DATA STRUCTURES

LECTURE-14

TREE

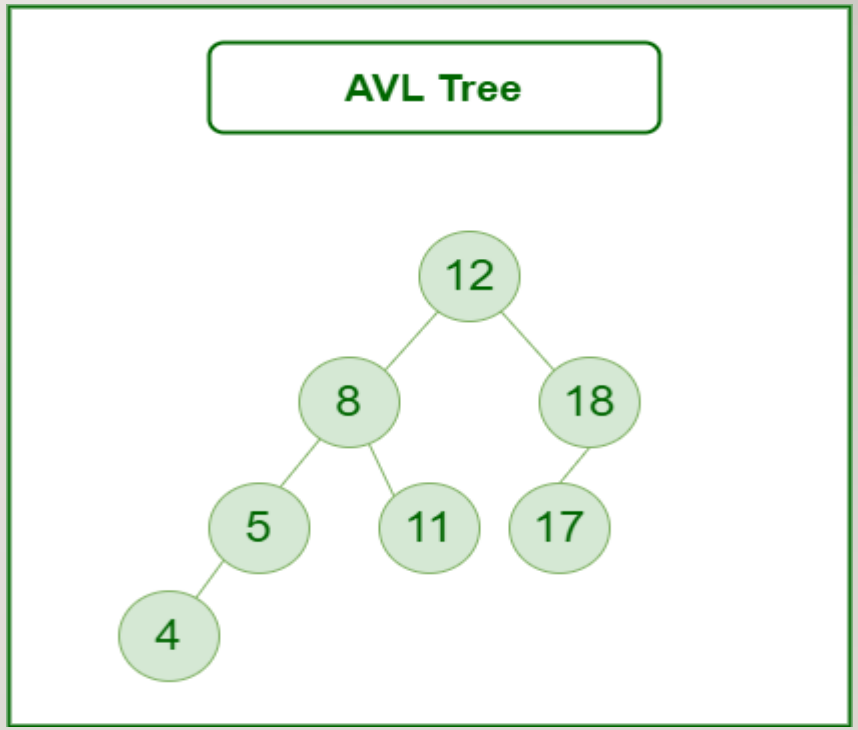
Dr. Sumitra Kisan

AVL Tree Data Structure

An AVL is a self-balancing Binary Search Tree (BST) where the difference between the heights of left and right subtrees of any node cannot be more than one.

KEY POINTS

- It is height balanced tree
- It is a binary search tree
- It is a binary tree in which the height difference between the left subtree and right subtree is almost one
- Height is the maximum depth from root to leaf



Characteristics of AVL Tree:

- It follows the general properties of a Binary Search Tree.
- Each subtree of the tree is balanced, i.e., the difference between the height of the left and right subtrees is at most 1.
- The tree balances itself when a new node is inserted. Therefore, the insertion operation is time-consuming

Advantages of AVL Tree:

- AVL trees can self-balance.
- It also provides faster search operations.
- AVL trees also have balancing capabilities with a different type of rotations
- Better searching time complexity than other trees, such as the binary Tree.
- Height must not be greater than log(N), where N is the total number of nodes in the Tree.

