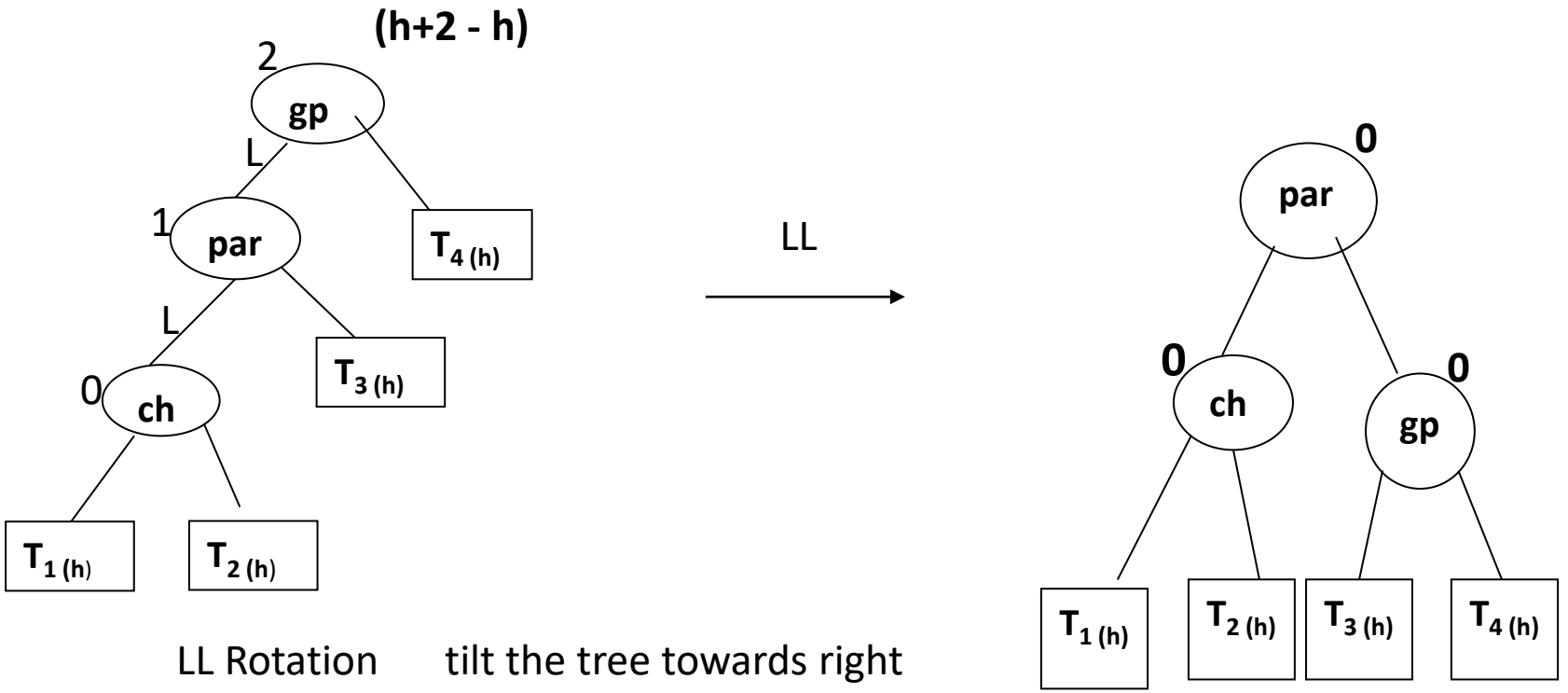
1)Perform LL on child subtree

2)Perform RR on the entire tree

LL Rotation

Let the grandparent (critical code) be represented as gp. Its child is parent (par) and its child is represented as ch in the path of the inserted node to root.

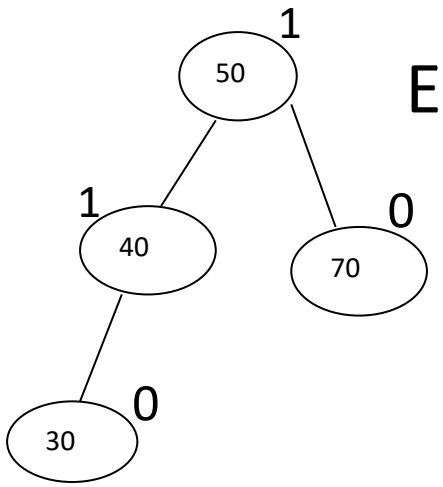
[h is the height of the subtrees T1, T2,T3,T4]



LL Rotation tilt the tree towards right

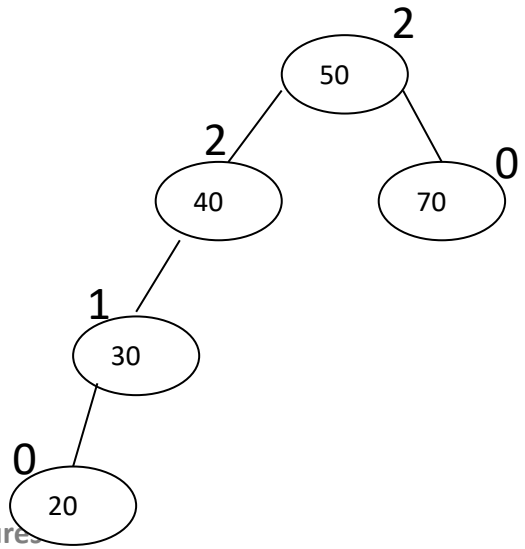
LL Rotation

Example: Insert 20 into the AVL tree

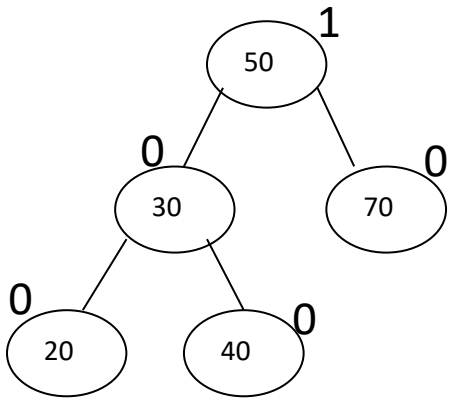


After insertion the tree is

The critical node (gp) is 40, par is 30 and ch=20. Perform LL rotation

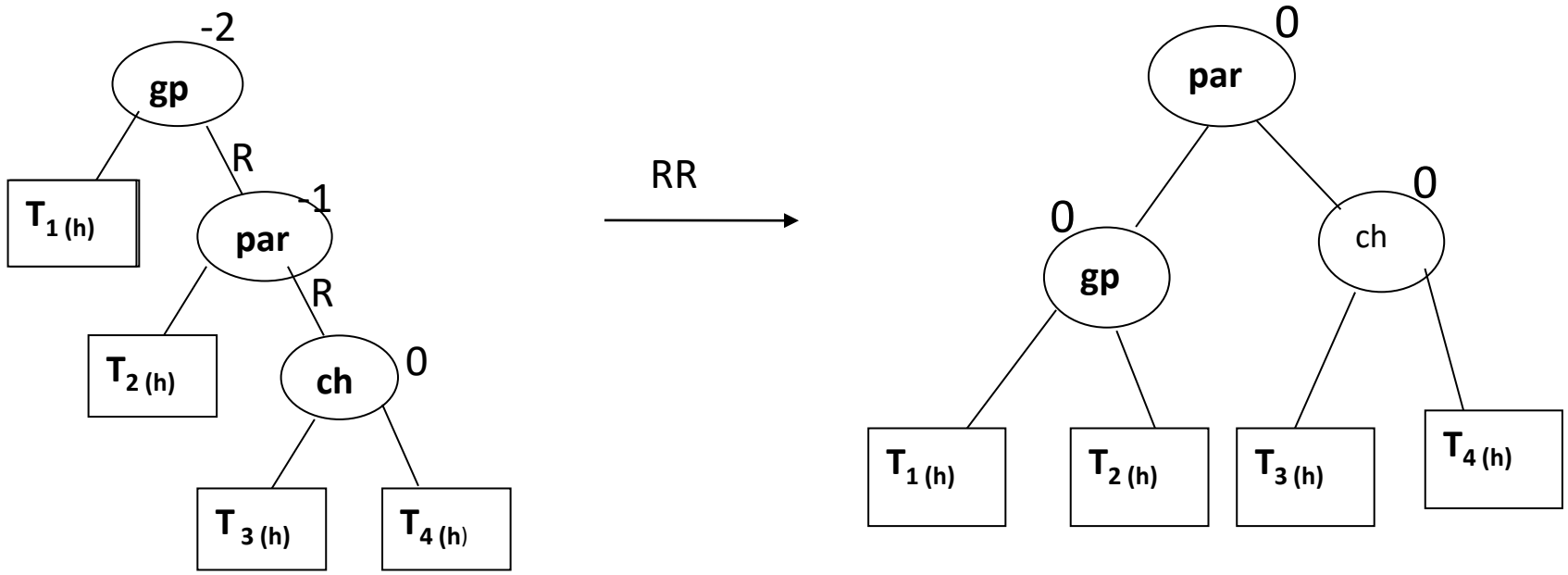


The final AVL tree after LL Rotation is

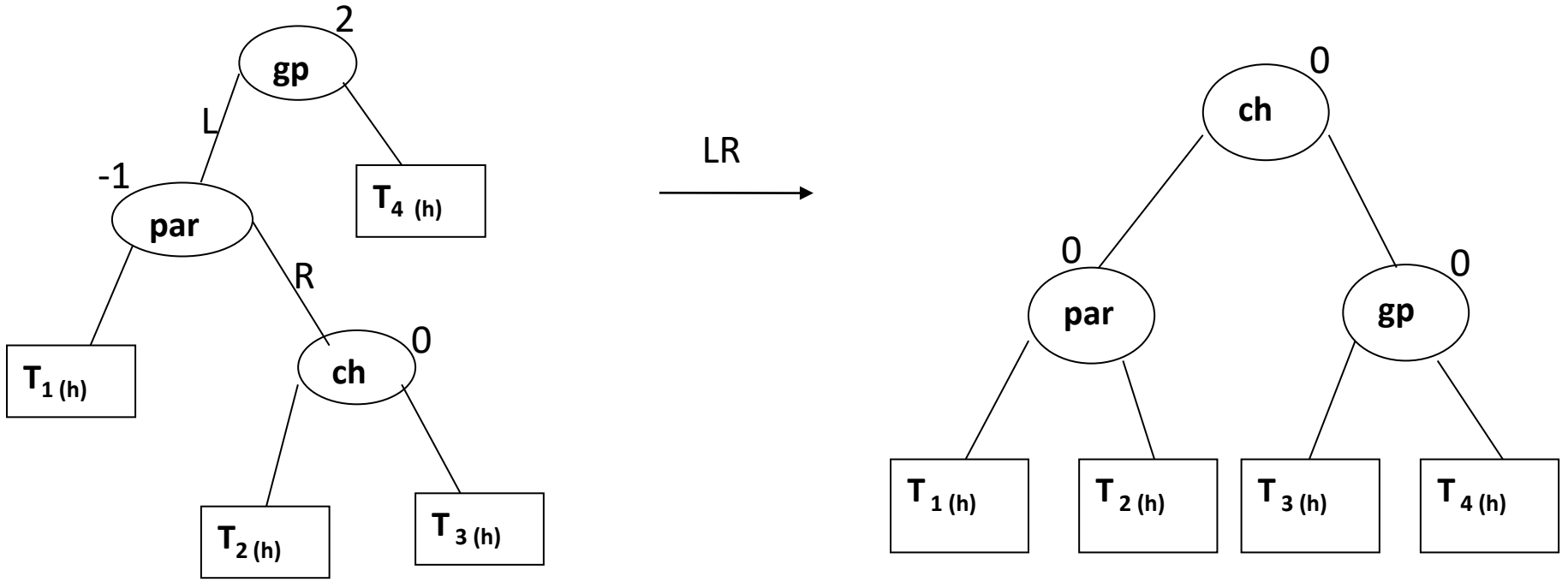


(The tree is Avl balanced)

