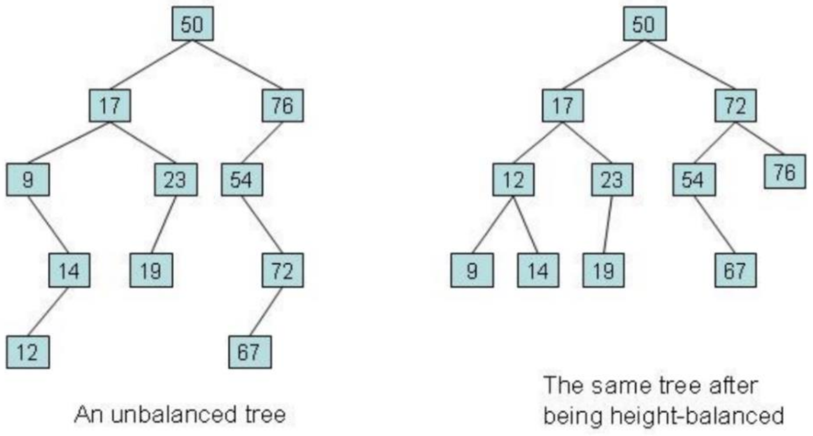
AVL Tree

- AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes.
- An AVL tree is a type of binary search tree. Named after it's inventors Adelson, Velskii, and Landis, AVL trees have the property of dynamic self-balancing in addition to all the other properties exhibited by binary search trees. A BST is a data structure composed of nodes.
- The name AVL tree is derived after its two creators, i.e. G.M. Abelson-Velvety and E.M. Landis.
- A balancing factor is a difference between the height of the left subtree and the right subtree. For a node to be balanced, it should be -1, 0, or 1.
- The performance of the lookup time depends on the shape of the tree. In the case of improper organization of nodes, forming a list, the process of searching for a given value can be the O(n) operation. With a correctly arranged tree, the performance can be significantly improved to O(log n).



Why AVL Trees?

- Most of the BST operations (e.g., search, max, min, insert, delete.. etc) take O(h) time where h is the height of the BST.
- The cost of these operations may become O(n) for a skewed Binary tree.
- If we make sure that height of the tree remains O(Log n) after every insertion and deletion, then we can guarantee an upper bound of O(Log n) for all these operations.
- The height of an AVL tree is always O(Log n) where n is the number of nodes in the tree.

Properties of an AVL tree:

- In an AVL tree, the heights of the two child subtrees of any node differ by at most one; therefore, it is also said to be height-balanced.
- Lookup, insertion, and deletion all take O(log n) time in both the average and worst cases, where n is the number of nodes in the tree.
- Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.
- The balance factor of a node is the height of its right subtree minus the height of its left subtree and a node with a balance factor 1, 0, or -1 is considered balanced.

Insertion*:*

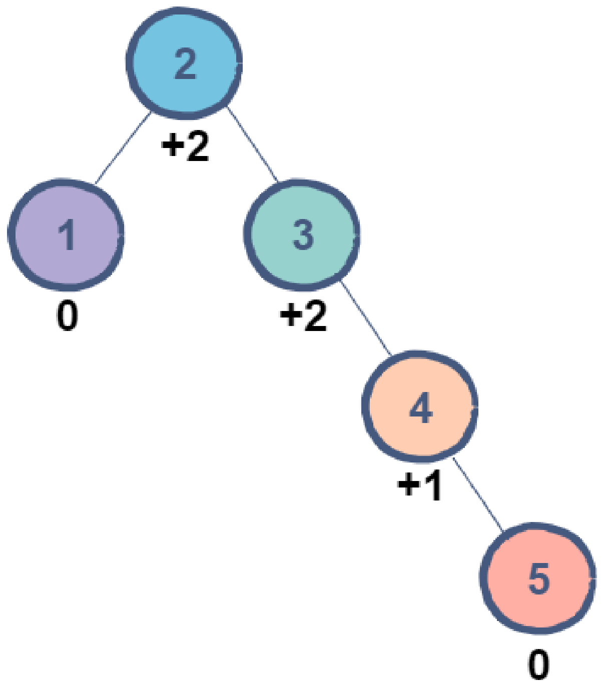
- After inserting a node, it is necessary to check *each of the node's ancestors* for consistency with the AVL rules.
- For each node checked, if the balance factor remains 1, 0, or -1 then no rotations are necessary. Otherwise, it's unbalanced.
- After each insertion, at most two tree rotations are needed to restore the entire tree.

