DATA STRUCTURES & ALGORITHMS

Trees : AVL

Instructor: Engr. Laraib Siddiqui

Balanced Binary Tree

- Self balancing binary search tree.
- Adjust itself in order to maintain a low logarithmic height allowing for faster operations: insertion & deletions.

Tree is balanced ????

- ✓ check balance factor

Balance Factor = height(leftSubTree) – height(rightSubTree)

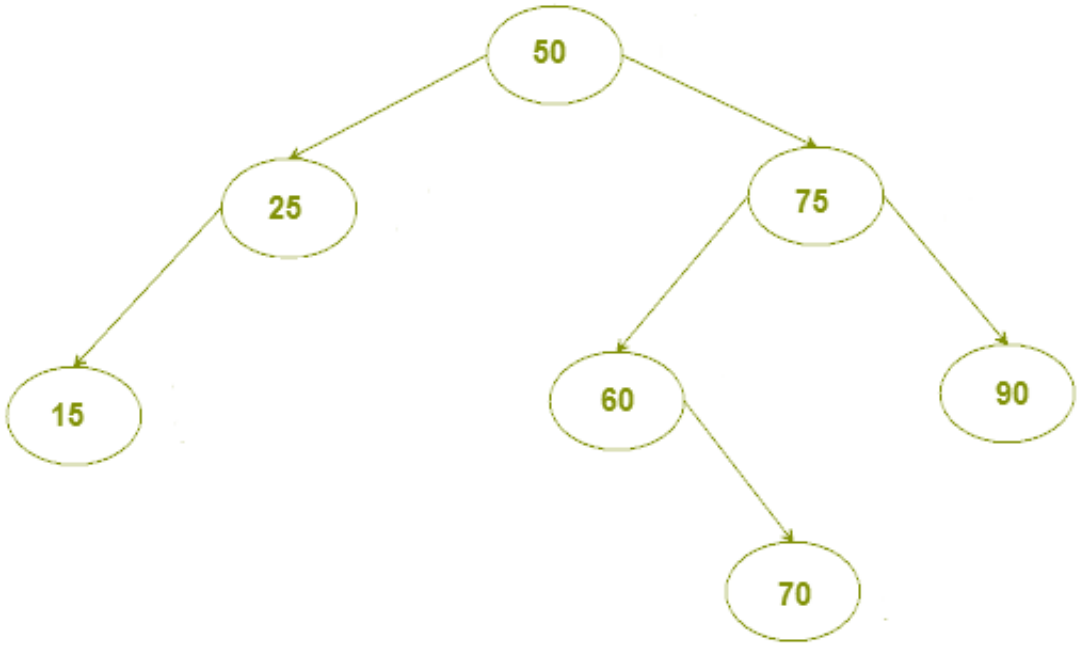
AVL Search Tree

- They are height-balanced binary search trees
- The invariant in the AVL which forces it to remain balanced is the requirement that the balance factor is always either -1 , 0 or +1.
- Balance factor can be determine by:

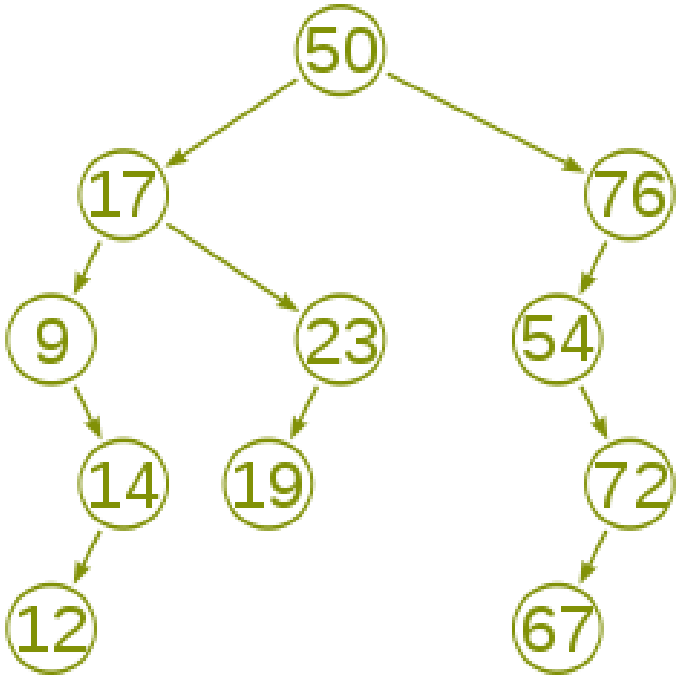
height(left subtree) - height(right subtree)

Note: If a node's BF is not -1 , 0 or +1 then it can be adjusted through rotations.