RR Rotation



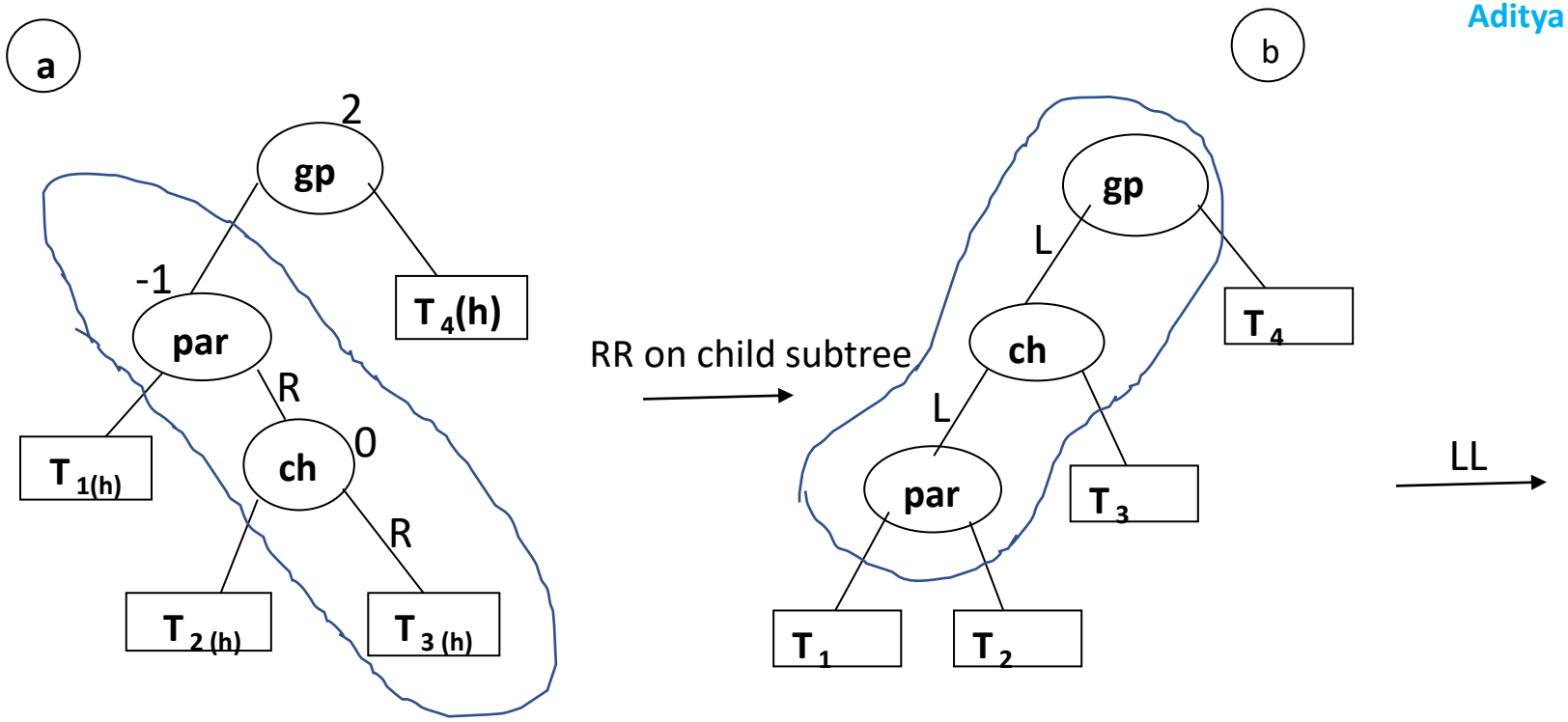
LR Rotation



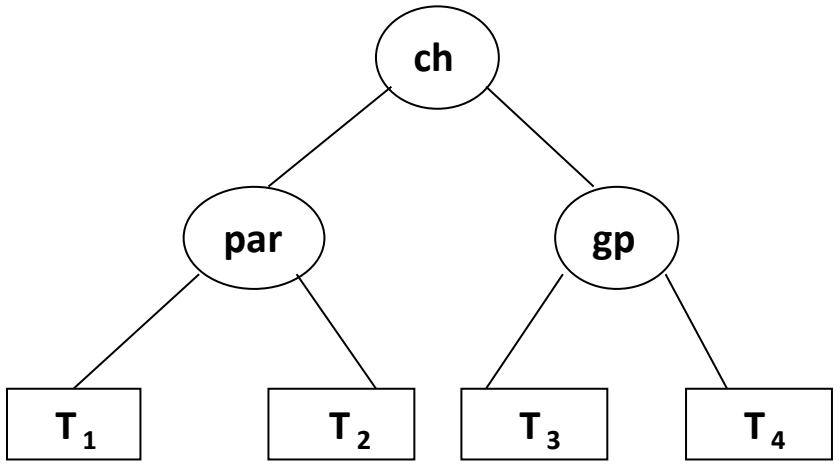
Consider ch moving in between par and gp on top. Therefore par is left and gp is right child.

LR rotation includes 2 steps

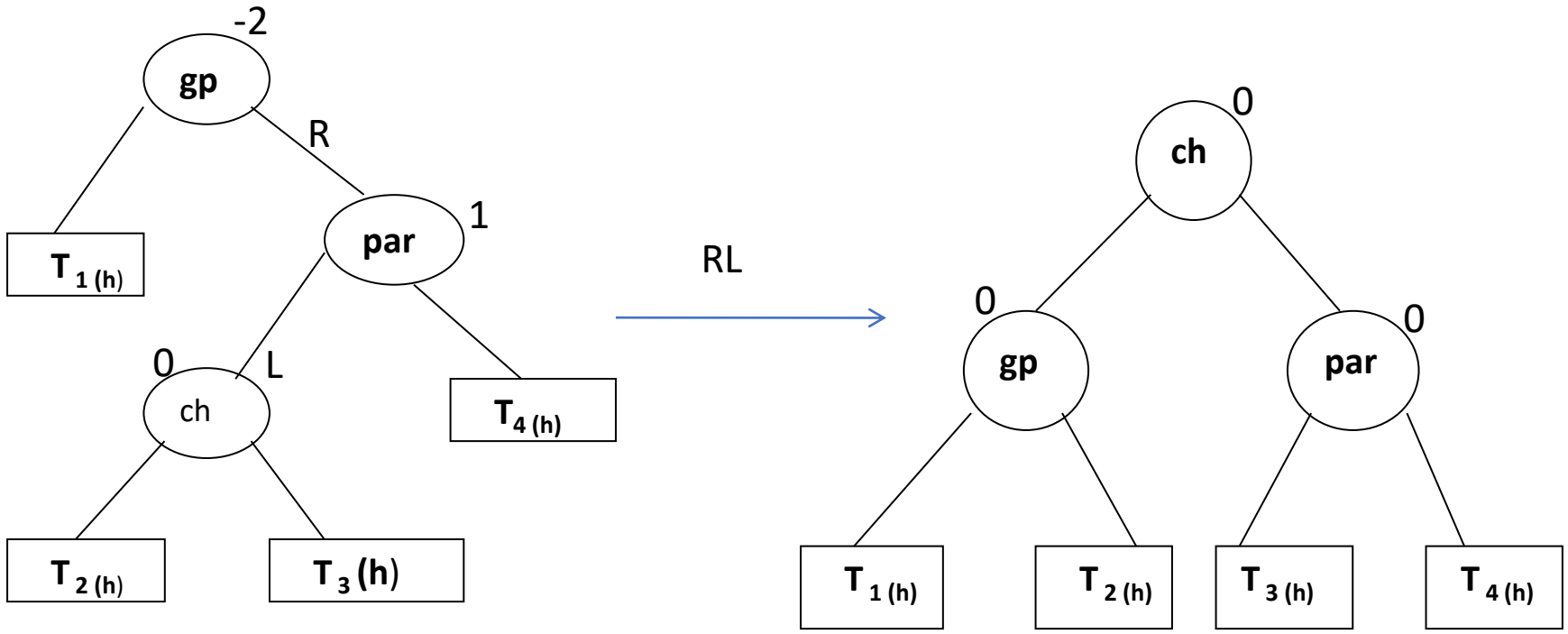
1) Perform RR on child subtree
2) Perform LL on entire tree