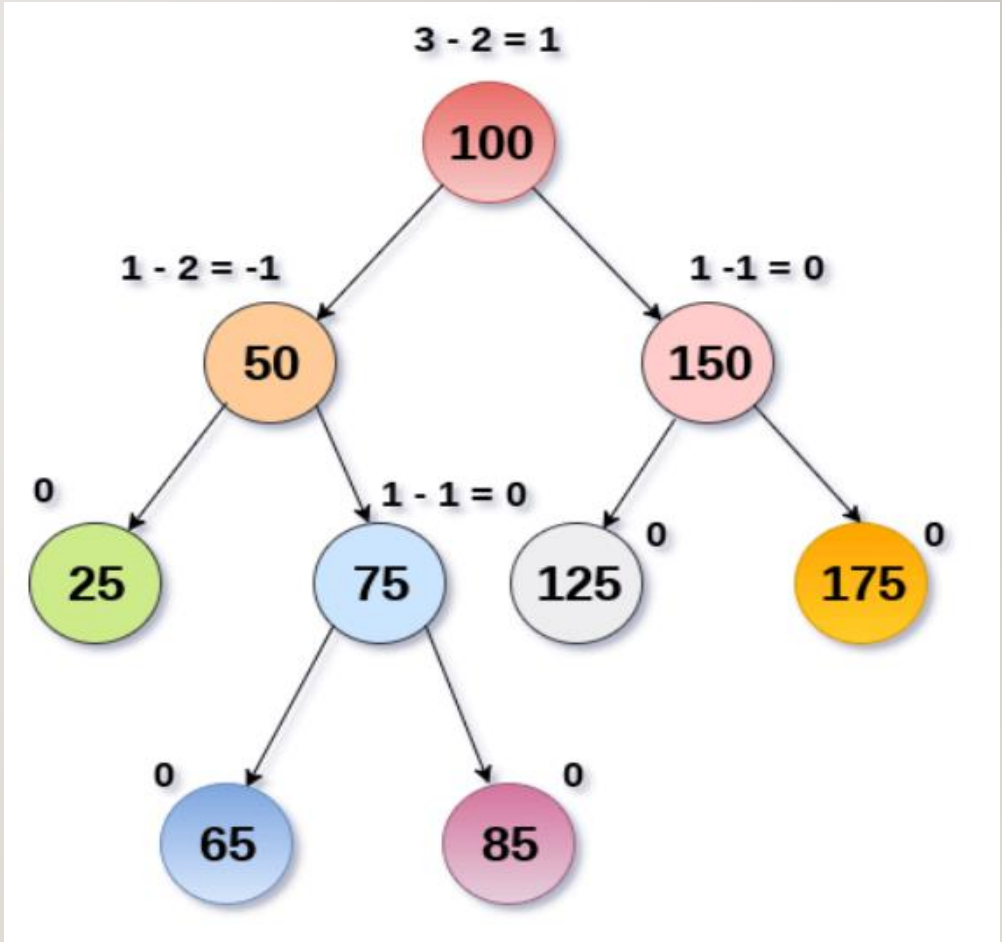
Disadvantages of AVL Tree:

- AVL trees are difficult to implement
- AVL trees have high constant factors for some operations

Tree is said to be balanced if balance factor of each node is in between -1 to 1, otherwise, the tree will be unbalanced and need to be balanced.

Balance Factor (k) = height (left(k)) - height (right(k))

- If balance factor of any node is 1, it means that the left sub-tree is one level higher than the right sub-tree.
- If balance factor of any node is 0, it means that the left sub-tree and right sub-tree contain equal height.
- If balance factor of any node is -1, it means that the left sub-tree is one level lower than the right sub-tree.



AVL Rotations

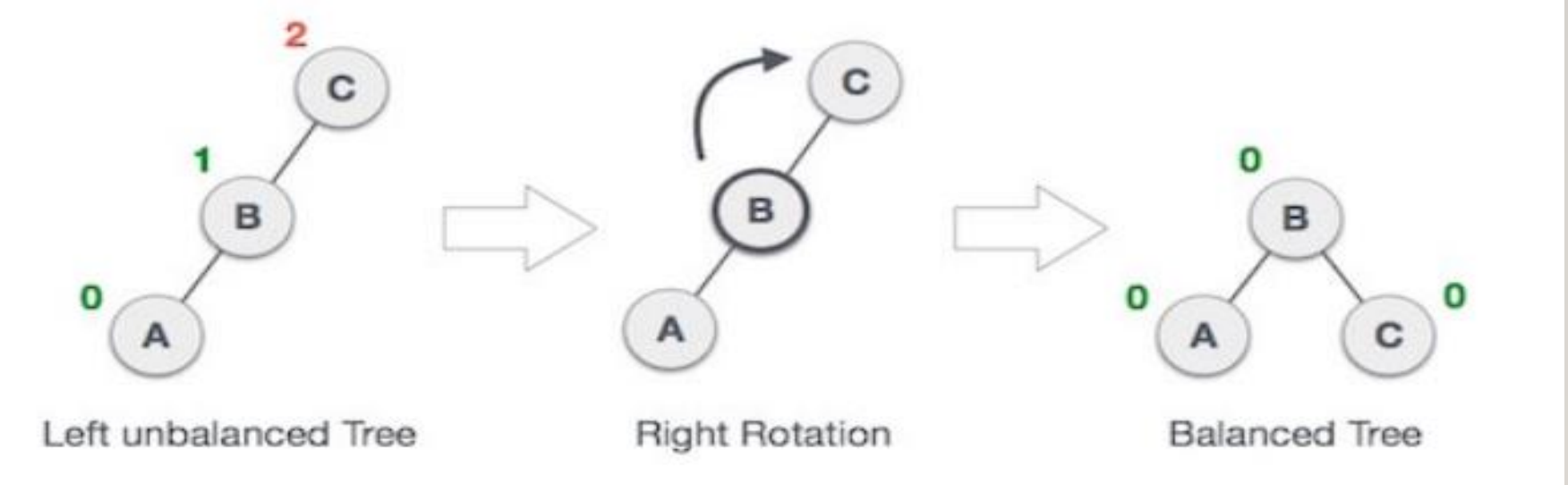
Rotation is performed in AVL tree only in case if Balance Factor is other than **-1, 0, and 1**. There are basically four types of rotations which are as follows:

- L L rotation: Inserted node is in the left subtree of left subtree of A
- R R rotation : Inserted node is in the right subtree of right subtree of A
- L R rotation : Inserted node is in the right subtree of left subtree of A
- R L rotation : Inserted node is in the left subtree of right subtree of A

Where node A is the node whose balance Factor is other than -1, 0, 1.

LL Rotation:

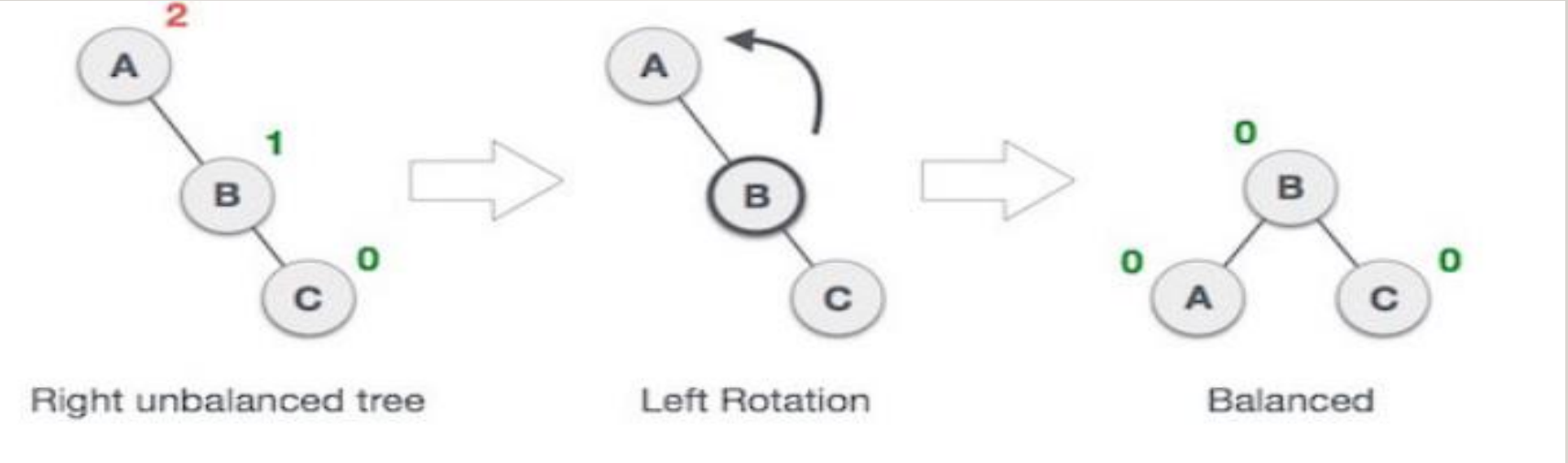
When BST becomes unbalanced, due to a node is inserted into the left subtree of the left subtree of C, then we perform LL rotation, LL rotation is clockwise rotation, which is applied on the edge below a node having balance factor 2.



In above example, node C has balance factor 2 because a node A is inserted in the left subtree of C left subtree. We perform the LL rotation on the edge below C.

RR Rotation:

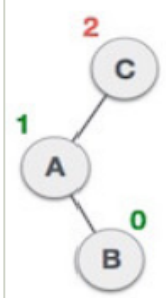
When BST becomes unbalanced, due to a node is inserted into the right subtree of the right subtree of A, then we perform RR rotation, RR rotation is an anticlockwise rotation, which is applied on the edge below a node having balance factor -2



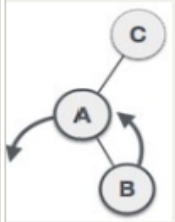
In above example, node A has balance factor -2 because a node C is inserted in the right subtree of A right subtree. We perform the RR rotation on the edge below A

LR Rotation:

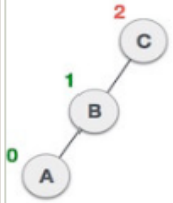
LR rotation = RR rotation + LL rotation, i.e., first RR rotation is performed on subtree and then LL rotation is performed on full tree, by full tree we mean the first node from the path of inserted node whose balance factor is other than -1, 0, or 1.