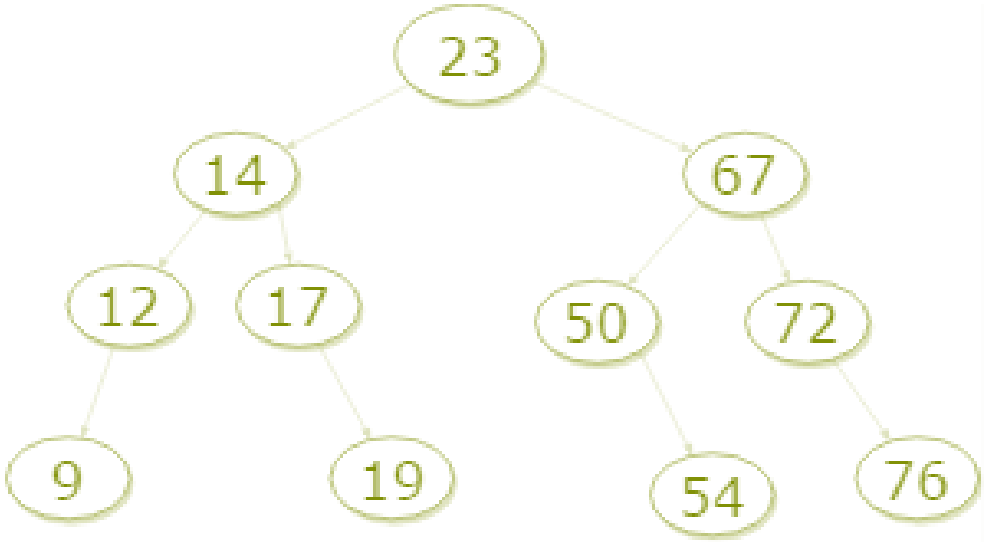
Is this tree is balanced?



Is this tree is balanced?

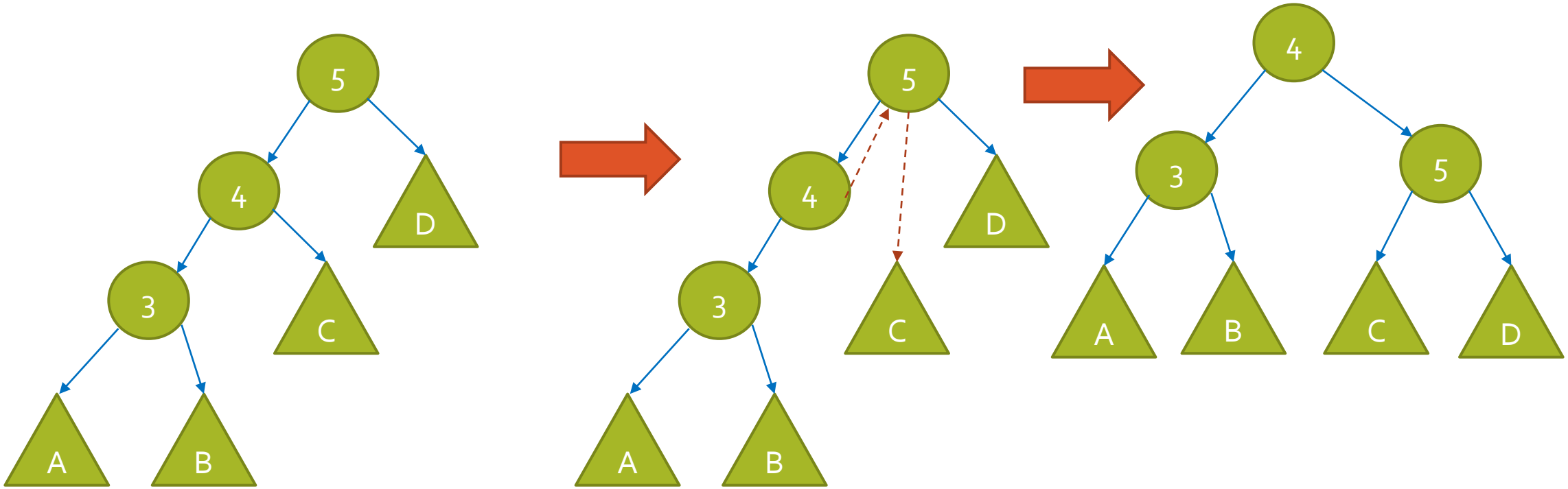


Is this tree is balanced?



Tree Rotations

We can shuffle/ transform / rotate the value and nodes in the tree as long as BST invariant satisfied!