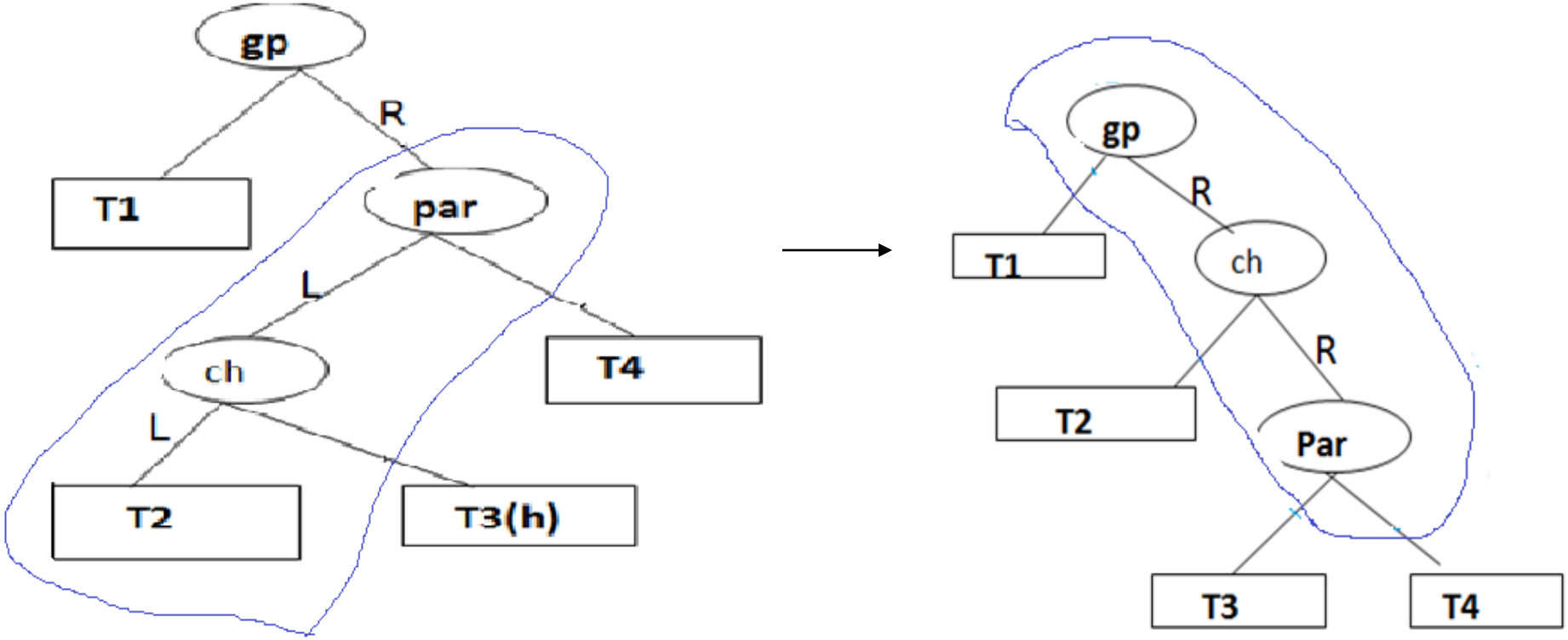
Perform LL on b

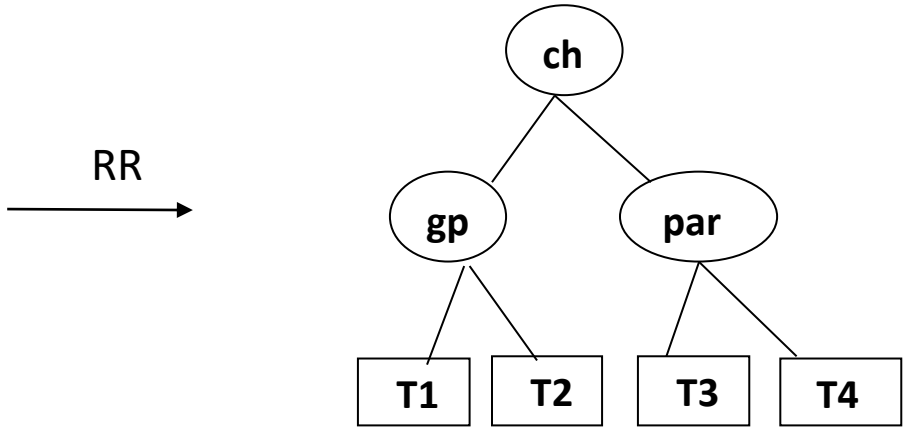


RL Rotation

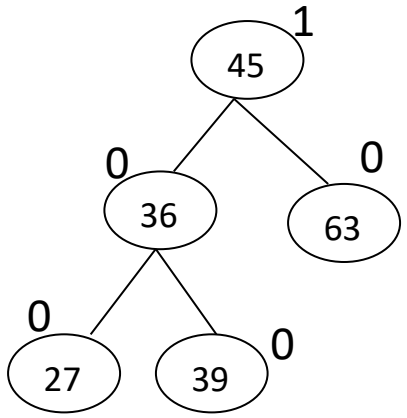


RL includes performing LL on child subtree & RR on entire tree

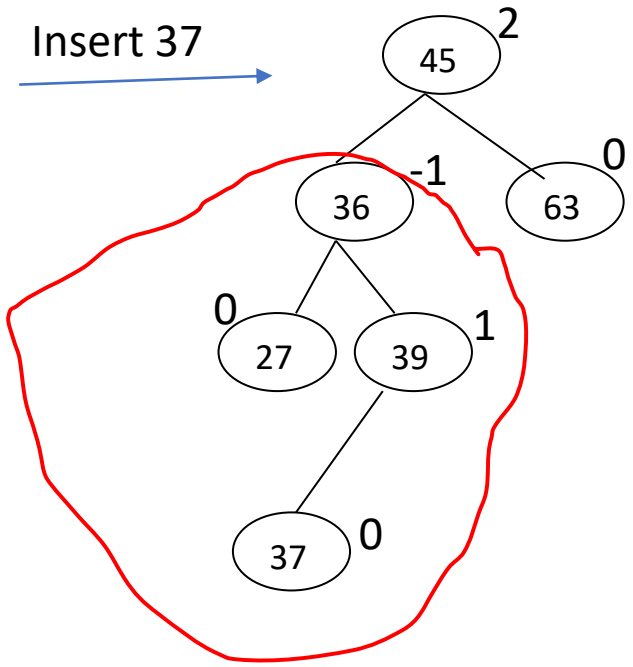

gp
R
T1
par
L
ch
T4
L
T2
T3(h)
gp
R
T1
ch
R
T2
Par
T3
T4


RR
ch
gp
par
T1
T2
T3
T4

Example: Perform LR rotation by inserting 37

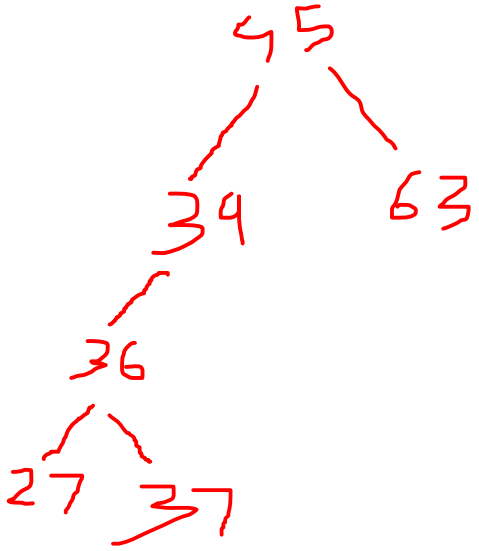


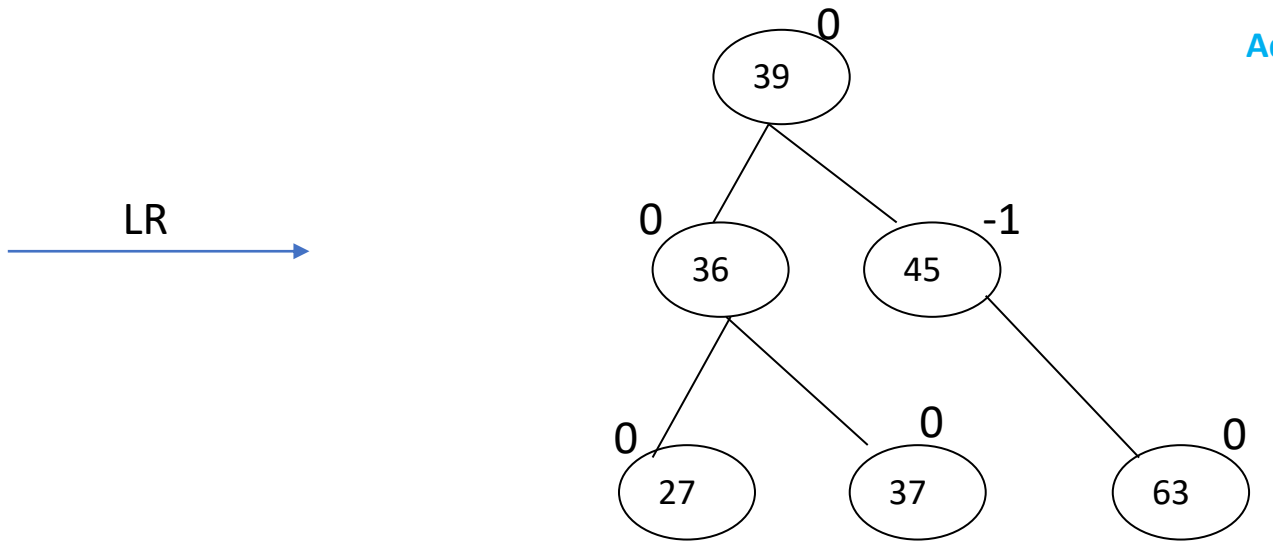
Insert 37