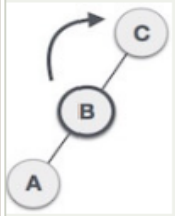
A node B has been inserted into the right subtree of A the left subtree of C, because of which C has become an unbalanced node having balance factor 2. This case is L R rotation where: Inserted node is in the right subtree of left subtree of C



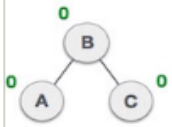
As LR rotation = RR + LL rotation, hence RR (anticlockwise) on subtree rooted at A is performed first. By doing RR rotation, node **A**, has become the left subtree of **B**.



After performing RR rotation, node C is still unbalanced, i.e., having balance factor 2, as inserted node A is in the left of left of **C**



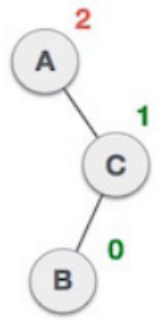
Now we perform LL clockwise rotation on full tree, i.e. on node C. node **C** has now become the right subtree of node B, A is left subtree of B



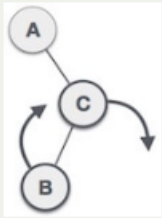
Balance factor of each node is now either -1, 0, or 1, i.e. BST is balanced now.

RL Rotation:

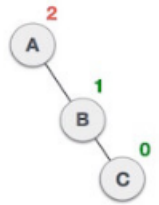
R L rotation = LL rotation + RR rotation, i.e., first LL rotation is performed on subtree and then RR rotation is performed on full tree, by full tree we mean the first node from the path of inserted node whose balance factor is other than -1, 0, or 1.



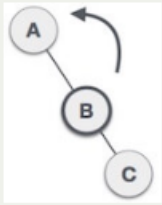
A node **B** has been inserted into the left subtree of **C** the right subtree of **A**, because of which A has become an unbalanced node having balance factor - 2. This case is RL rotation where: Inserted node is in the left subtree of right subtree of A



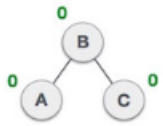
As RL rotation = LL rotation + RR rotation, hence, LL (clockwise) on subtree rooted at **C** is performed first. By doing RR rotation, node **C** has become the right subtree of **B**.



After performing LL rotation, node **A** is still unbalanced, i.e. having balance factor -2, which is because of the right-subtree of the right-subtree node A.



Now we perform RR rotation (anticlockwise rotation) on full tree, i.e. on node A. node **C** has now become the right subtree of node B, and node A has become the left subtree of B.

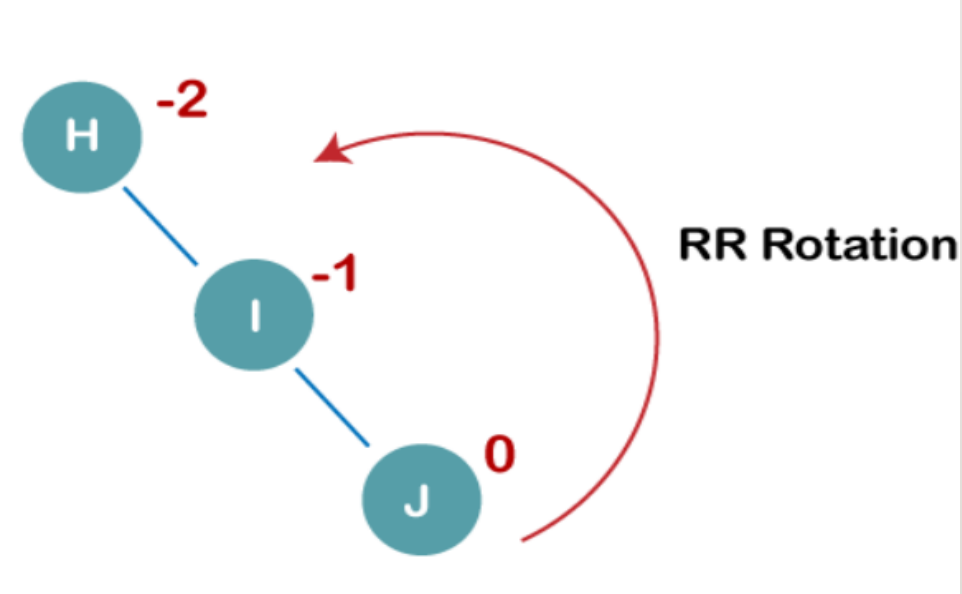


Balance factor of each node is now either -1, 0, or 1, i.e., BST is balanced now.