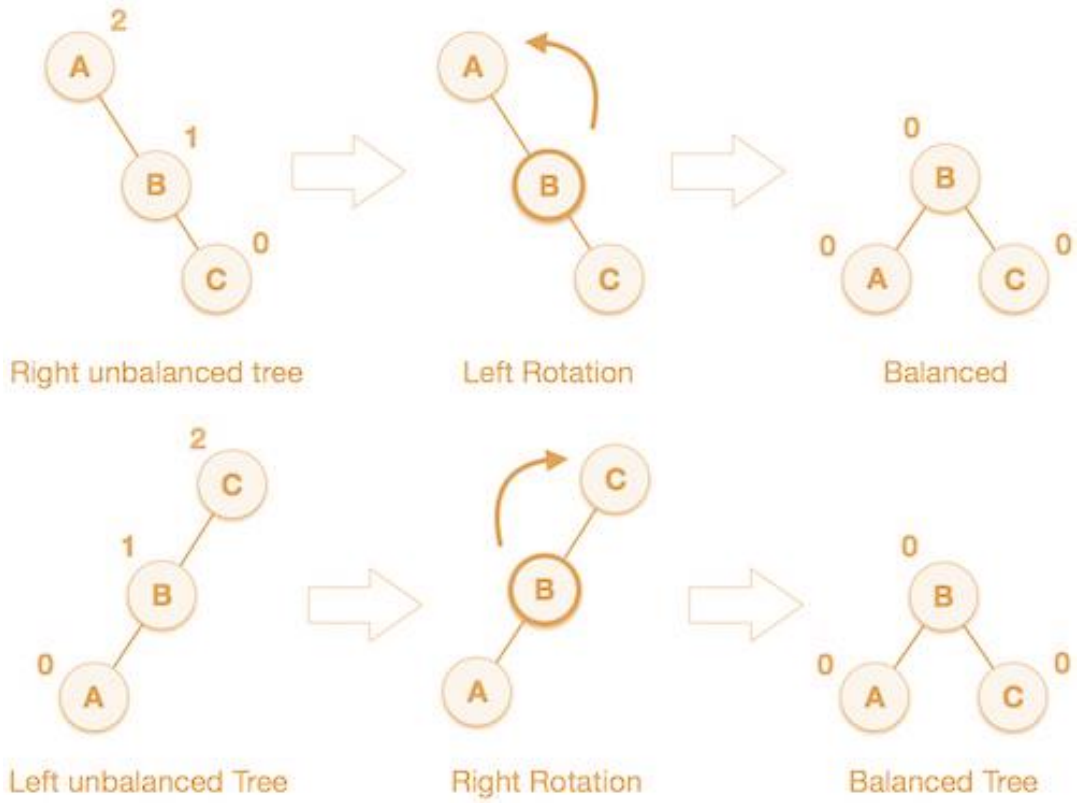
AVL Rotations

Left Rotation

If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation

Right Rotation

If a tree becomes unbalanced, when a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