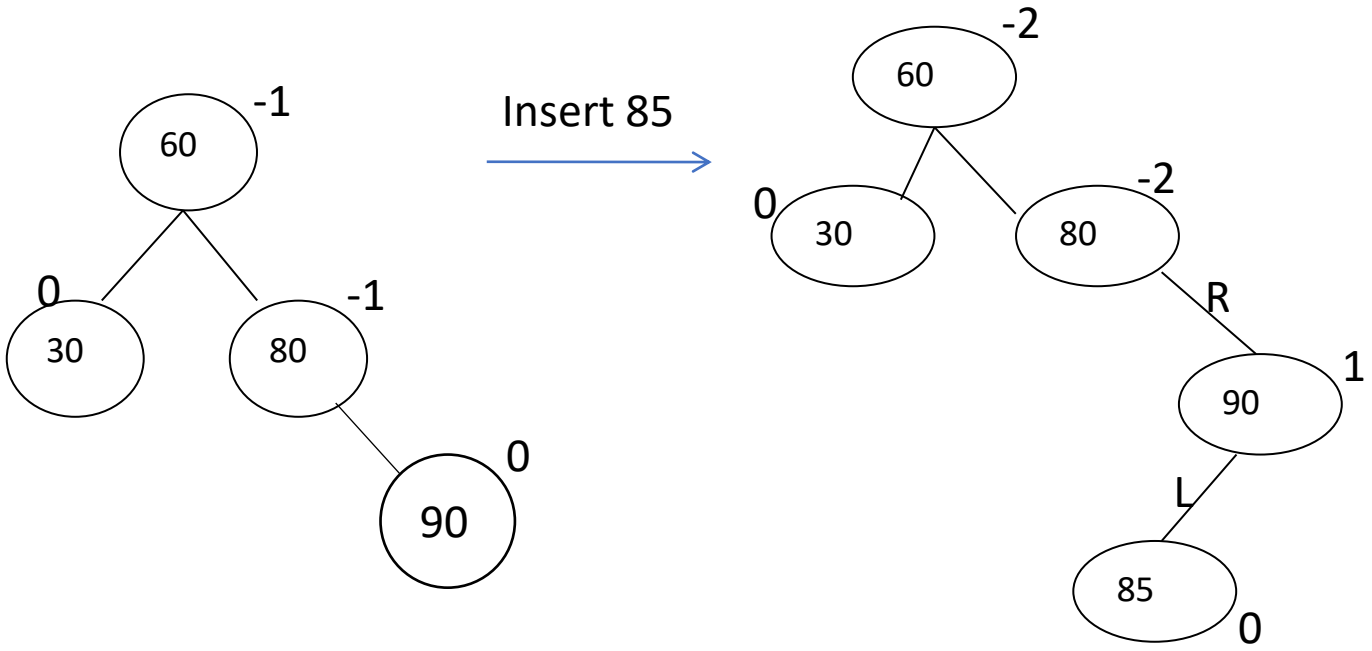
Imbalance at 45

∴ gp=45
par=36
ch=39
Now Perform LR



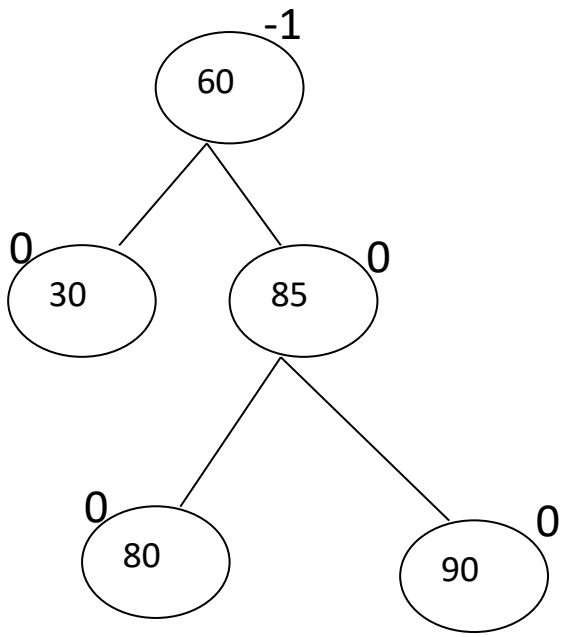


Example: Perform RL by inserting 85 into the tree



gp=80
par=90
ch=85

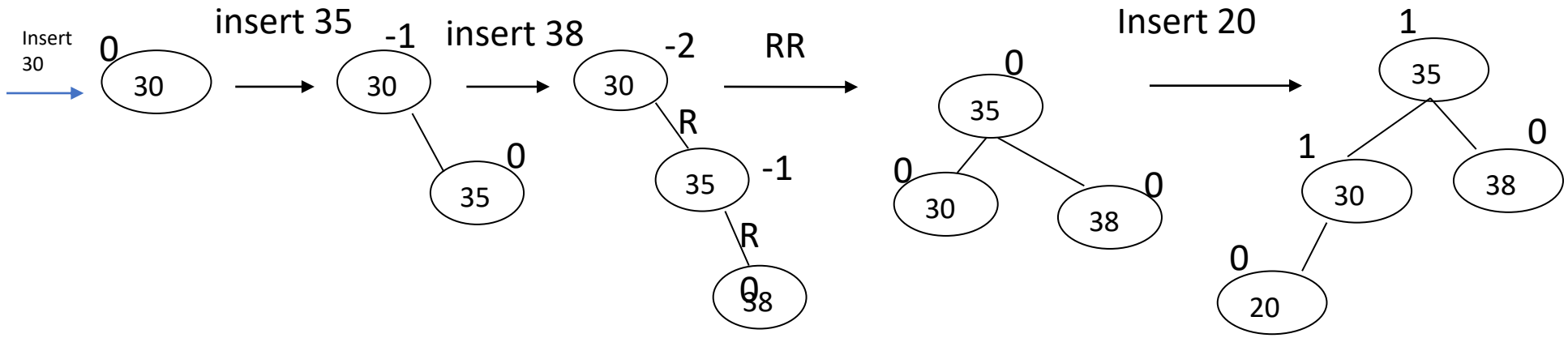
Perform RL

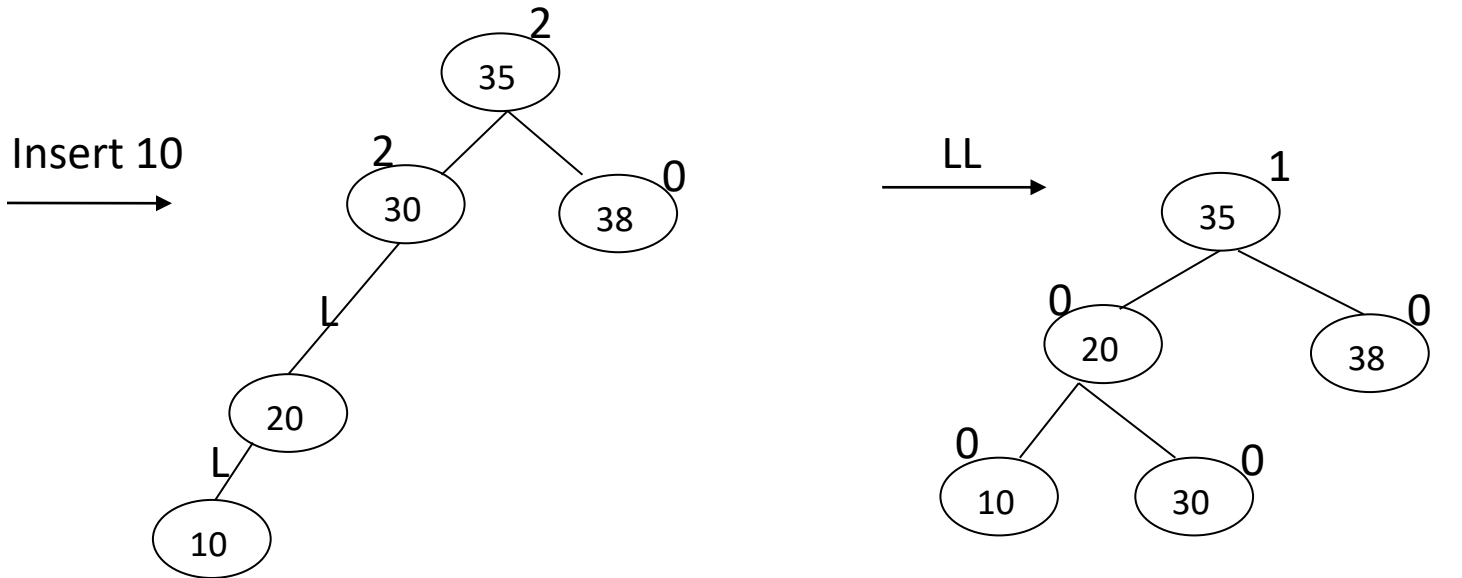


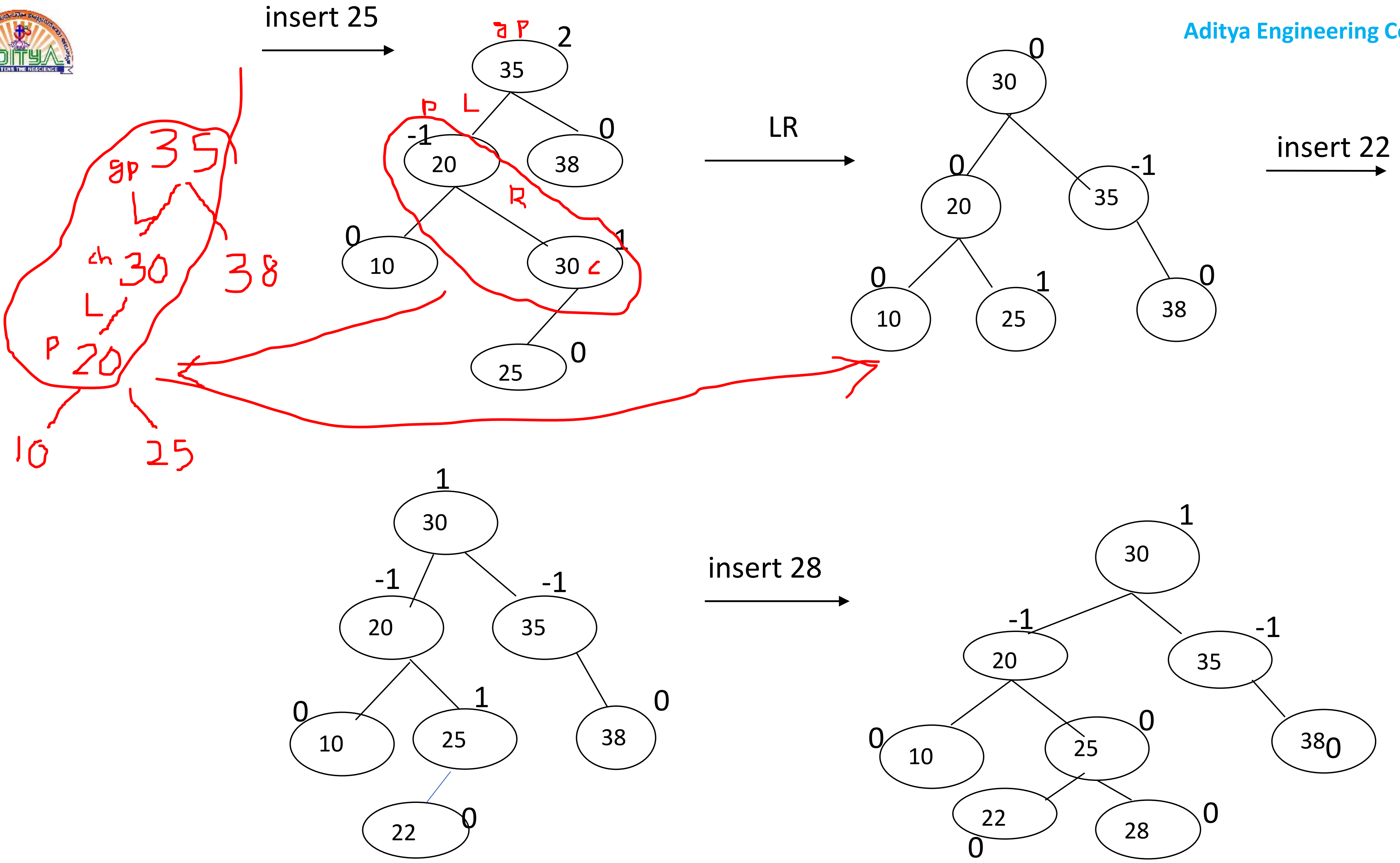
Resultant tree is AVL Tree

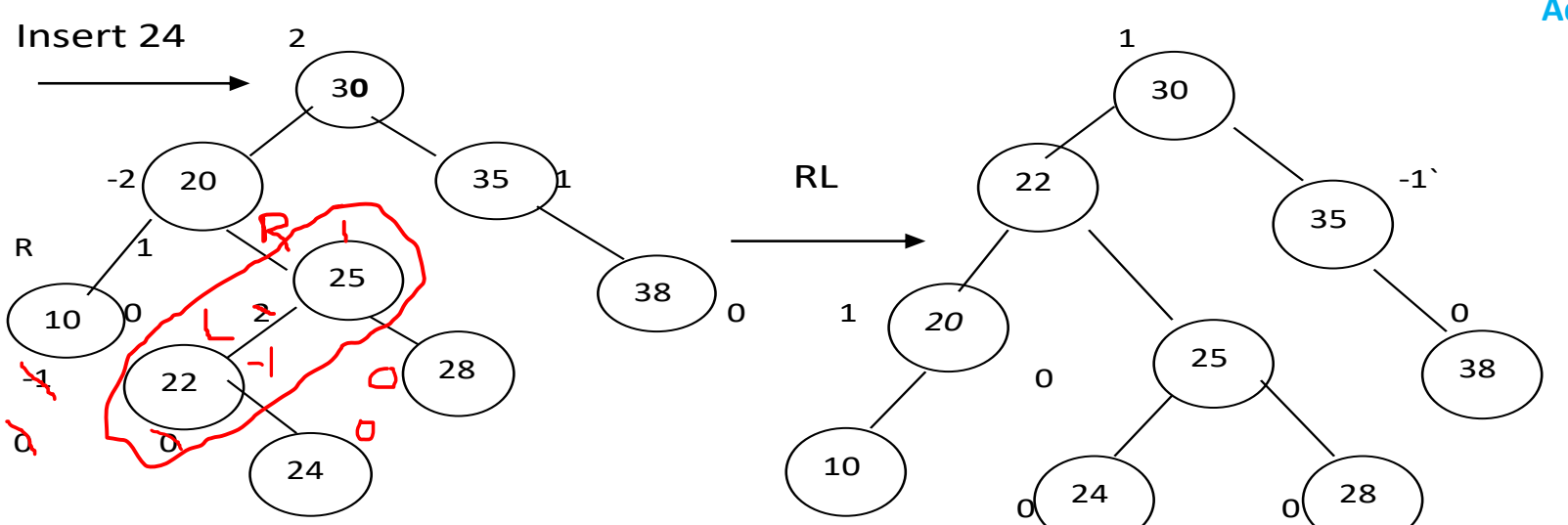
Example: Construct AVL tree with values 30,35,38, 20, 10, 25, 22, 28, 24, 29, 40, 50

