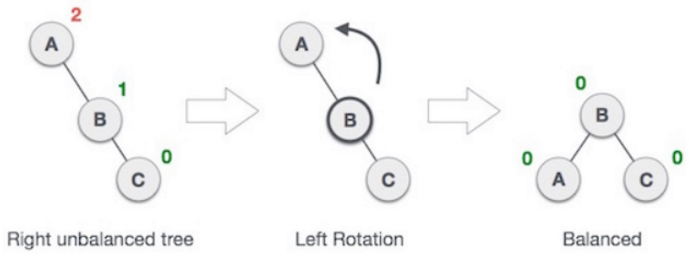
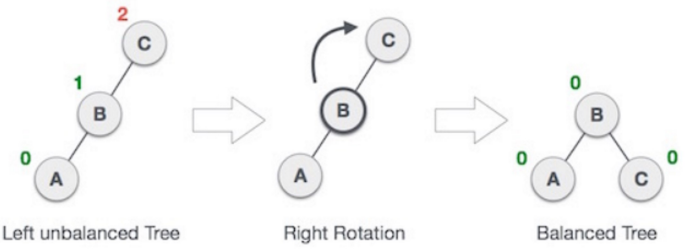
Left Rotation (LL Rotation)

In left rotations, every node moves one position to left from the current position.



Right Rotation (RR Rotation)

In right rotations, every node moves one position to right from the current position.



Balancing and Balance Factor

The balancing condition of AVL tree:

Balance factor = height(Left subtree) – height(Right subtree),

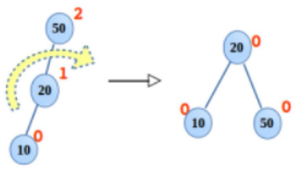
And it should be -1, 0 or 1. Other than this will cause restructuring (or balancing) the tree. Balancing performed is carried in the following ways,

1. Right rotation(RR)

In the binary search tree shown below is a case of right rotation. There is a single rotation required at the root 50, done as followed,

In the binary search tree shown below is a case of right rotation. There is a single rotation required at the root 50, done as followed,

- 20 will be the new root.
- 50 takes ownership of 20's right child,
- 10 is still the left child of 20.

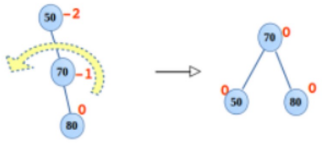


Right Rotation

2. Left rotation(LL)

In the binary search tree shown below is a case of left rotation where required. There is a single rotation required at the root 50, done as followed,

- 70 will be the new root.
- 50 takes ownership of 70's left child.
- 80 is still the right child of 70.

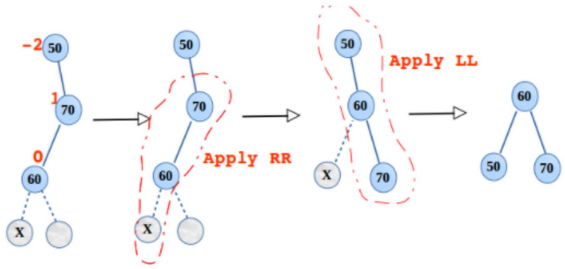


Left Rotation

3. Left right double rotation(LR)

Shown below is the case of LR rotation, here two rotations are performed. First RR and then, LL as follows,

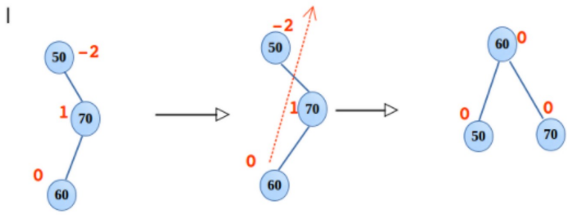
- Right rotation is applied at 70, after restructuring, 60 takes the place of 70 and 70 as the right child of 60.
- Now left rotation is required at the root 50, 60 becomes the root. 50 and 70 become the left and right child respectively.



LR Rotation

We could also think of the shown way to balance quickly rather than going with two rotations.

We could also think of the shown way to balance quickly rather than going with two rotations.