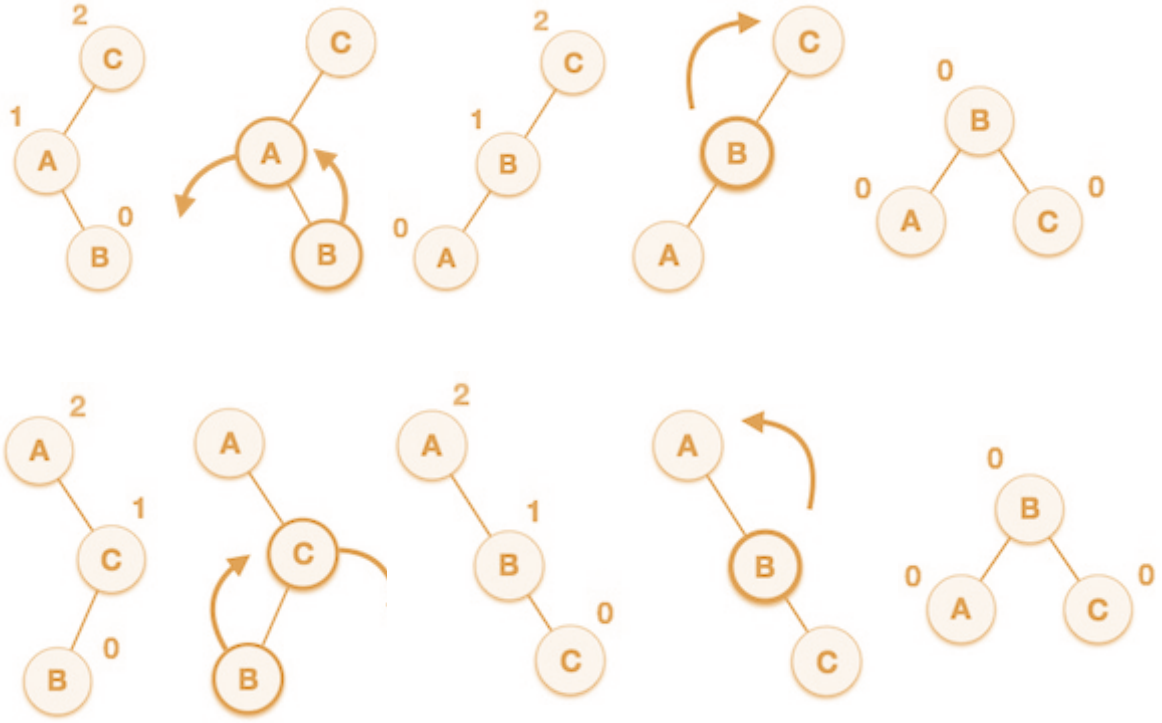
AVL Rotations

Left-Right Rotation

If a node has been inserted into the right subtree of the left subtree. This makes an unbalanced node. This scenario cause to perform left-right rotation.

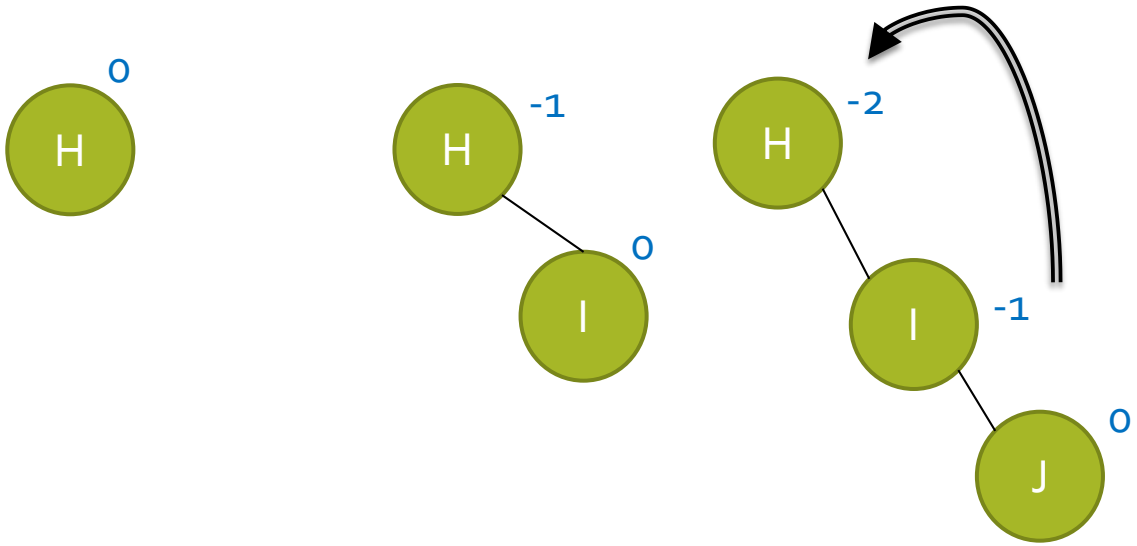
Right-Left Rotation

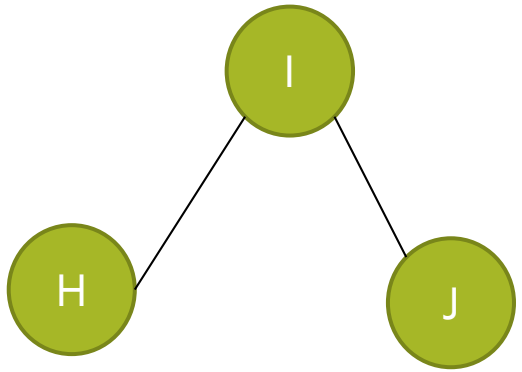
If a node has been inserted into the left subtree of the right subtree. This makes an unbalanced node.