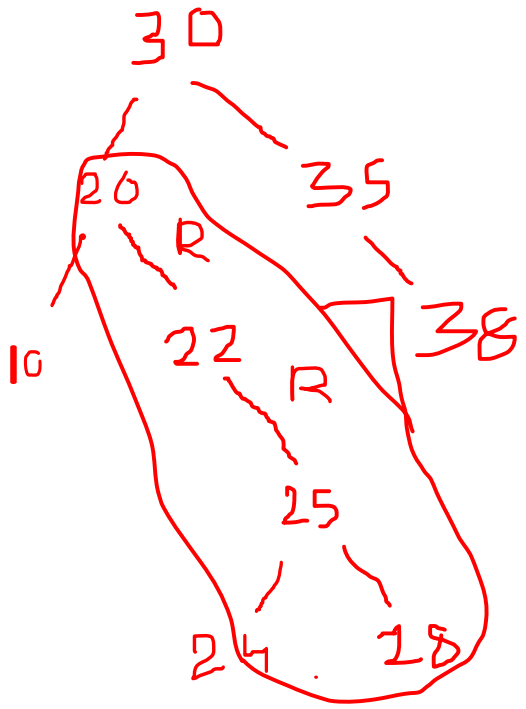
Solution

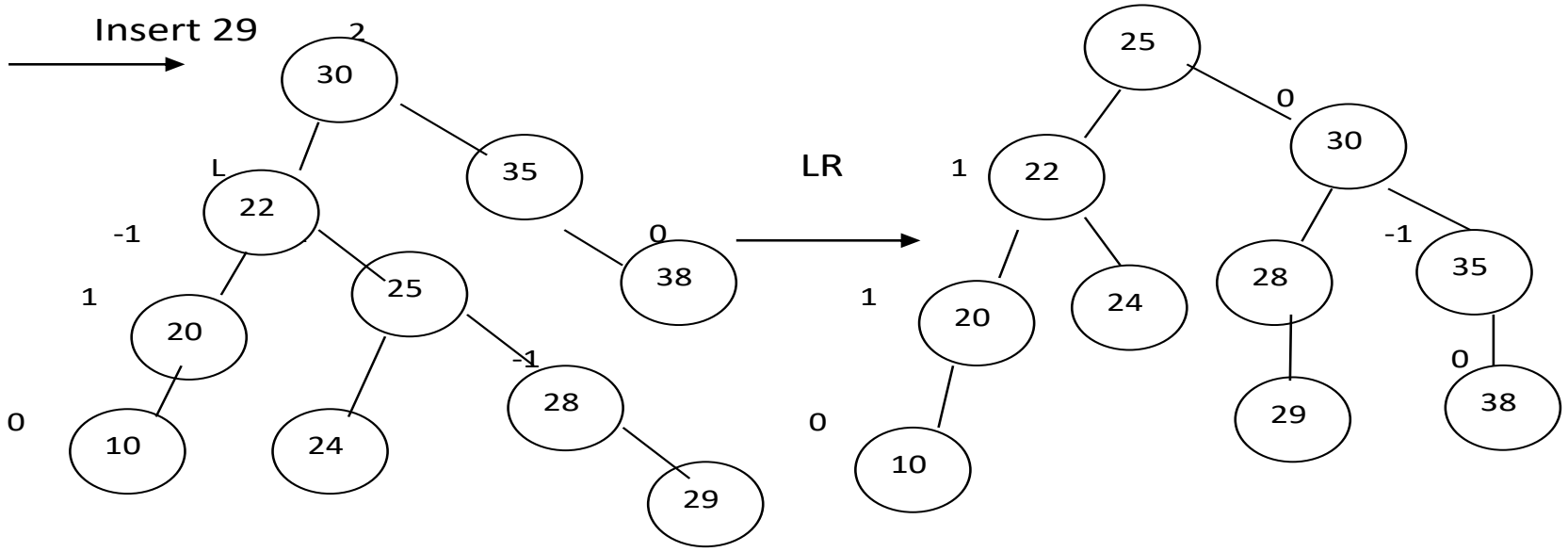


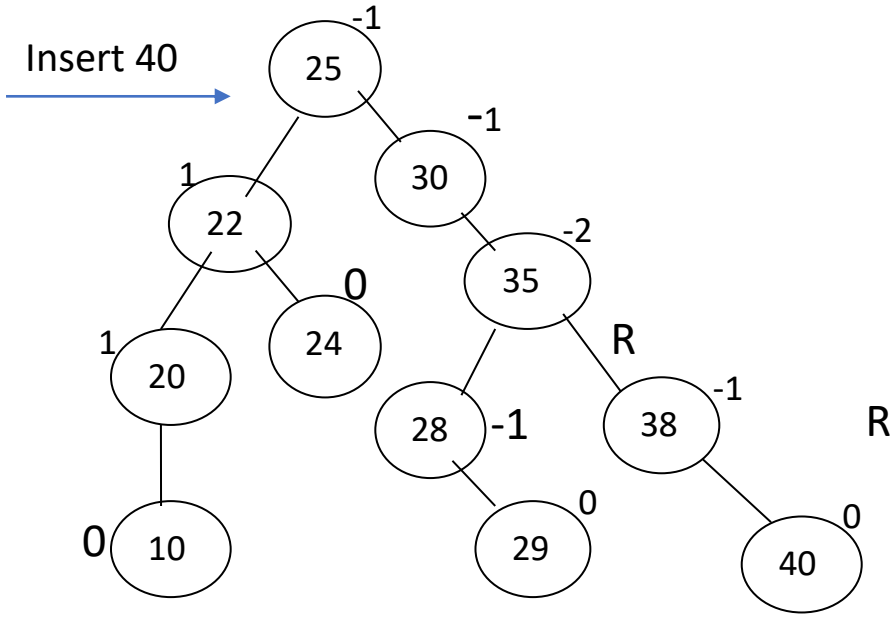


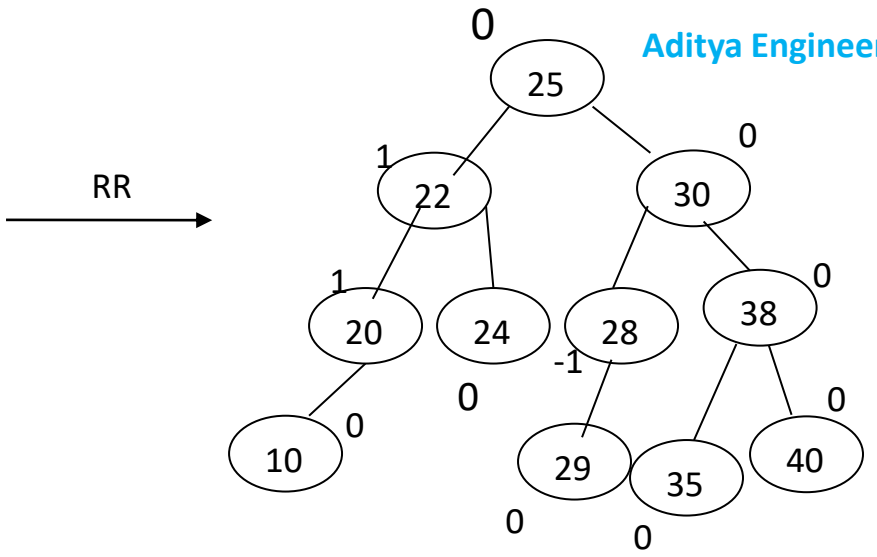

insert 25
LR
insert 22
insert 28
35
20
38
10
30
25
22
28

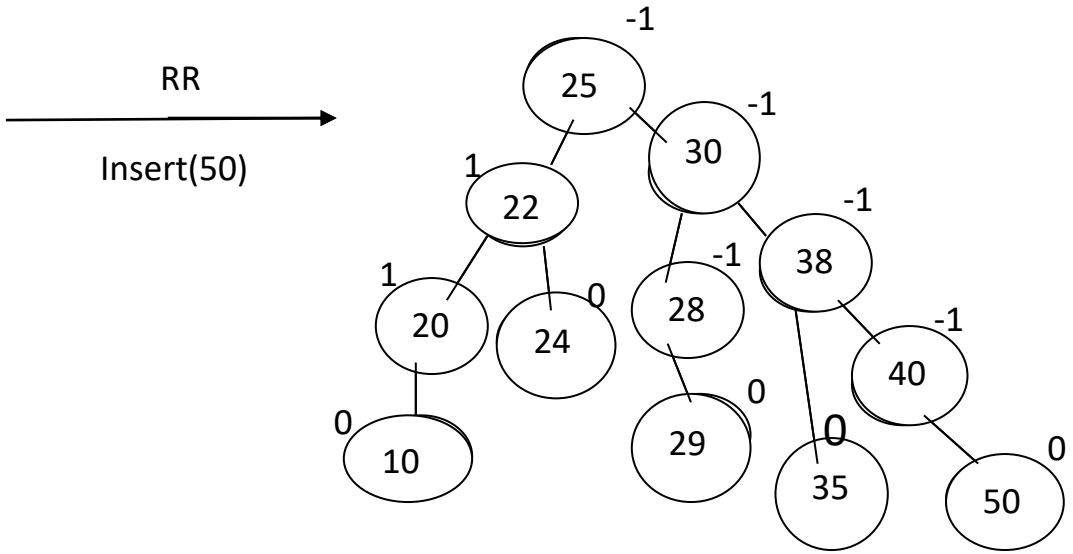

Insert 24
30
20
35
10
25
38
22
28
24
RL
30
22
35
20
25
38
10
24
28

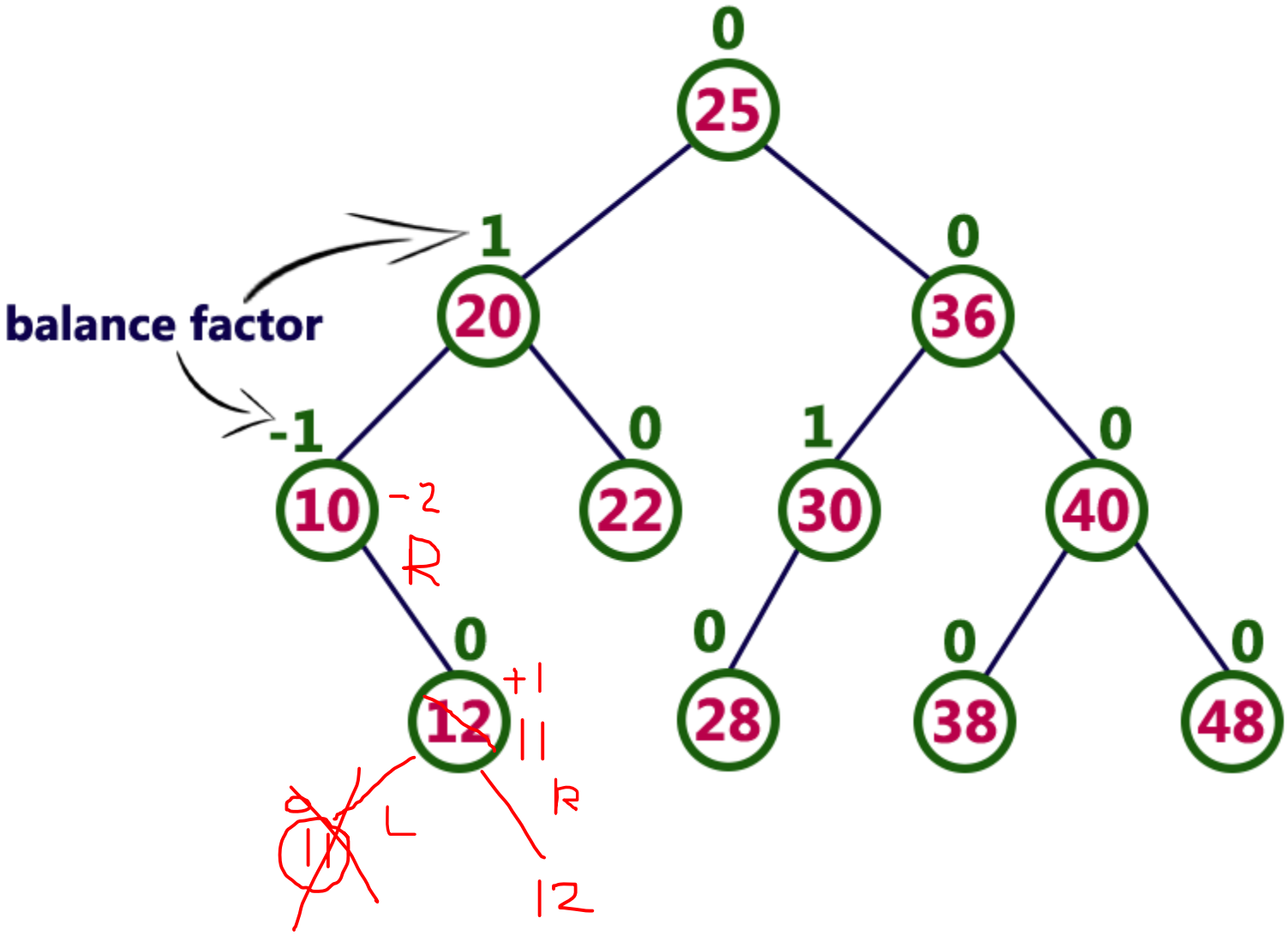

30
20
35
R
38
10
22
R
25
24
28


Insert 29
30
22
35
20
