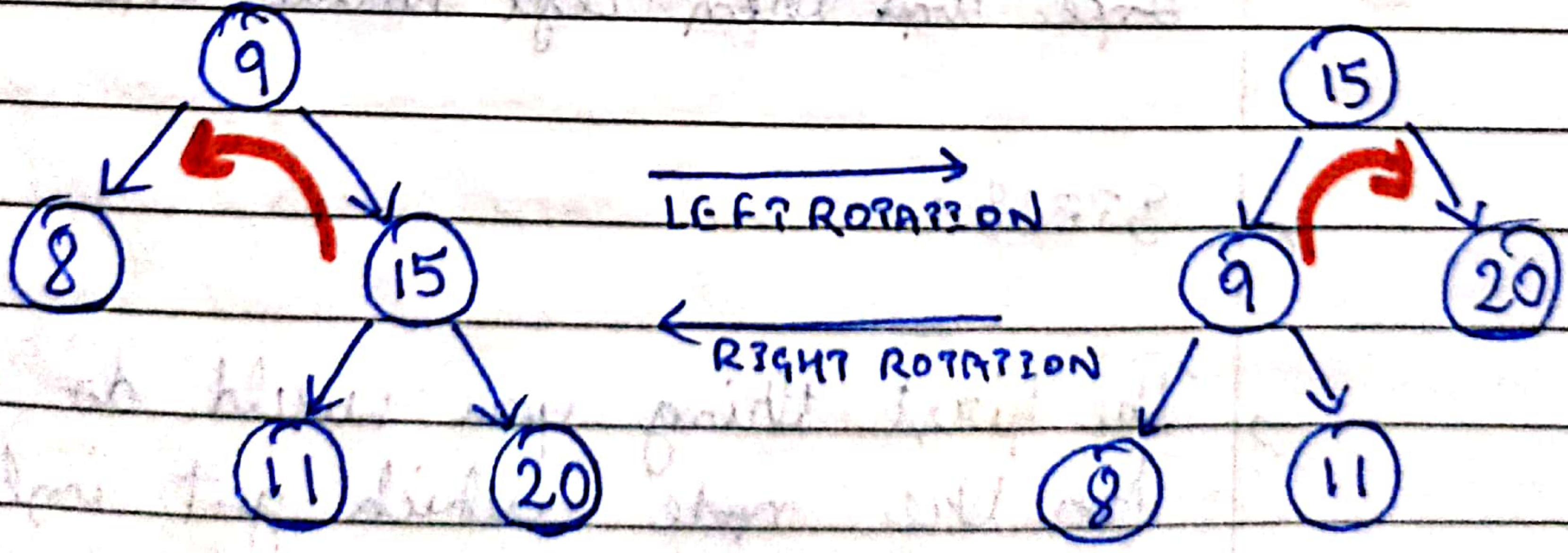
AVL Trees → LL LR RL and RR rotations :

→ Rotate Operations :

1) Left Rotate Operations.
2) Right Rotate Operations :



11 had to change its parent after the rotation to be able to maintain the balance of the tree. And rotating a tree to its left, and then again to the right, yields the same original tree as you can see from the above examples.

∴ When it comes to complex trees, in order to balance an AVL tree after insertion, we can follow the below-mentioned rules:

1) For Left-Left insertion → Right rotate once with respect to the first imbalance node.

2) For Right-Right insertion → Left rotate once with respect to the first imbalance node.

3) For Left-Right insertion → Left rotate once and then Right rotate once.

4) For Right-Left insertion → Right rotate once and then Left rotate once.

* STEPS :

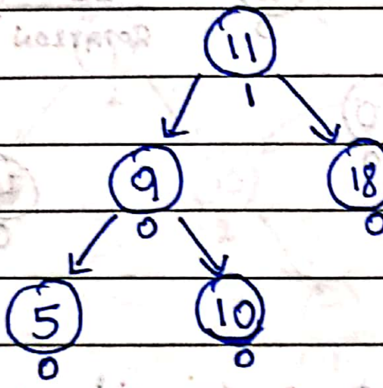
→ The first thing you would do is search for the node which got imbalanced

Page No.
Date

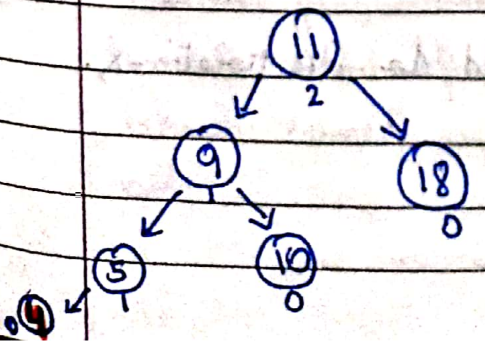
first. We start iterating from the node we inserted at and move upwards looking for that first imbalance node.

→ Second, you see what type of insertion was this with respect to the node we found.

→ LL Rotations :
LL rotations were discussed already. ~~So~~ We would just use it to balance a relatively complex AVL tree.