AVL

Insert H, I, J, B, A, E, C, F, D

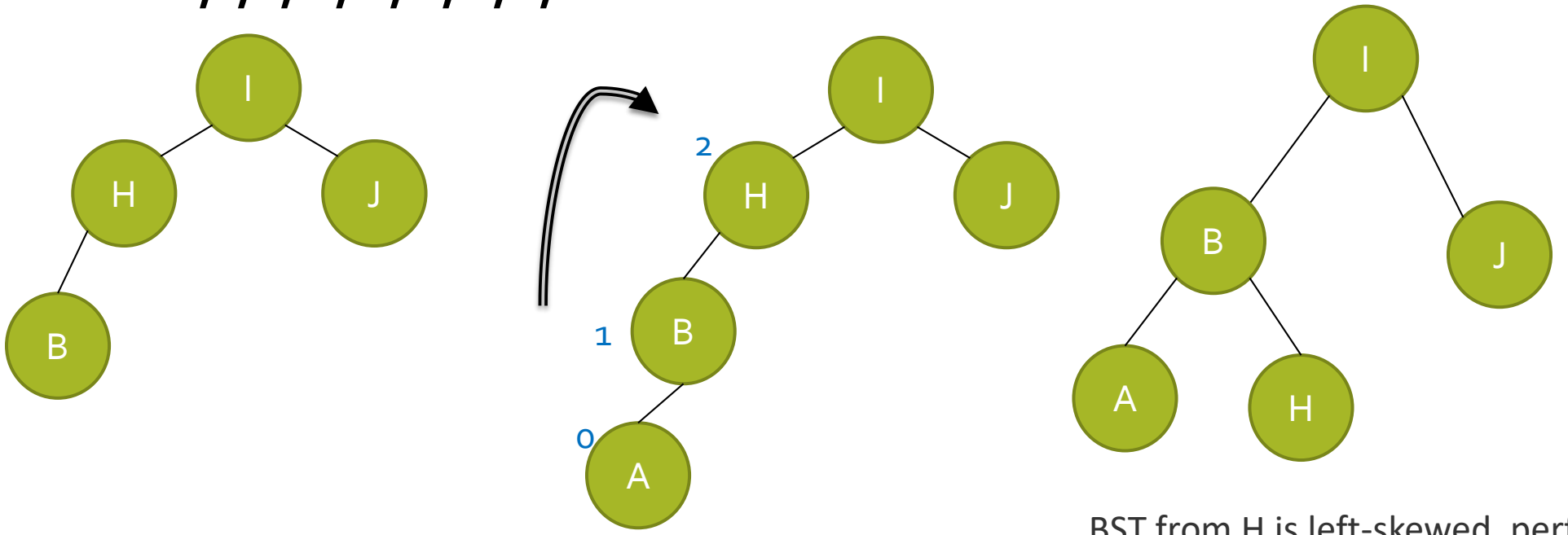




BST is right-skewed, perform RR Rotation on node H.