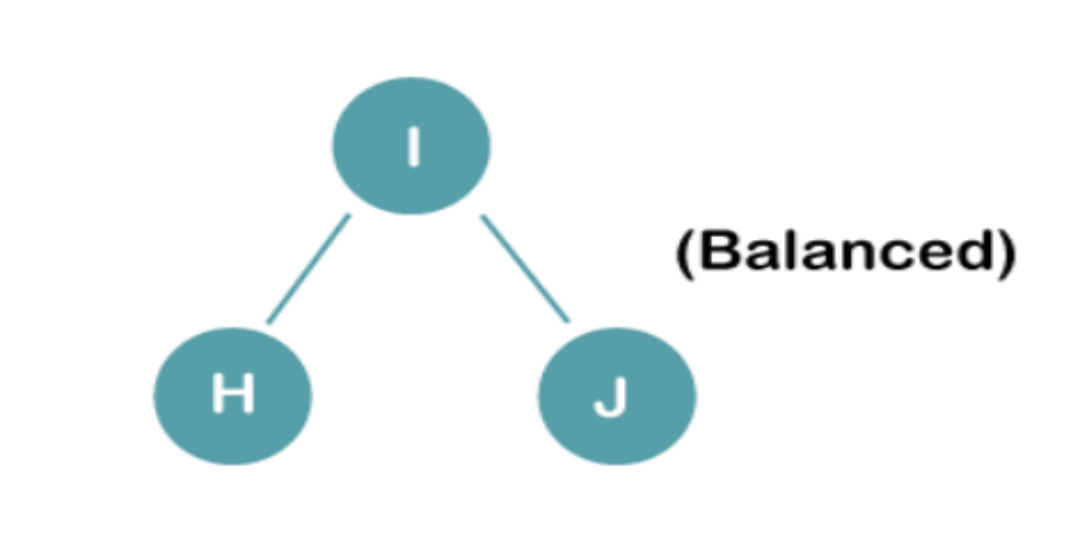
Construct an AVL tree having the following elements:

H, I, J, B, A, E, C, F, D, G, K, L

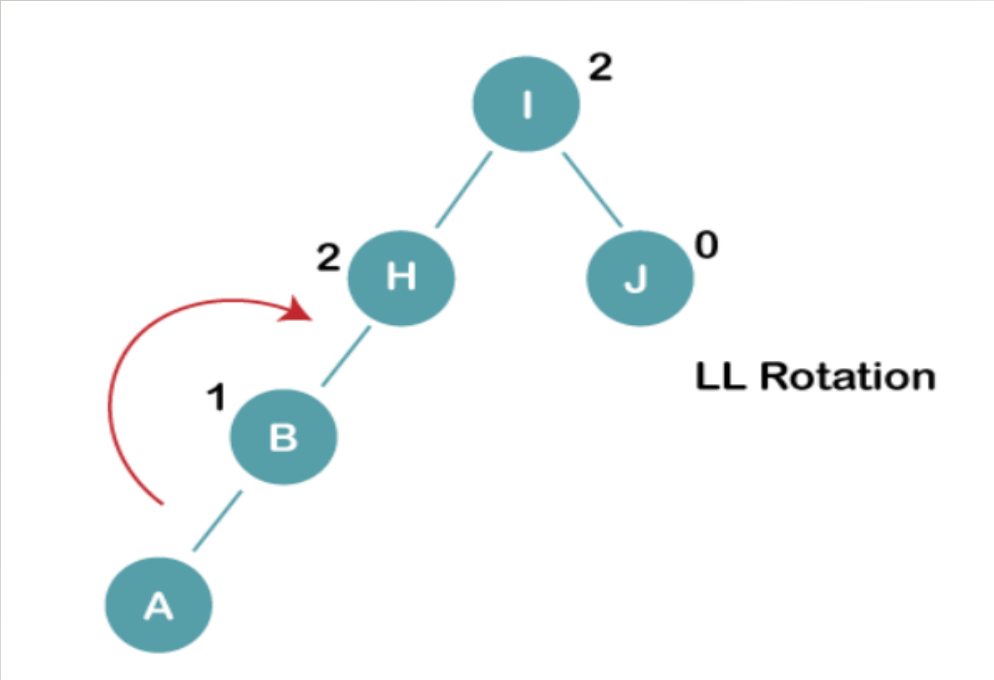
1. Insert H, I, J



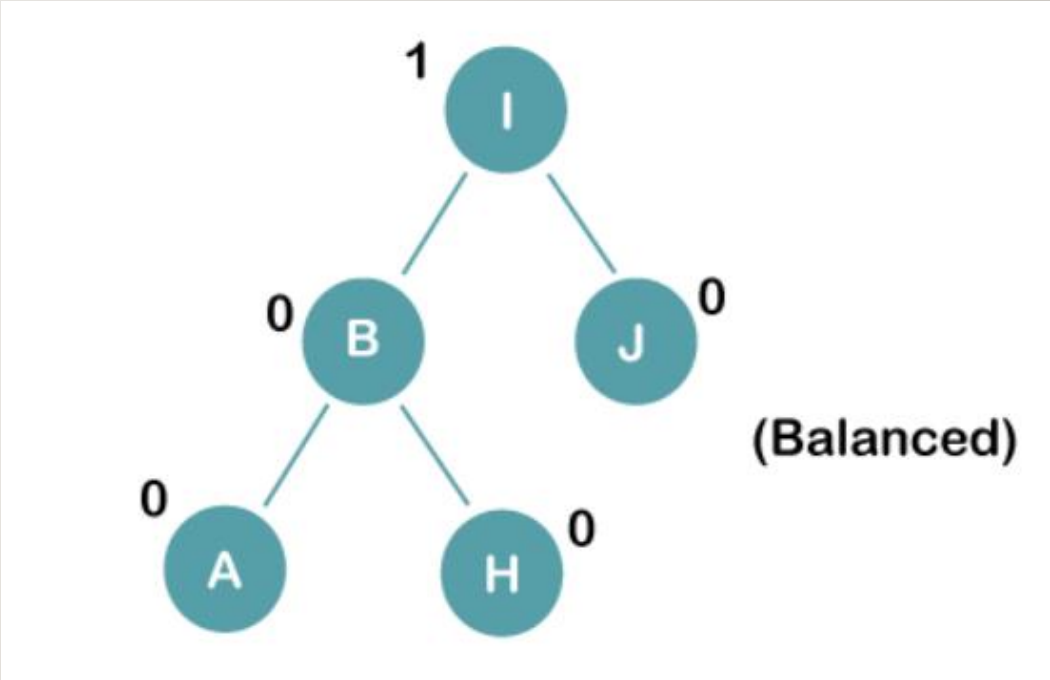
The resultant balance tree is:



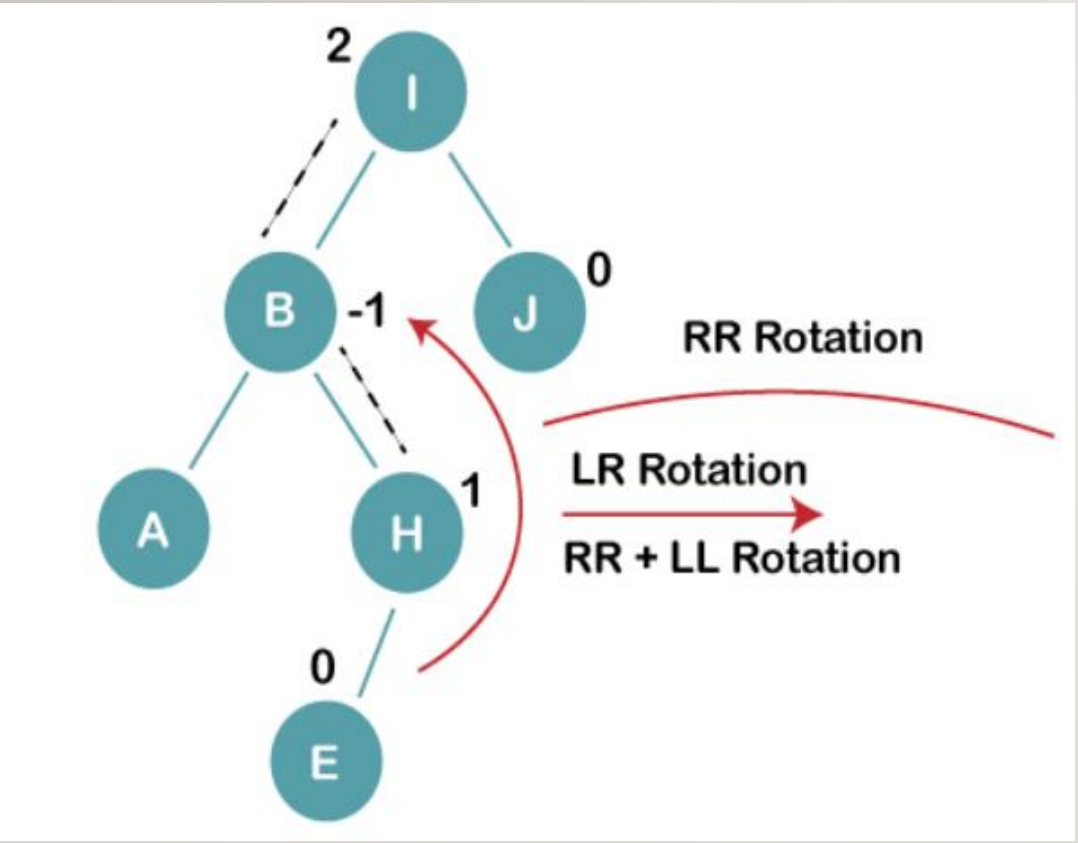
2. Insert B, A



The resultant balance tree is:



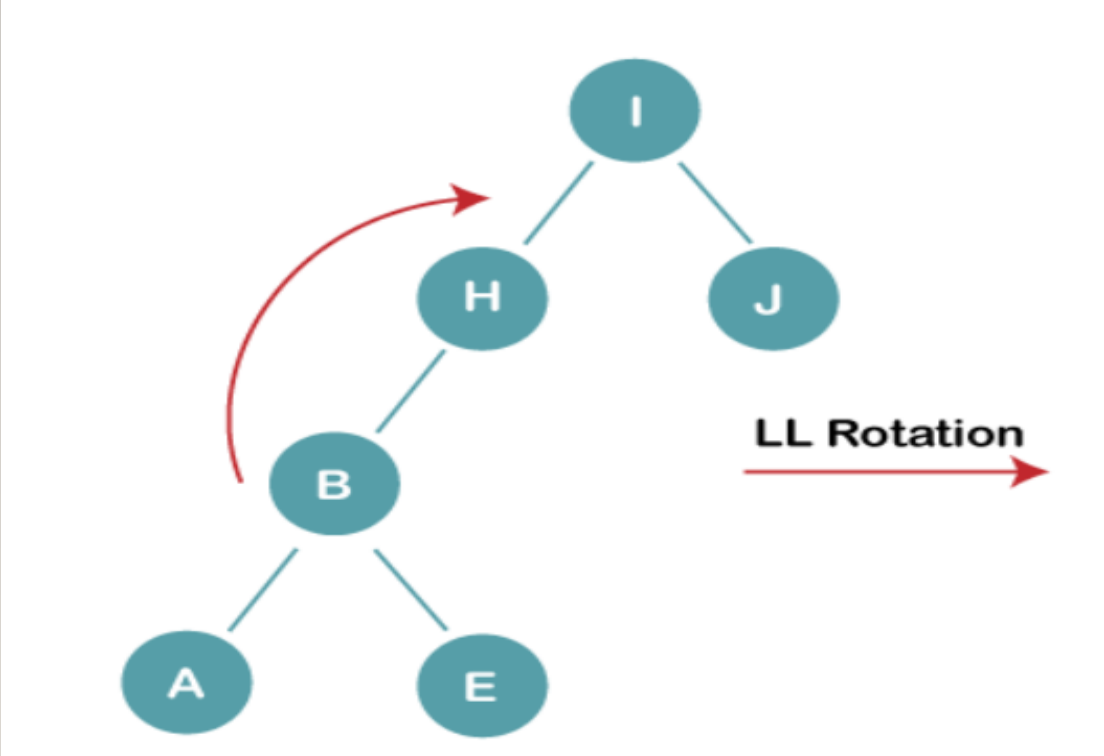
3. Insert E



On inserting E, BST becomes unbalanced as the Balance Factor of I is 2, since if we travel from E to I we find that it is inserted in the left subtree of right subtree of I, we will perform LR Rotation on node I. **LR = RR + LL** rotation