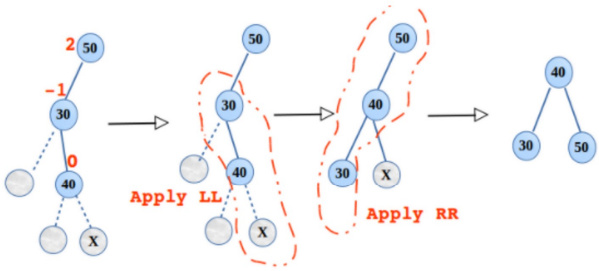


LR Rotation in a tricky way

4. Right left double rotation(RL)

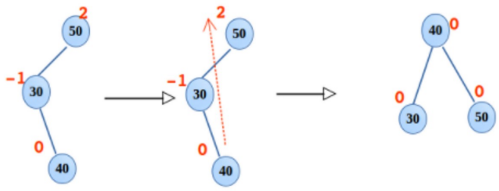
Shown below is the case of RL rotation, here two rotations are performed. First LL and then, RR as follows,

- Left rotation is applied at 30, after restructuring 40 takes the place of 30 and 30 as the left child of 40.
- Now right rotation is required at the root 50, 40 becomes root. 30 and 50 becomes the left and right child respectively.



RL Rotation

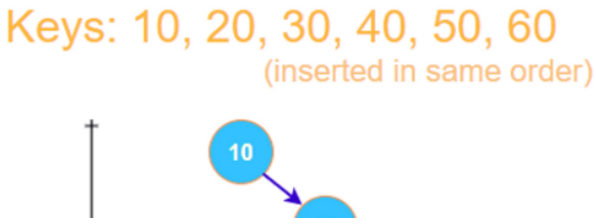
We could also think of the shown way to balance quickly rather than going with two rotations.

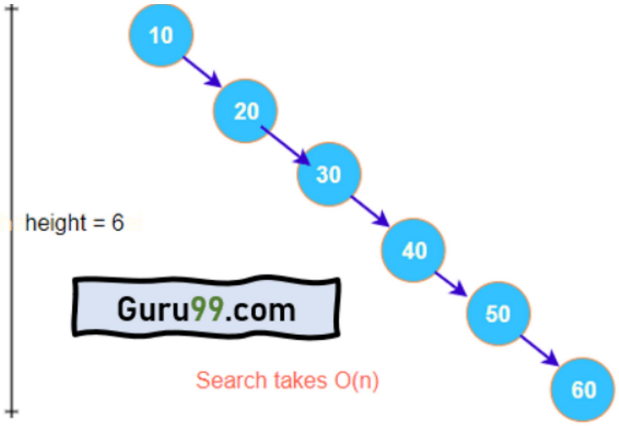


RL Rotation in a tricky way

To better understand the need for AVL trees, let us look at some disadvantages of simple binary search trees.

Consider the following keys inserted in the given order in the binary search tree.





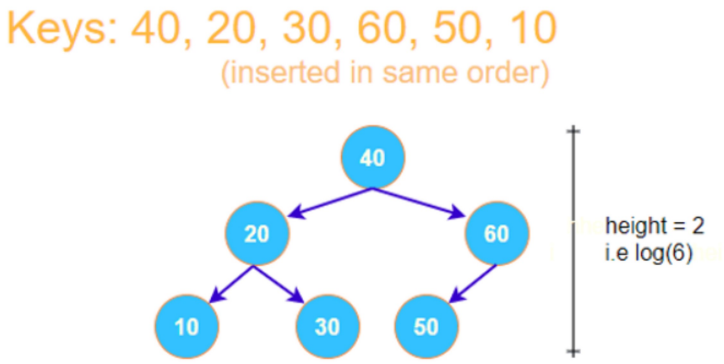
Height Unbalanced

AVL tree visualization

The height of the tree grows linearly in size when we insert the keys in increasing order of their value. Thus, the search operation, at worst, takes O(n).

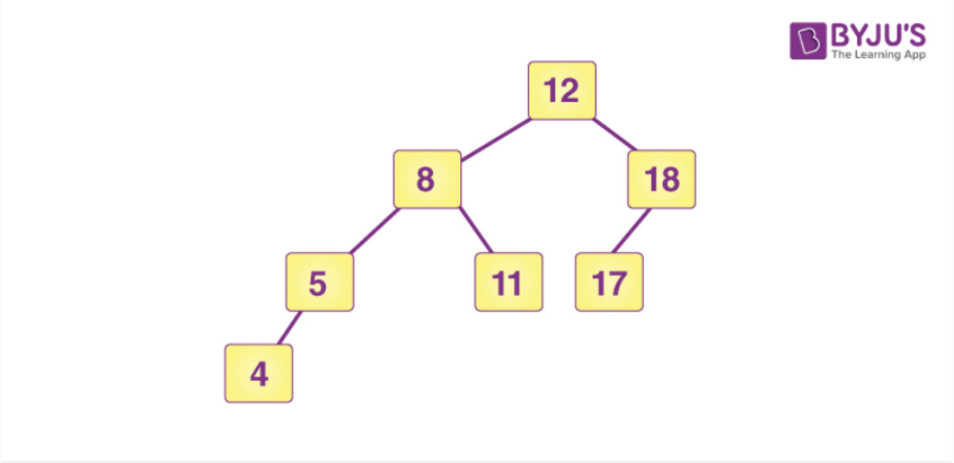
It takes linear time to search for an element; hence there is no use of using the Binary Search Tree structure. On the other hand, if the height of the tree is balanced, we get better searching time.