AVL

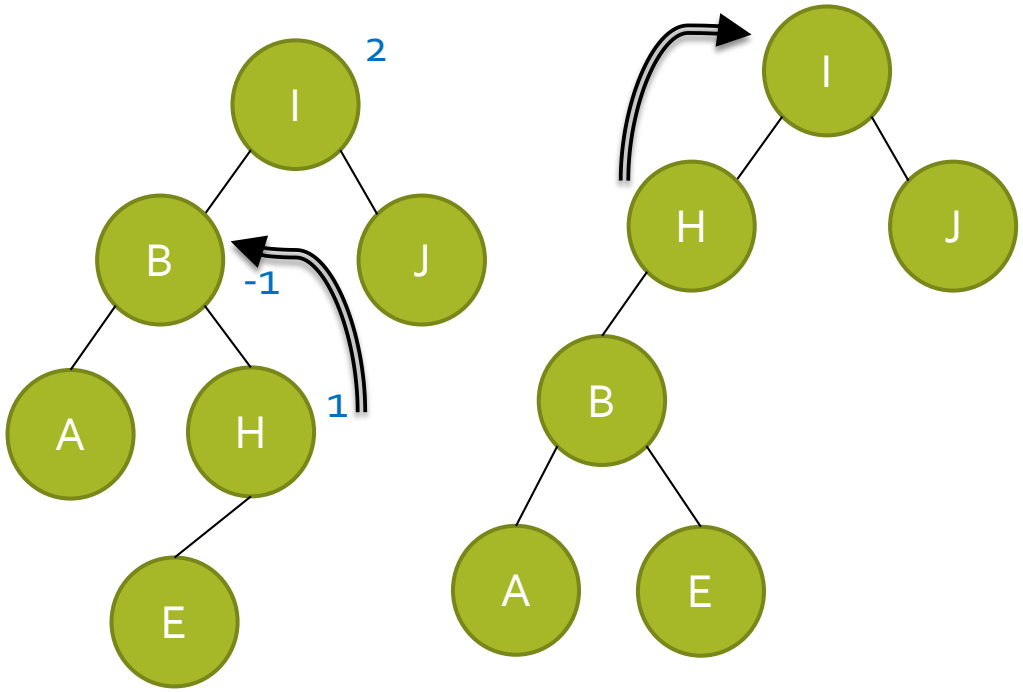
Insert H, I, J, B, A, E, C, F, D

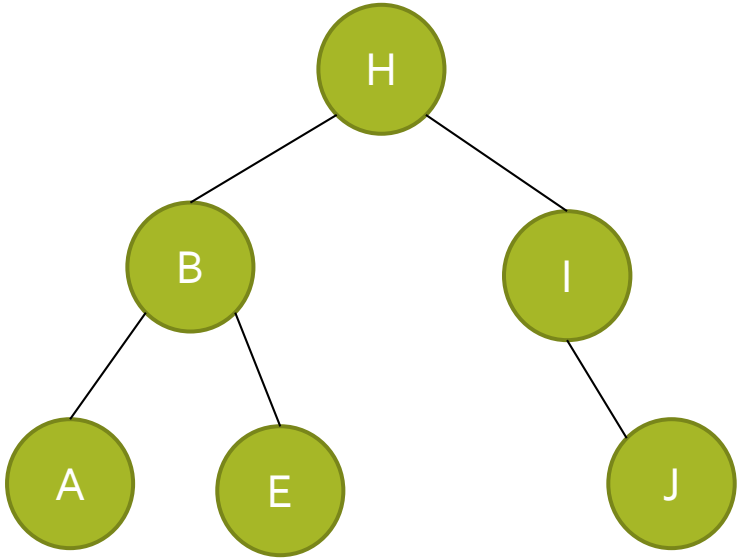


BST from H is left-skewed, perform LL Rotation on node H.