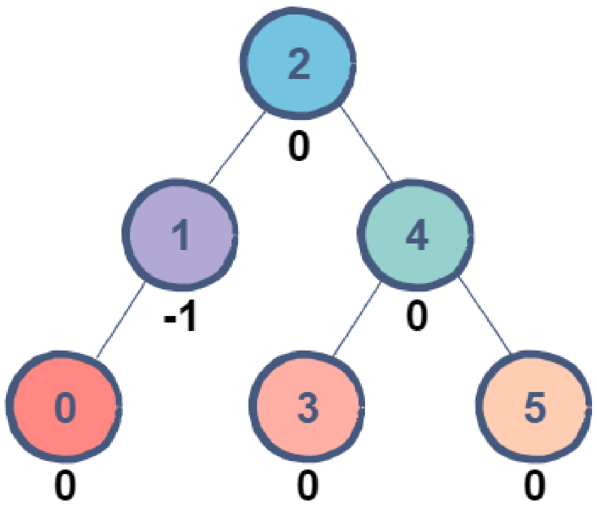
Deletion:

- If a node is a leaf, remove it.
- If the node is not a leaf, replace it with either the largest in its left subtree (rightmost) or the smallest in its right subtree (leftmost), and remove that node. The node that was found as replacement has at most one subtree.
- After deletion, retrace the path from parent of the replacement to the root, adjusting the balance factors as needed.
- More complicated rules for stopping. The retracing can stop if the balance factor becomes -1 or +1 indicating that the height of the subtree has remained unchanged.
- If the balance factor becomes 0 then the height of the subtree has decreased by one and the retracing needs to continue.
- This is in contrast to an insertion where a rotation resulting in a balance factor of 0 indicated that the subtree's height has remained unchanged.

Balance Factor:

- The balance factor of a node is the difference in the heights of the left and right subtrees. The balance factor of every node in the AVL tree should be either +1, 0 or -1.

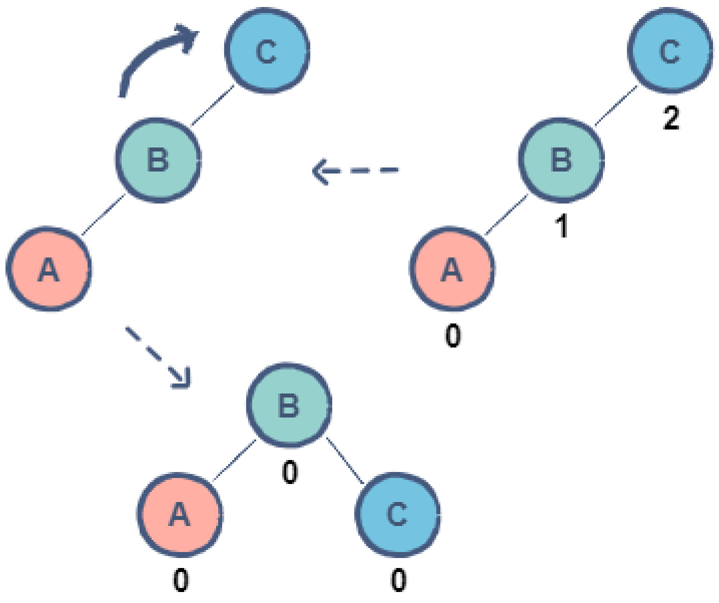




Rotation Techniques

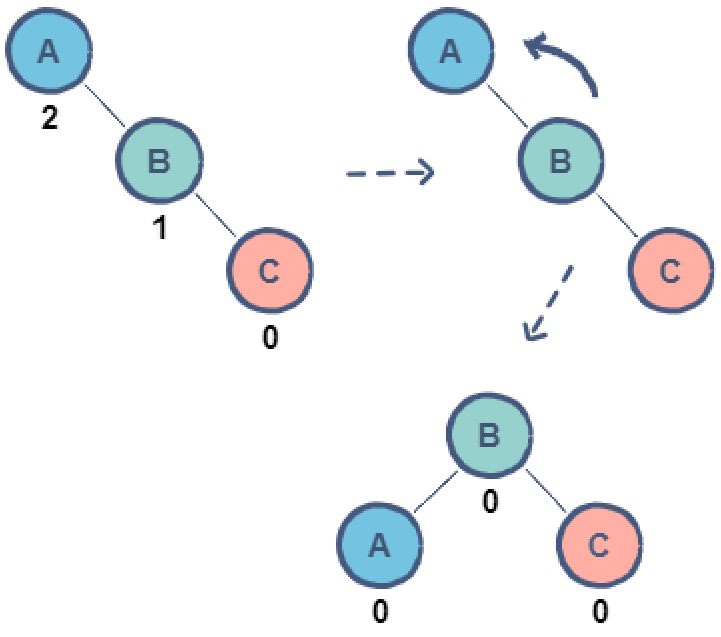
Right Rotation

- A single rotation applied when a node is inserted in the left subtree of a left subtree. In the given example, node C now has a balance factor of 2 after the insertion of node A. By rotating the tree right, node B becomes the root resulting in a balanced tree.