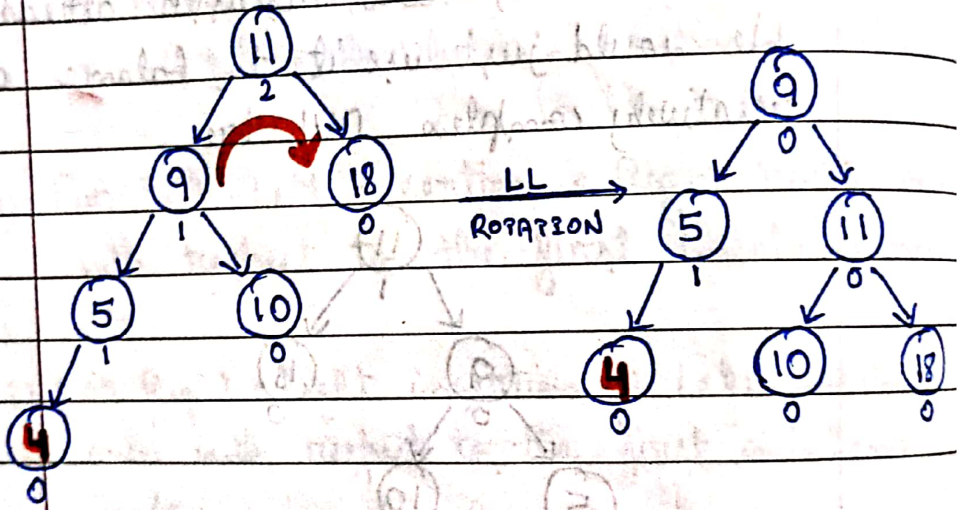
The tree is balanced and good for now. But now, we want to insert a node with data 4. So, that would get inserted to the left to node 5. The updated tree and their balance factors are:



LL insertion

Scanned with CamScanner

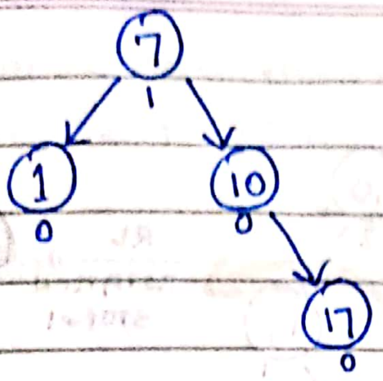
And since this is a case of left-left insertion, we would rotate right once with respect to the root node, since that's the first one to get imbalanced. And in that process, we might lose the position of node 10. So, we give it a new position to the left of node 11 to accomodate it again into the tree. And our tree gets balanced again.



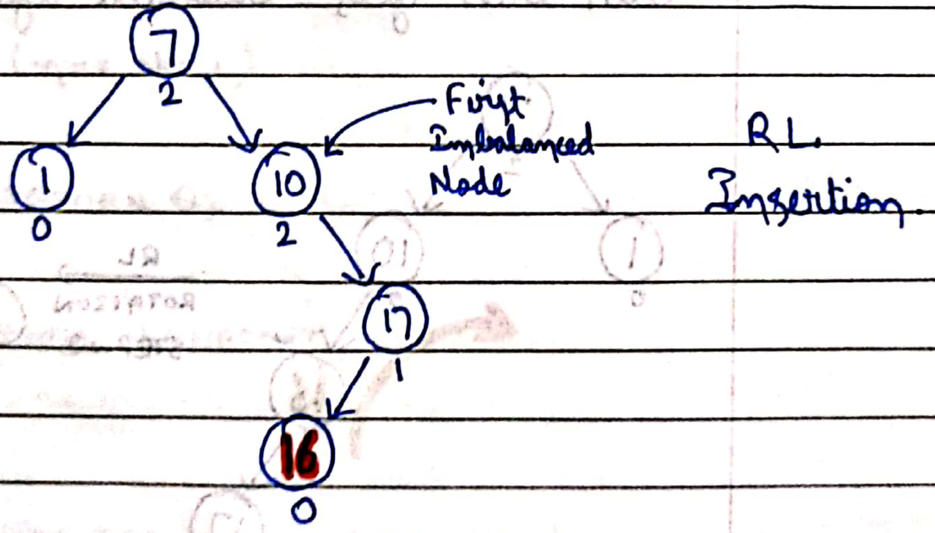
→ This is just a coincidence that our root node is the one we are rotating with respect to. We could come across examples where the first imbalanced node is not the root. So, we would rotate with respect to the one we'll find first, not the root.

→ Similarly, we would do RR rotations.