AVL

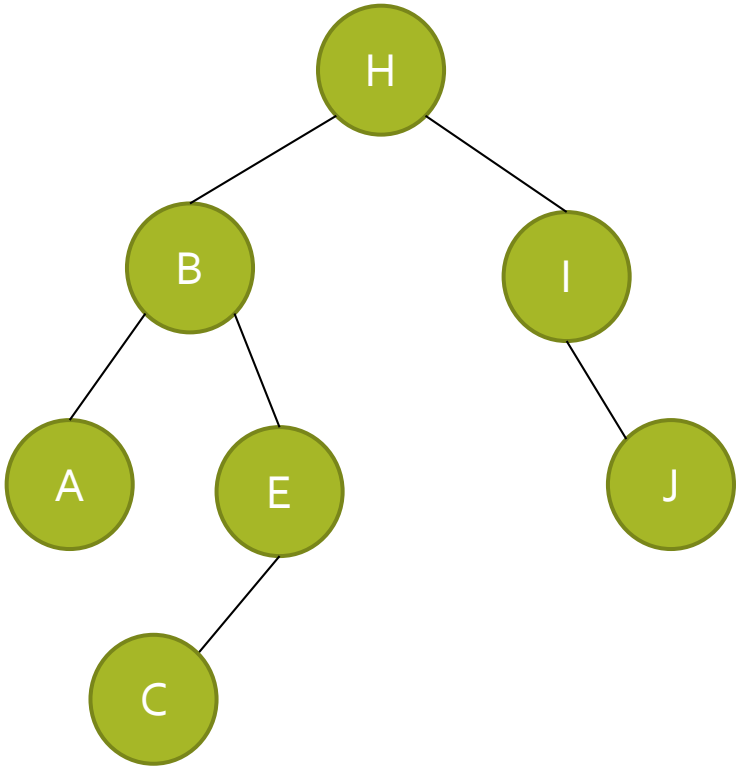
Insert H, I, J, B, A, E, C, F, D

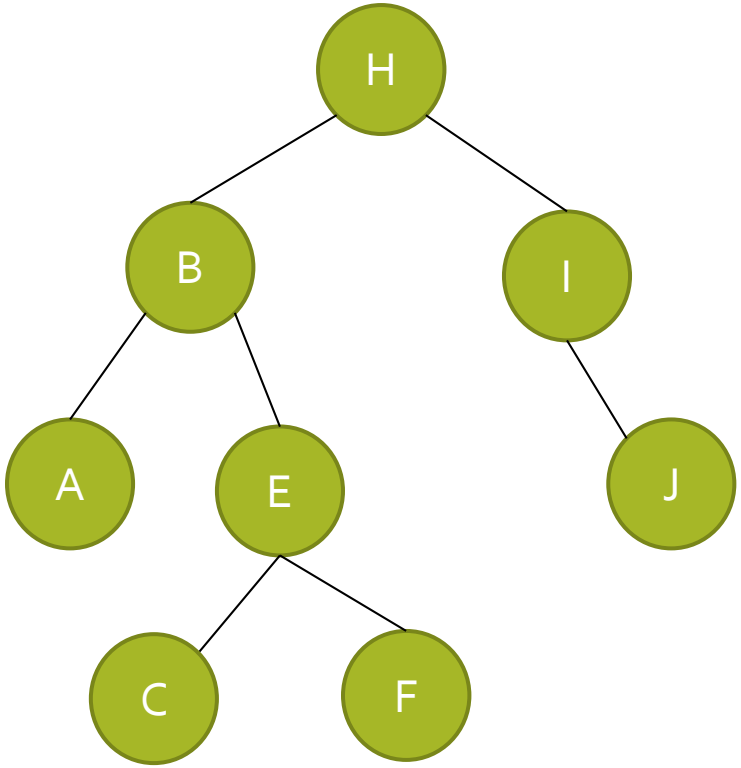




AVL

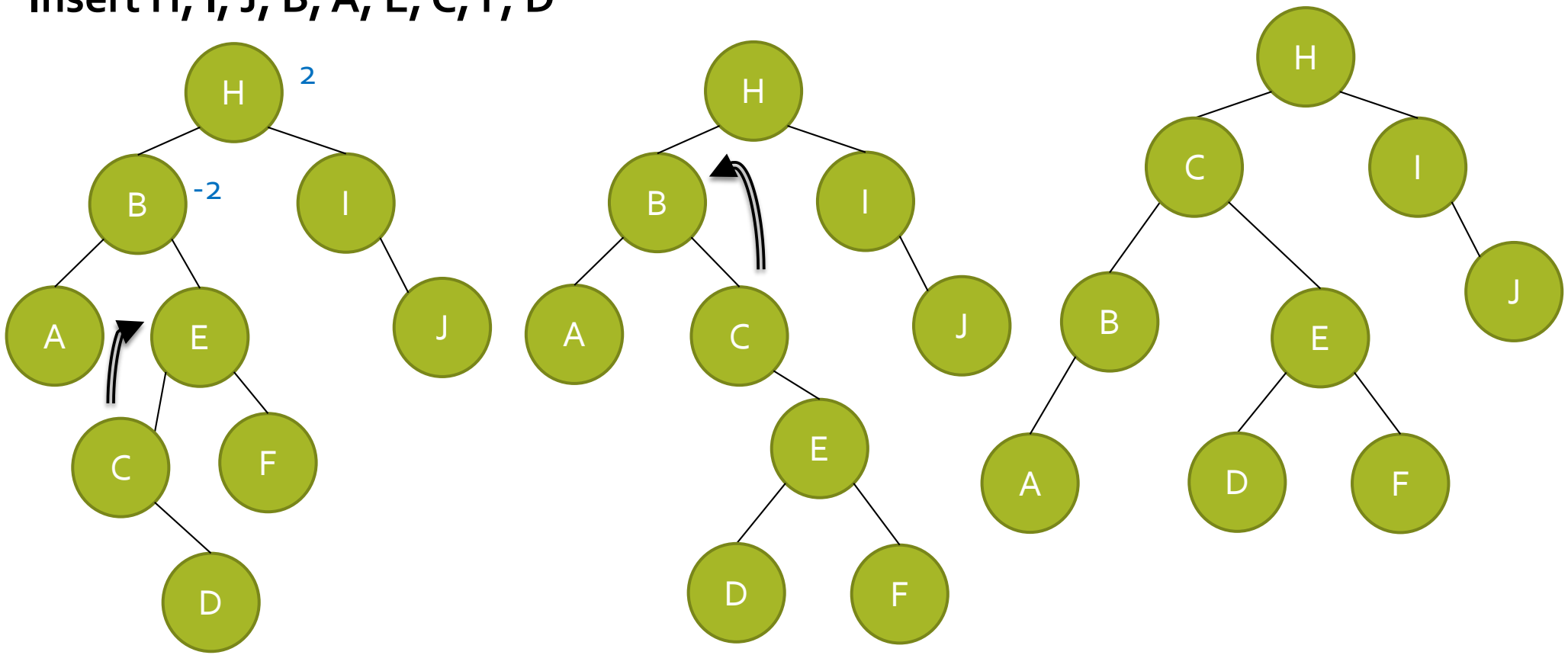
Insert H, I, J, B, A, E, C, F, D



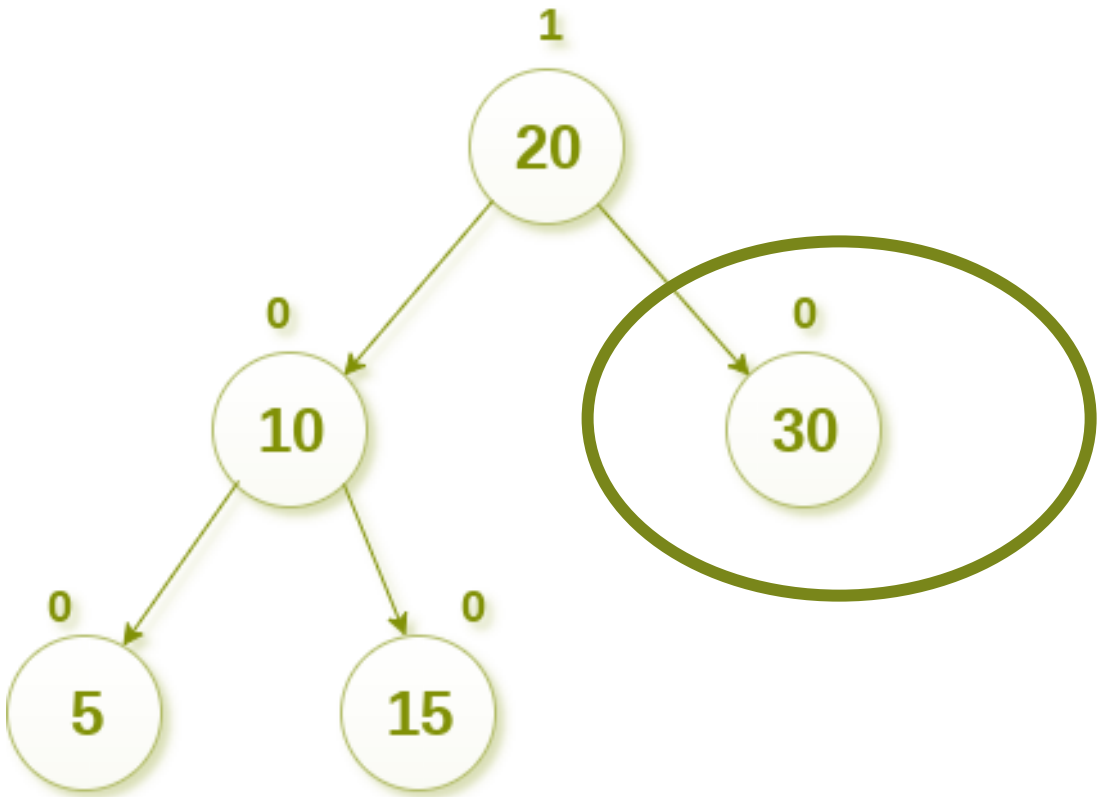