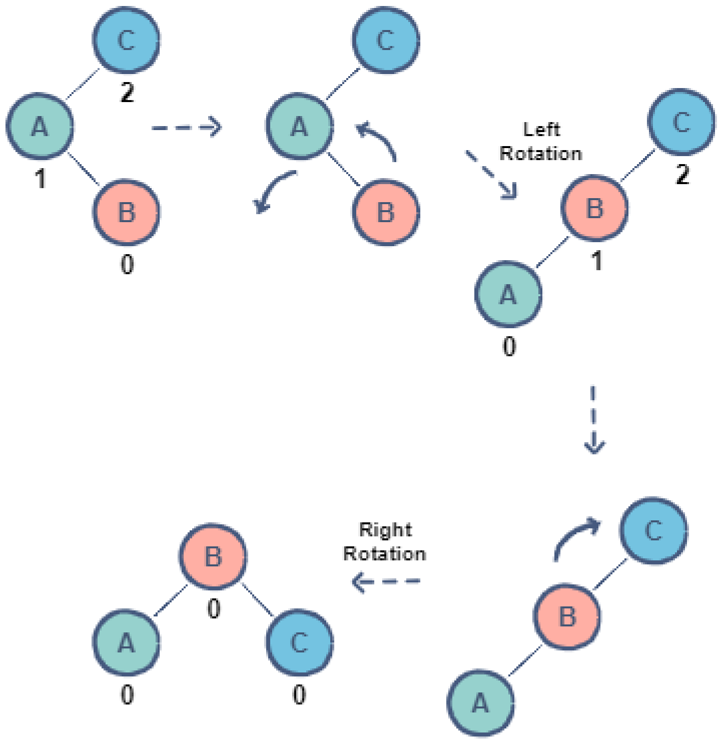
Left Rotation

- A single rotation applied when a node is inserted in the right subtree of a right subtree. In the given example, node A has a balance factor of 2 after the insertion of node C. By rotating the tree left, node B becomes the root resulting in a balanced tree.



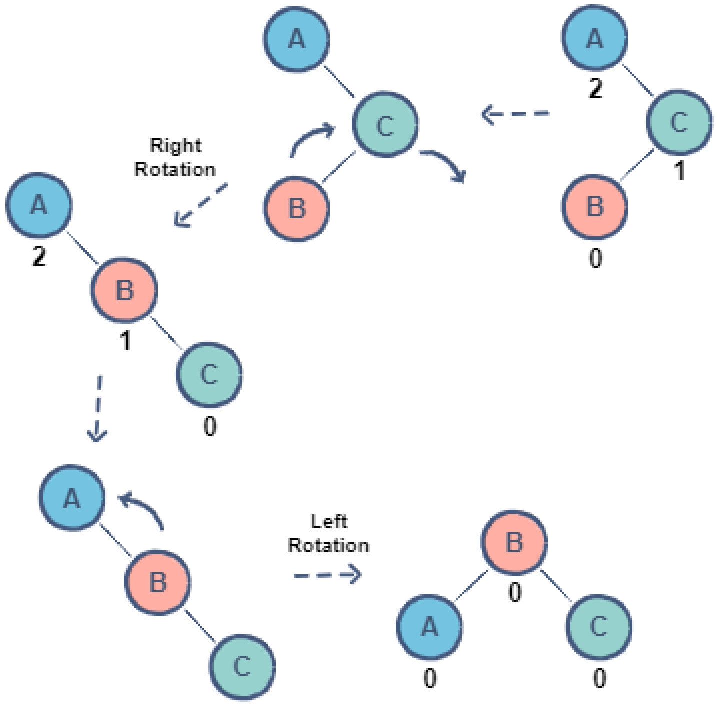
Left-Right Rotation

- A double rotation in which a left rotation is followed by a right rotation. In the given example, node B is causing an imbalance resulting in node C to have a balance factor of 2. As node B is inserted in the right subtree of node A, a left rotation needs to be applied. However, a single rotation will not give us the required results. Now, all we have to do is apply the right rotation as shown before to achieve a balanced tree.



Right-Left Rotation

- A double rotation in which a right rotation is followed by a left rotation. In the given example, node B is causing an imbalance resulting in node A to have a balance factor of 2. As node B is inserted in the left subtree of node C, a right rotation needs to be applied. However, just as before, a single rotation will not give us the required results. Now, by applying the left rotation as shown before, we can achieve a balanced tree.


A
2
C
1
B
0
A
C
B
Right Rotation
A
2
B
1
C
0
A
B
C
Left Rotation
B
0
A
0
C
0