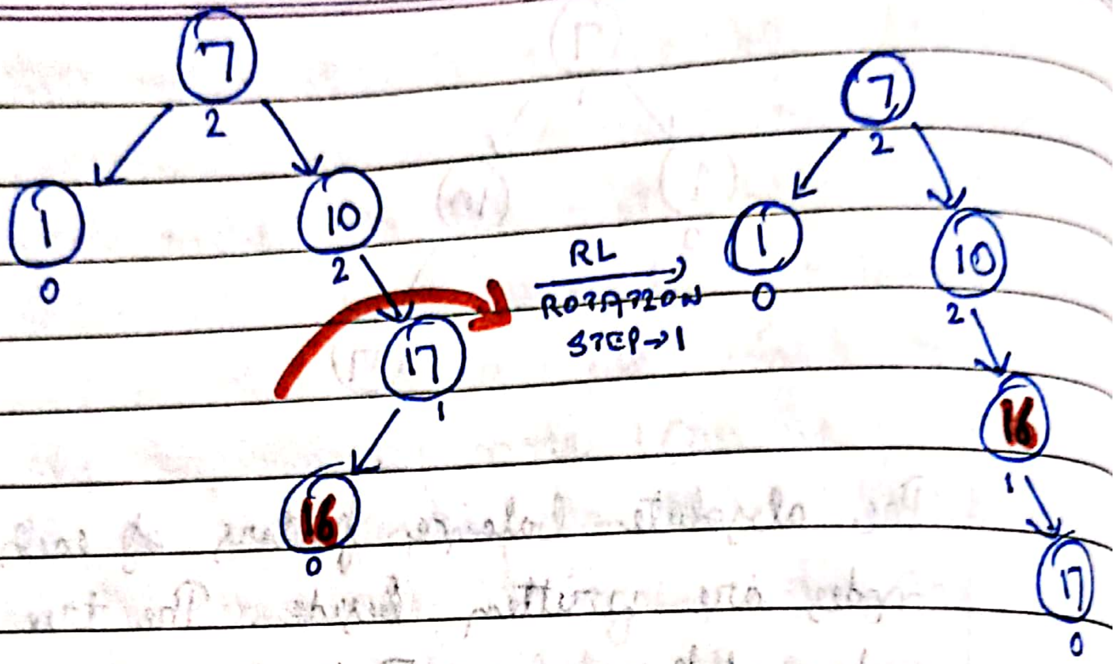
→ RL Rotations:



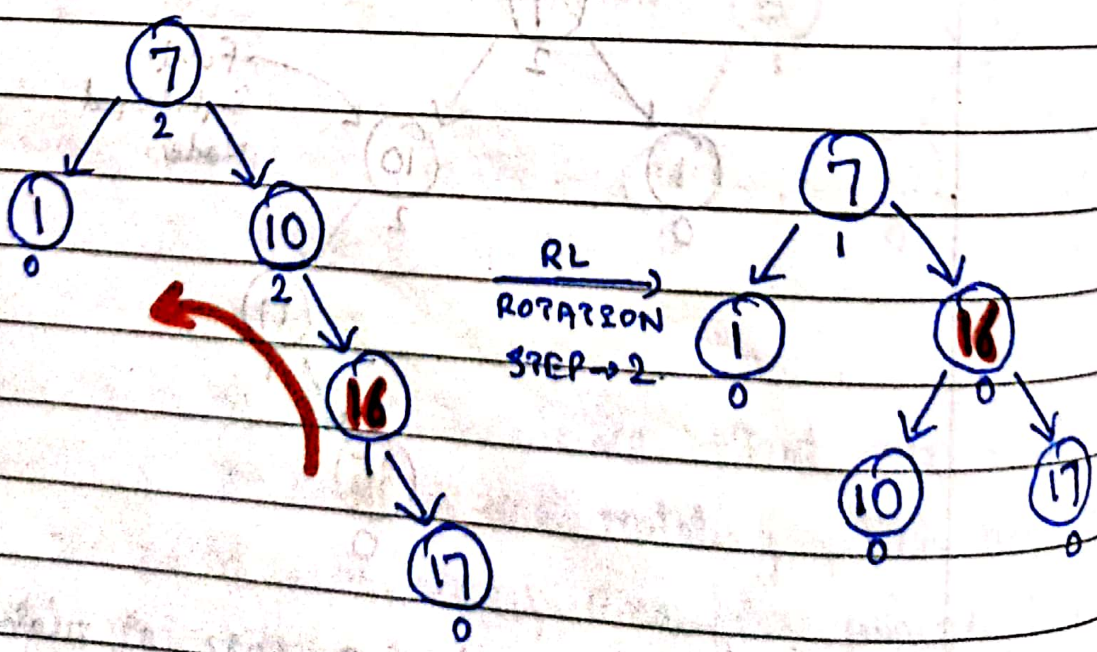
The absolute balance factors of each of the nodes are written beside. The tree is balanced and good for now. But now, we want to insert a node with data 16. So, that would get inserted to the left of node 17. The updated tree and their balance factors are:



And since this is a case of right-left insertion with respect to the first imbalanced node which is node 10, we would first rotate right once with respect to the child of the first imbalanced node which comes into the path of the insertion node.



And now, we rotate left with respect to the node we found first imbalanced, here 10. And this would do our job. Our tree gets balanced again.



→ Similarly, we would do LR rotations.

AVL insertio