AVL

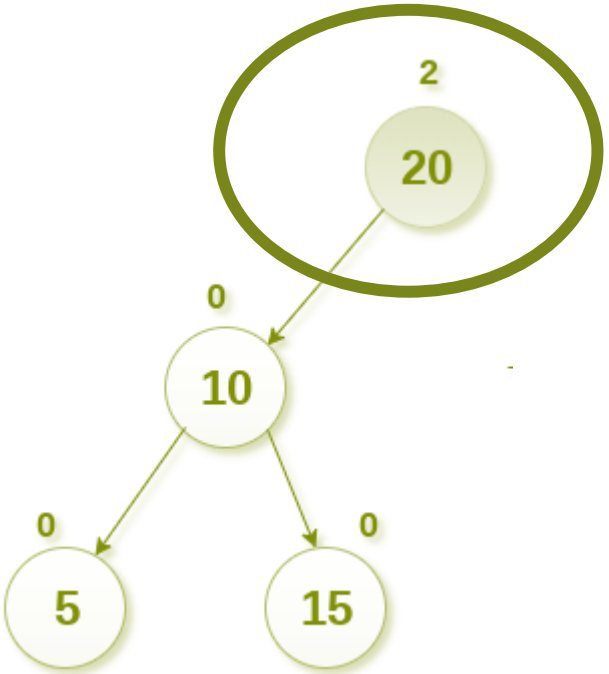
Insert H, I, J, B, A, E, C, F, D

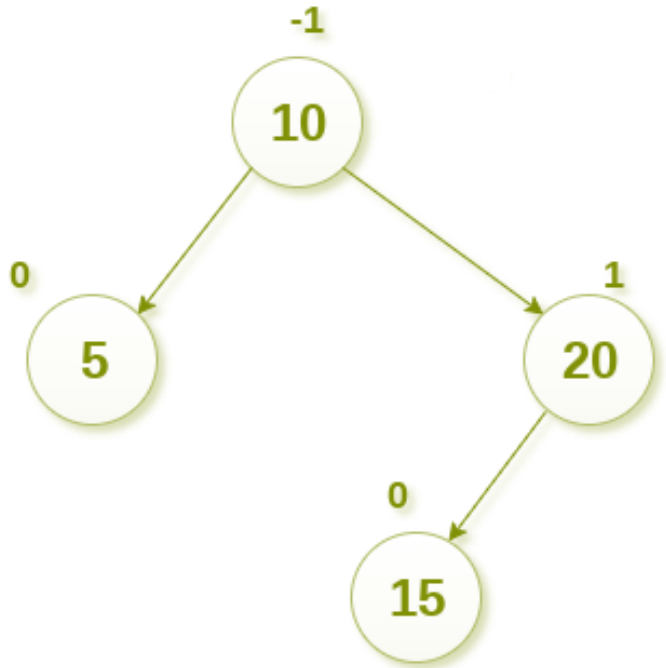


Deletion



Deletion





Complexity

	AVL Tree
Access	O(logn)
Search	O(logn)
Deletion	O(logn)
Insertion	O(logn)