25
38
10
24
28
29
LR
25
22
30
20
24
28
35
10
29
38


Insert 40
25
-1
22
1
30
-1
20
1
24
0
35
-2
10
0
28
-1
29
0
R
38
-1
40
0
R

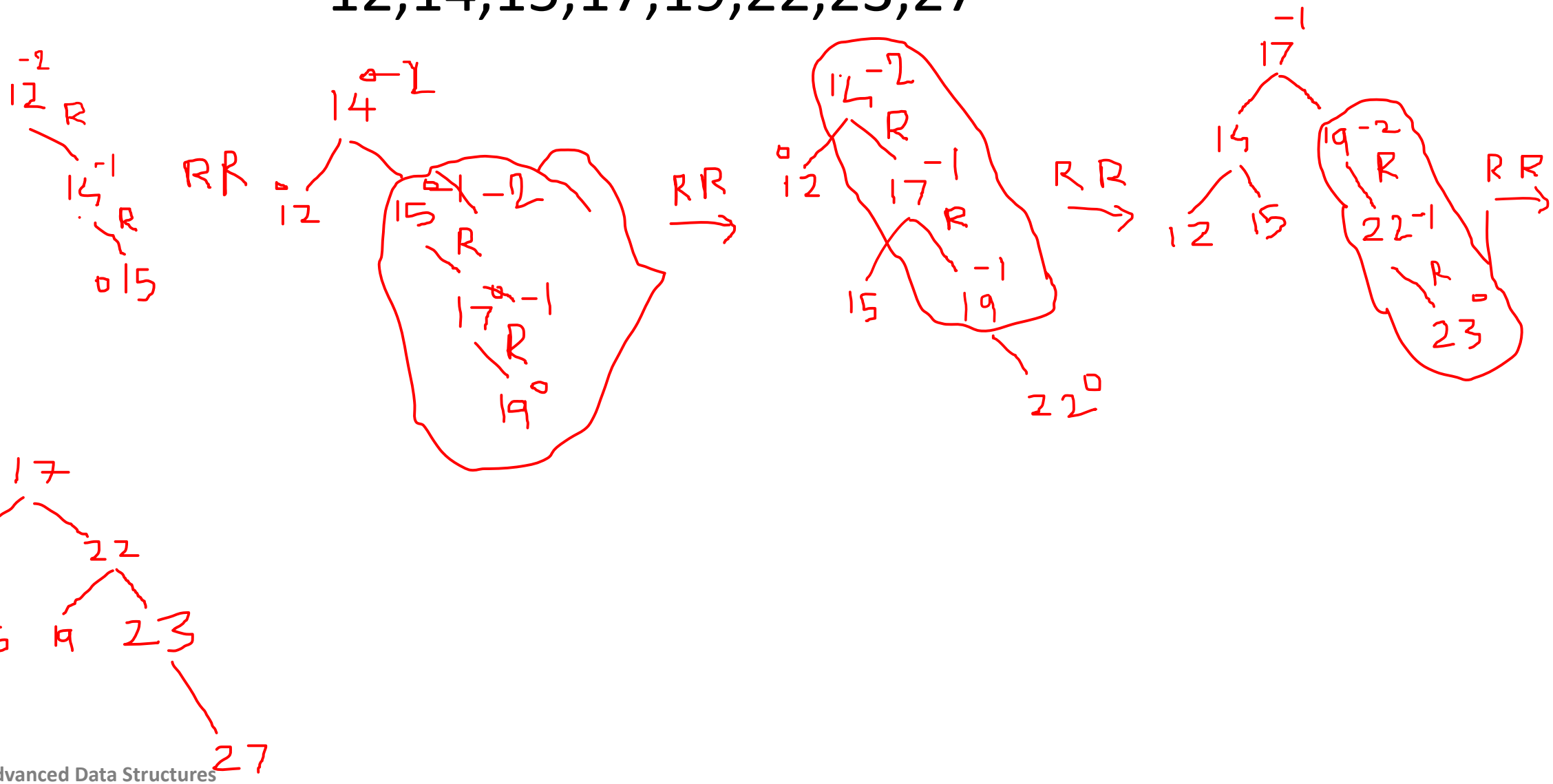

RR
0
25
1
22
0
30
1
20
24
0
28
-1
38
0
10
0
29
0
35
0
40
0


RR
Insert(50)
-1
25
-1
30
1
22
-1
38
1
20
0
24
-1
28
-1
40
0
10
0
29
0
35
0
50
0

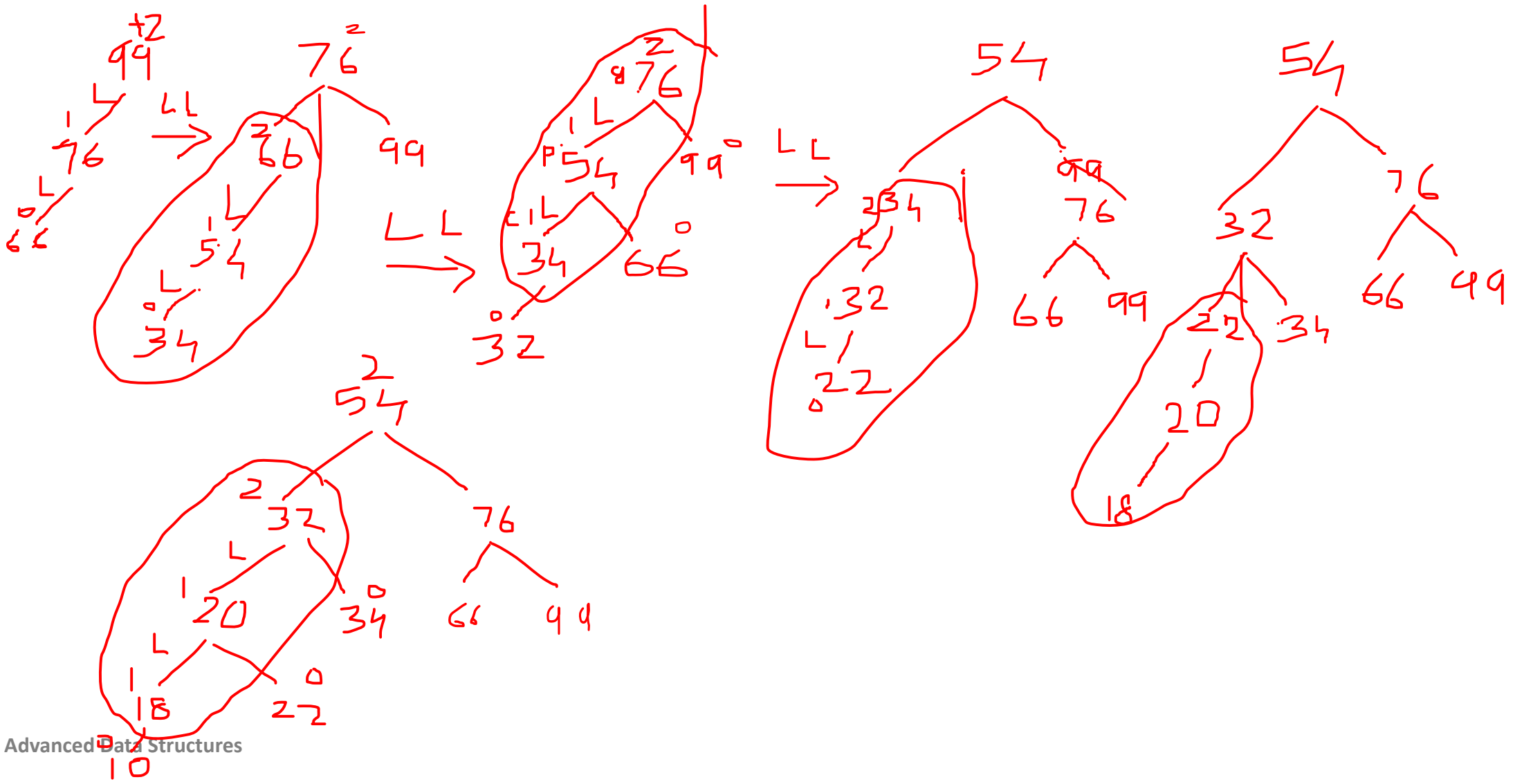

0
25
1
20
0
36
balance factor
-1
10
-2
R
0
22
1
30
0
40
0
12
+1
11
R
0
28
0
38
0
48
L
12