Let us now look at the same keys but inserted in a different order.

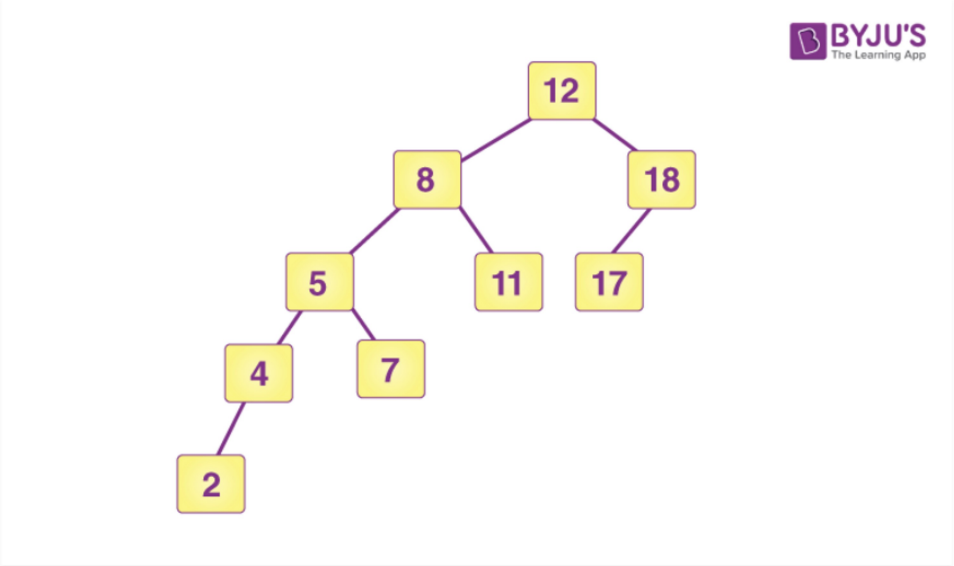


The term AVL tree was introduced by Adelson-Velsky and Landis. It is a balanced binary search tree and is the first data structure like this. In the AVL tree, the heights of the subtree cannot be more than one for all nodes.

first data structure like this. In this AVL tree, the heights of the subtree cannot be more than one for all nodes.



The above picture is of an AVL tree. Here, it is clearly visible that the heights of the left and right subtrees are equal to, or less than one.



The above tree is not an AVL tree. And, the reason is simple. Here, the heights of the left and right subtrees are higher than 1.

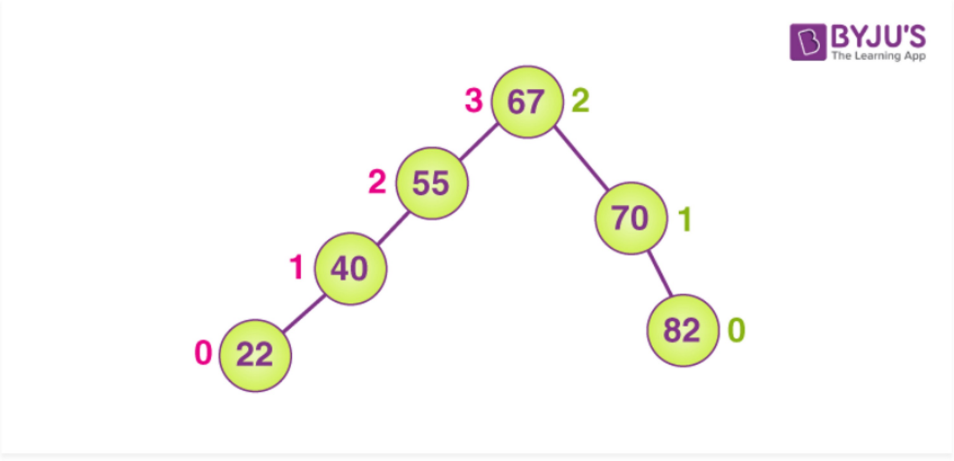
Balance Factor in AVL Tree

In the AVL tree, the term balance factor is very important.

Balance Factor = height(left-subtree) – height(right – subtree)

The balanced factor should be -1, 0 or +1. Otherwise, the tree will be considered an unbalanced tree.

Example Of AVL Tree & Balance Factor:



In the above example, the height of the left subtree is 3 and the height of the right subtree is 2. That means the

In the above example, the height of the left subtree is 3 and the height of the right subtree is 2. That means the balance factor is <=1 therefore the tree is supposed to be balanced.