Construct AVL Tree for given values

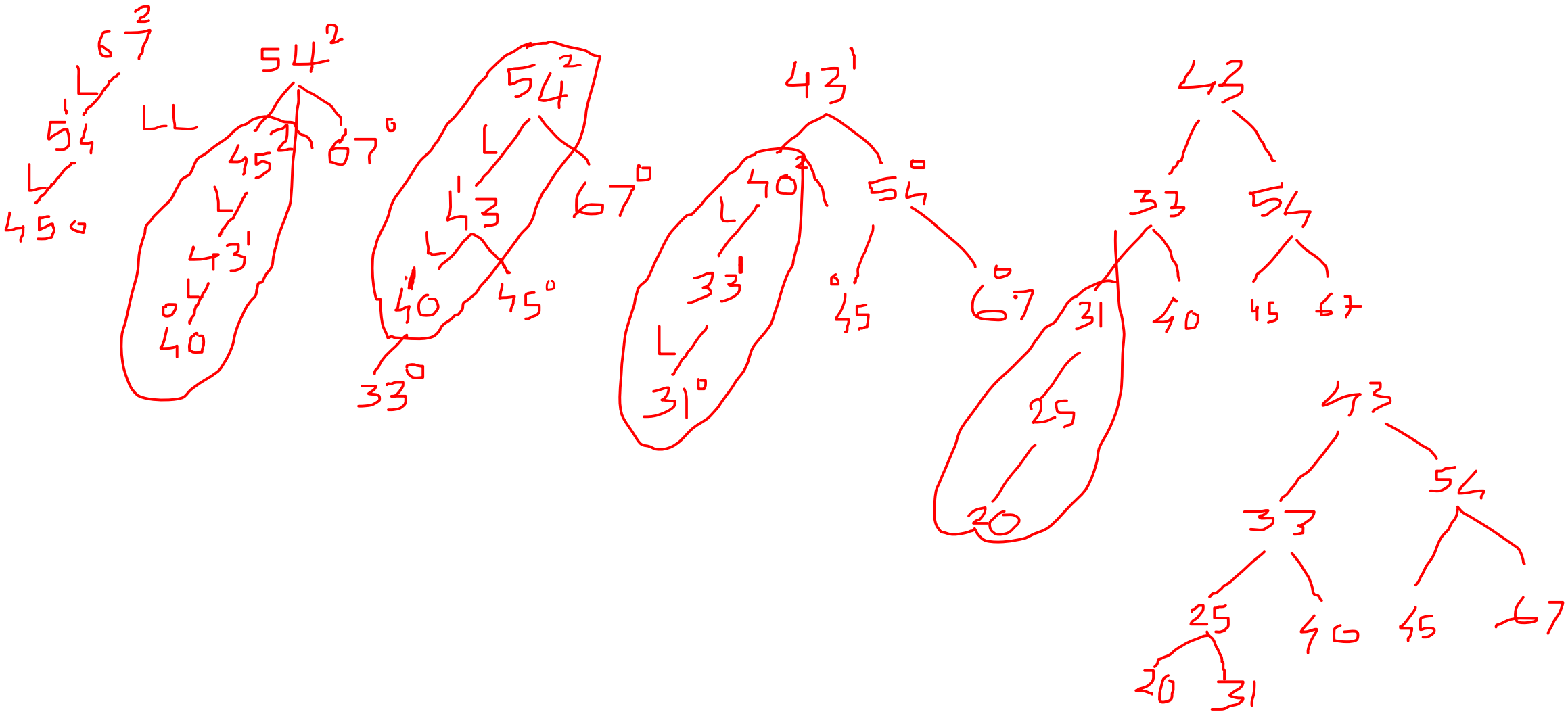
12,14,15,17,19,22,23,27



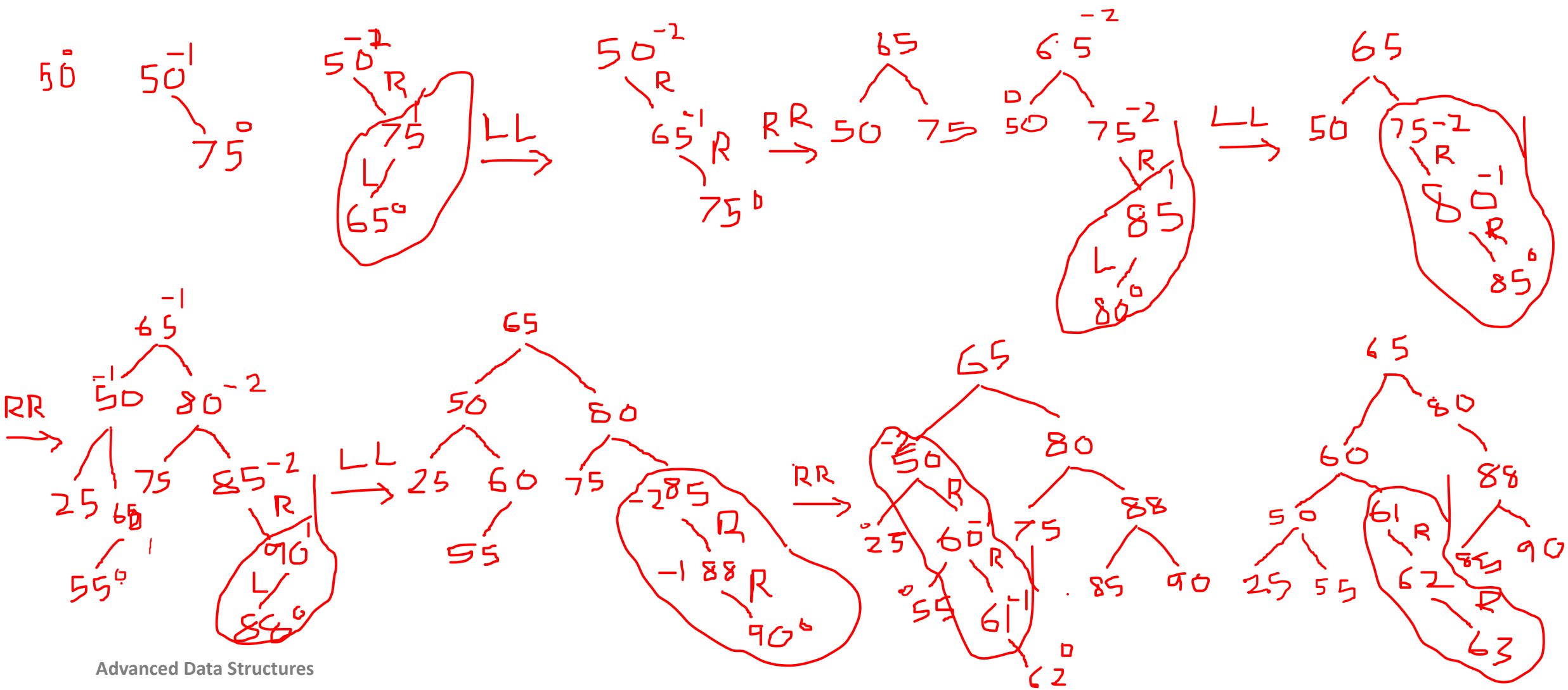
Construct AVL Tree for given values 99,76,66,54,34,32,22,20,18 and 10



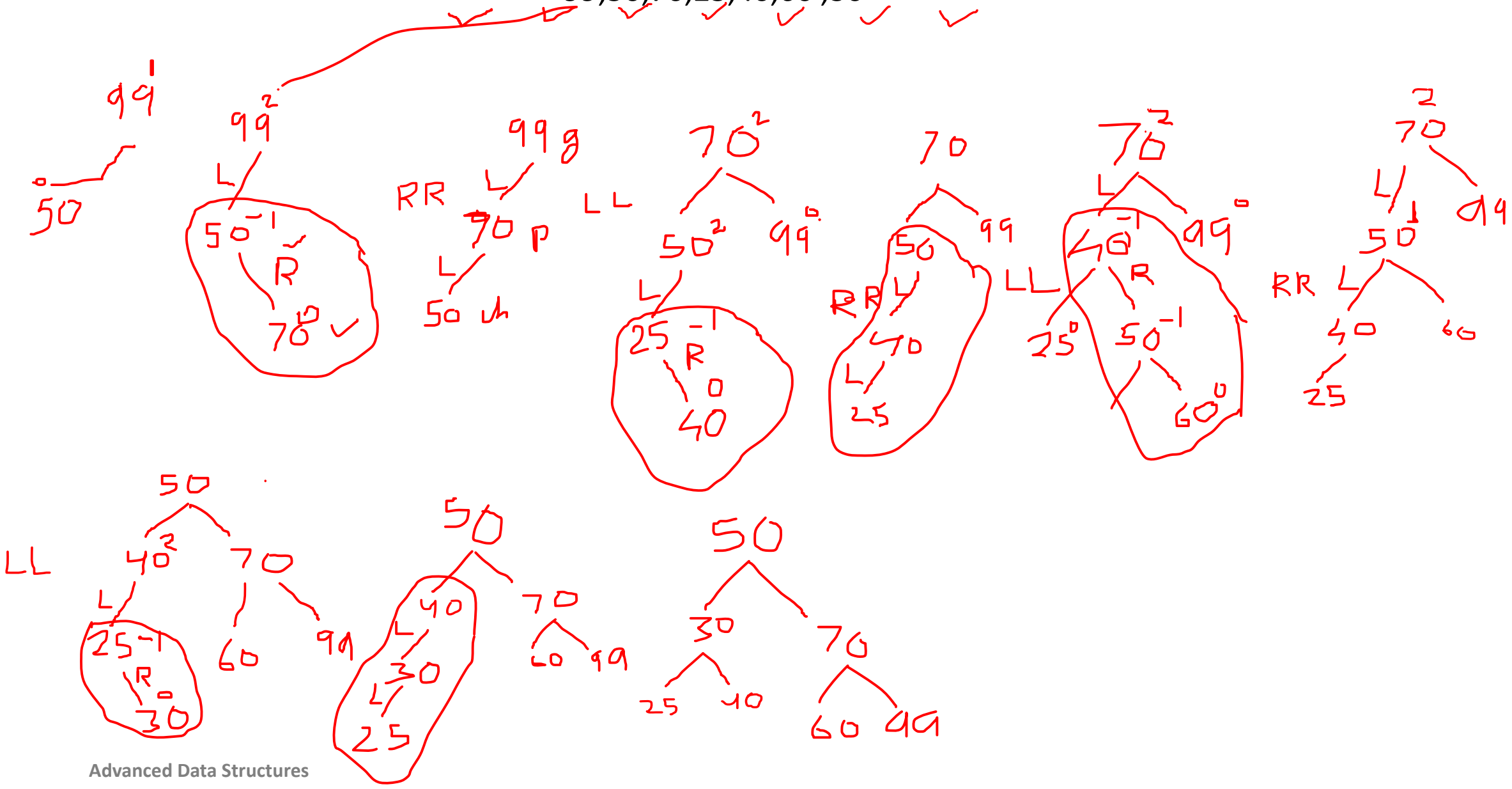
Construct AVL Tree for given values 67, 54, 45, 43, 40, 33, 31, 25 and 20



Construct AVL Tree for given values
50,75,65,85,80,25,60,55,90 and 88



Construct AVL Tree for given values
99,50,70,25,40,60 ,30



Construct AVL Tree for given values 30,10,15,20,35,45,12,13,14,11 and 90

AVL Tree Deletion Operation

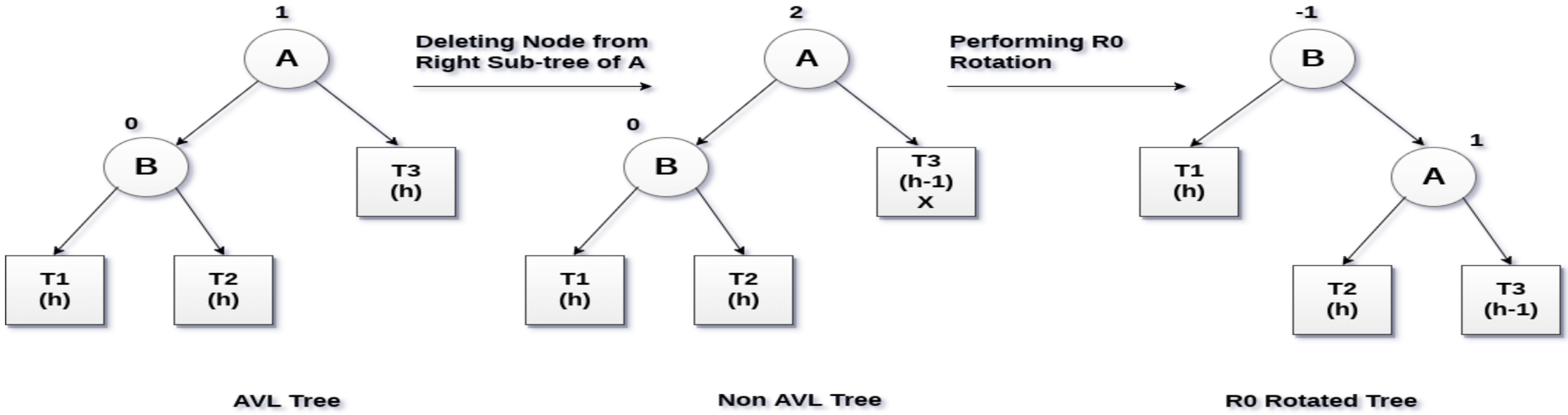
- Deleting a node from an AVL tree is similar to that in a binary search tree. Deletion may disturb the balance factor of an AVL tree and therefore the tree needs to be rebalanced in order to maintain the AVL tree. For this purpose, we need to perform rotations. The two types of rotations **are L rotation and R rotation**. Here, we will discuss R rotations. L rotations are the mirror images of them.
- If the node which is to be deleted is present in the left sub-tree of the critical node, then L rotation needs to be applied else if, the node which is to be deleted is present in the right sub-tree of the critical node, the R rotation will be applied.
- Let us consider that, A is the critical node and B is the root node of its left sub-tree. If node X, present in the right sub-tree of A, is to be deleted, then there can be three different situations:

AVL Tree Deletion Operation

R0 rotation (Node B has balance factor 0)

If the node B has 0 balance factor, and the balance factor of node A disturbed upon deleting the node X, then the tree will be rebalanced by rotating tree using R0 rotation.

The critical node A is moved to its right and the node B becomes the root of the tree with T1 as its left sub-tree. The sub-trees T2 and T3 becomes the left and right sub-tree of the node A. the process involved in R0 rotation is shown in the following image.

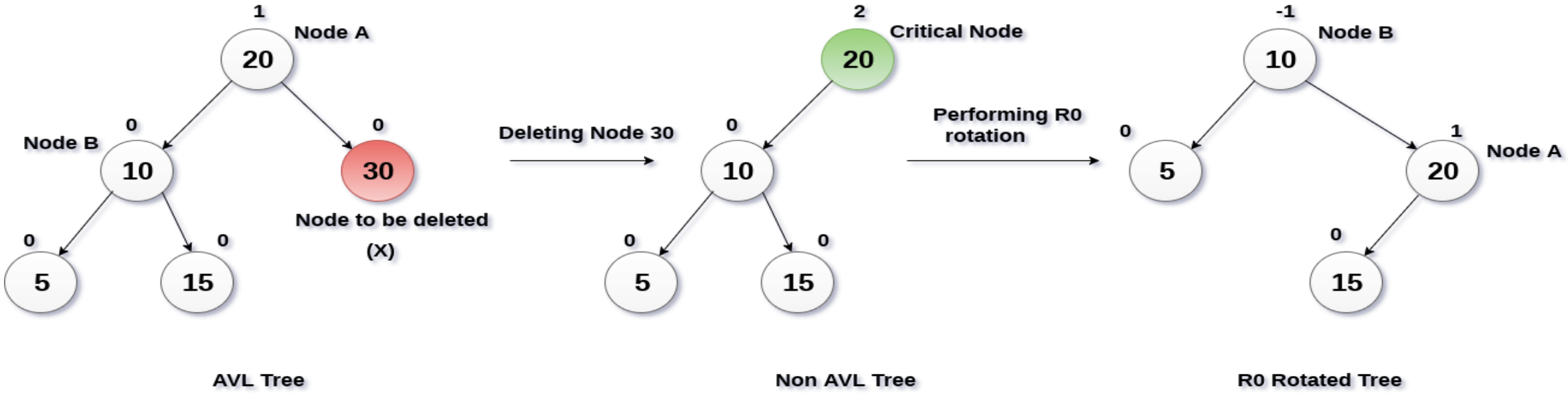


AVL Tree Deletion Operation

Example:

Delete the node 30 from the AVL tree shown in the following image.

In this case, the node B has balance factor 0, therefore the tree will be rotated by using R0 rotation as shown in the following image. The node B(10) becomes the root, while the node A is moved to its right. The right child of node B will now become the left child of node A.

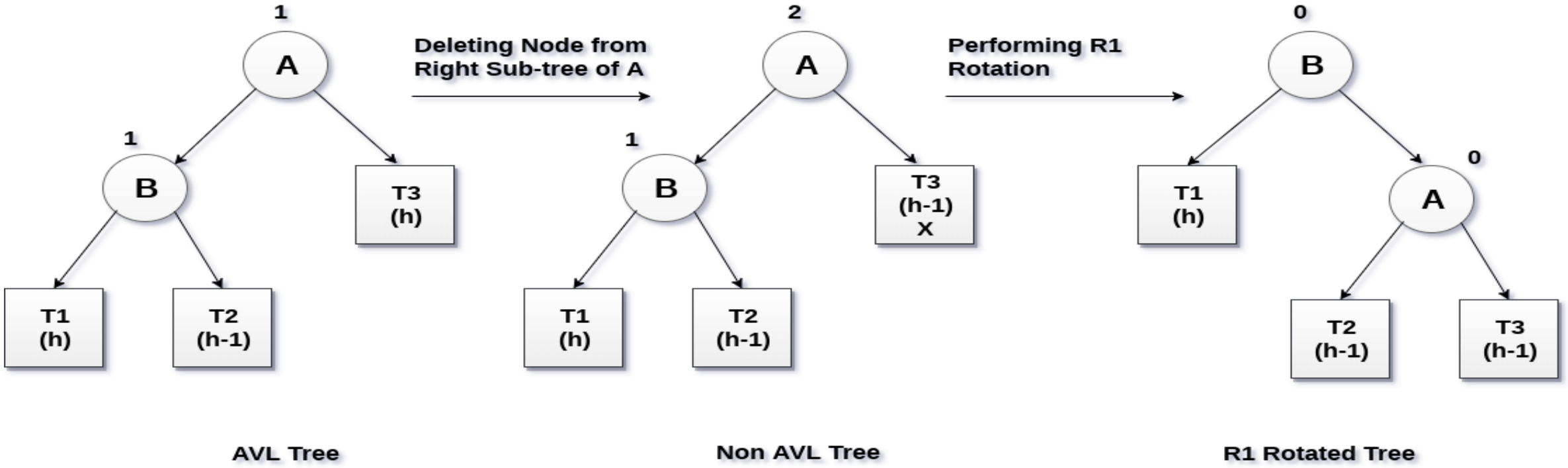


AVL Tree Deletion Operation

R1 Rotation (Node B has balance factor 1)

R1 Rotation is to be performed if the balance factor of Node B is 1. In R1 rotation, the critical node A is moved to its right having sub-trees T2 and T3 as its left and right child respectively. T1 is to be placed as the left sub-tree of the node B.

The process involved in R1 rotation is shown in the following image.

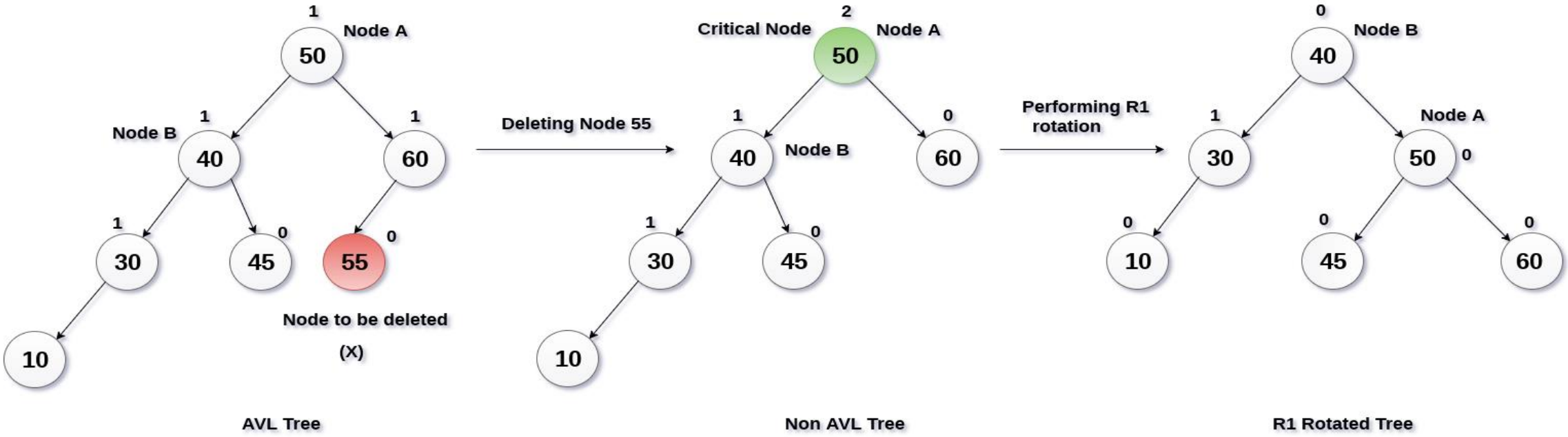


AVL Tree Deletion Operation

Example:

Deleting 55 from the AVL Tree disturbs the balance factor of the node 50 i.e. node A which becomes the critical node. This is the condition of R1 rotation in which, the node A will be moved to its right (shown in the image below). The right of B is now become the left of A (i.e. 45).

The process involved in the solution is shown in the following image.

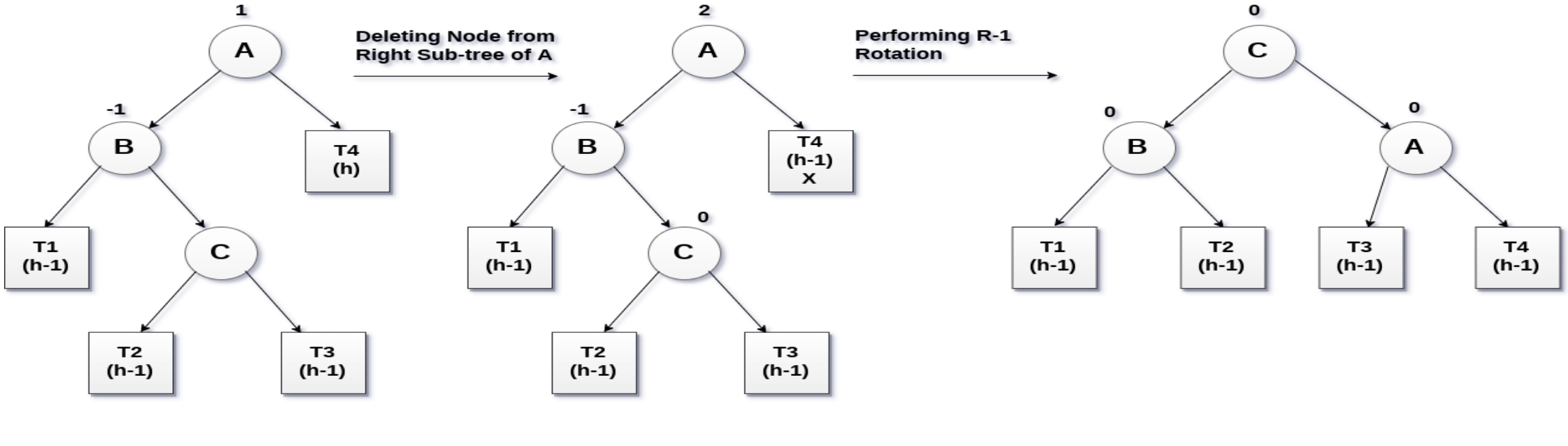


AVL Tree Deletion Operation

R-1 Rotation (Node B has balance factor -1)

R-1 rotation is to be performed if the node B has balance factor -1. This case is treated in the same way as LR rotation. In this case, the node C, which is the right child of node B, becomes the root node of the tree with B and A as its left and right children respectively. The sub-trees T1, T2 becomes the left and right sub-trees of B whereas, T3, T4 become the left and right sub-trees of A.

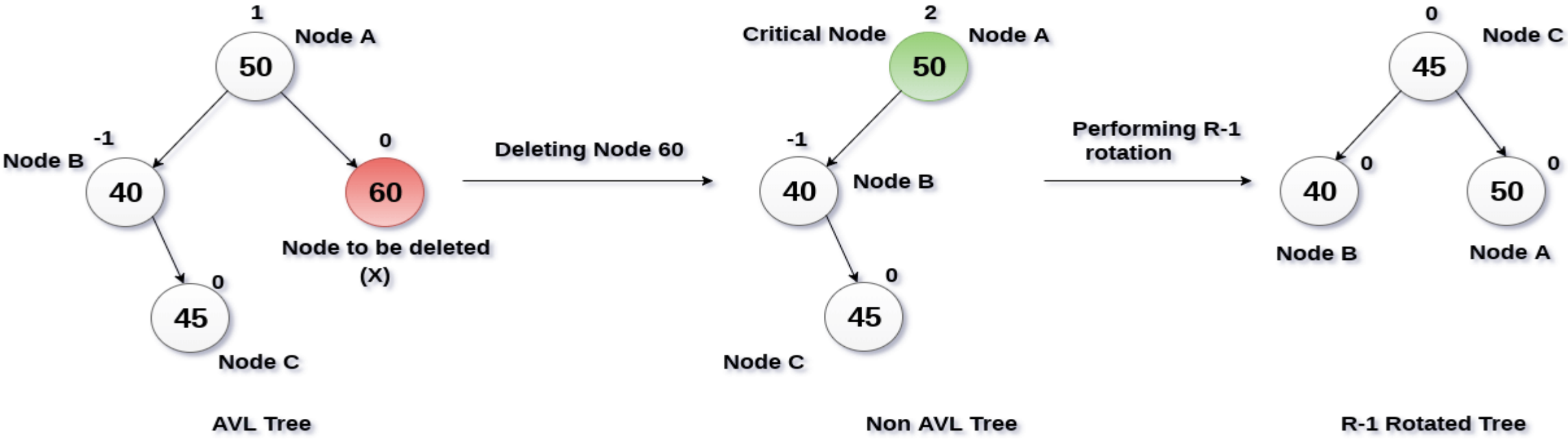
The process involved in R-1 rotation is shown in the following image.



AVL Tree Deletion Operation

Example:

in this case, node B has balance factor -1. Deleting the node 60, disturbs the balance factor of the node 50 therefore, it needs to be R-1 rotated. The node C i.e. 45 becomes the root of the tree with the node B(40) and A(50) as its left and right child.



Thank You