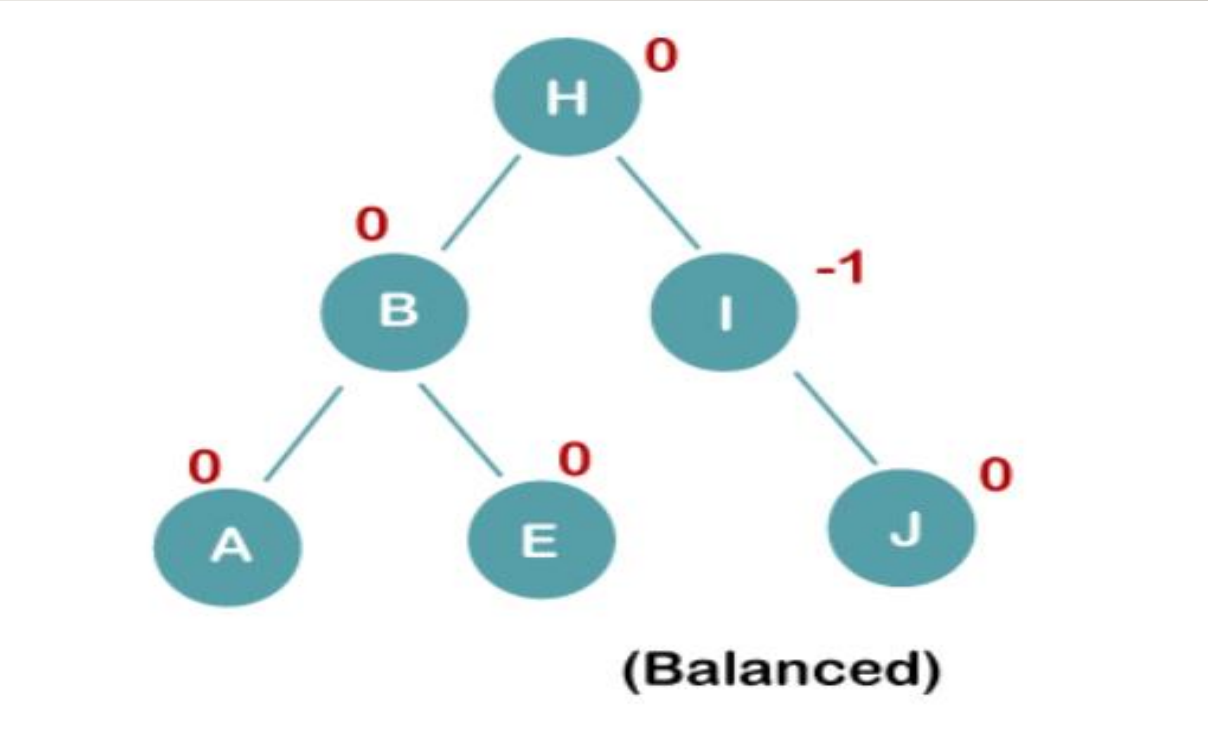
3 a) We first perform RR rotation on node B

The resultant tree after RR rotation is:

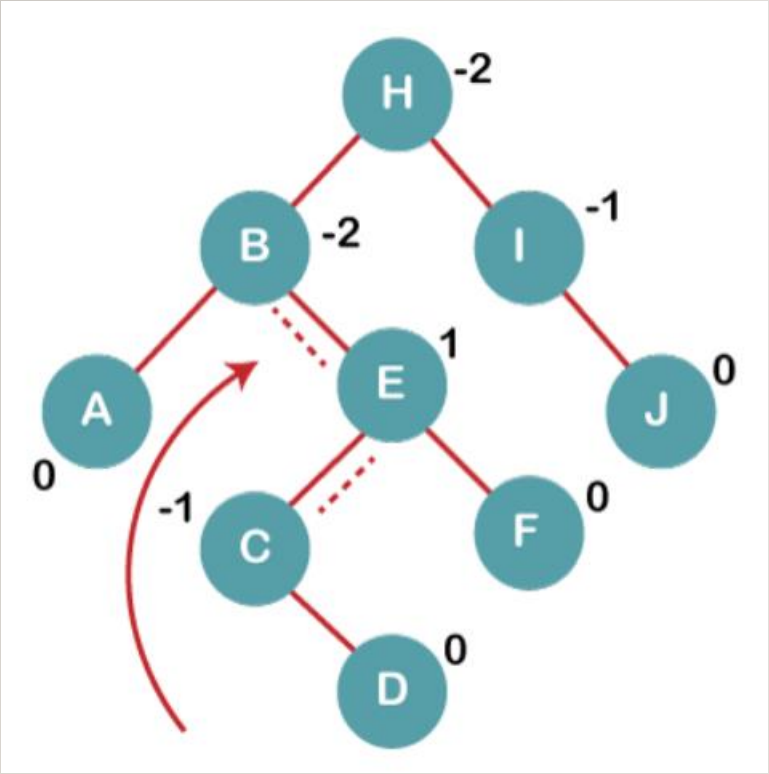


3b) We first perform LL rotation on the node I

The resultant balanced tree after LL rotation is:



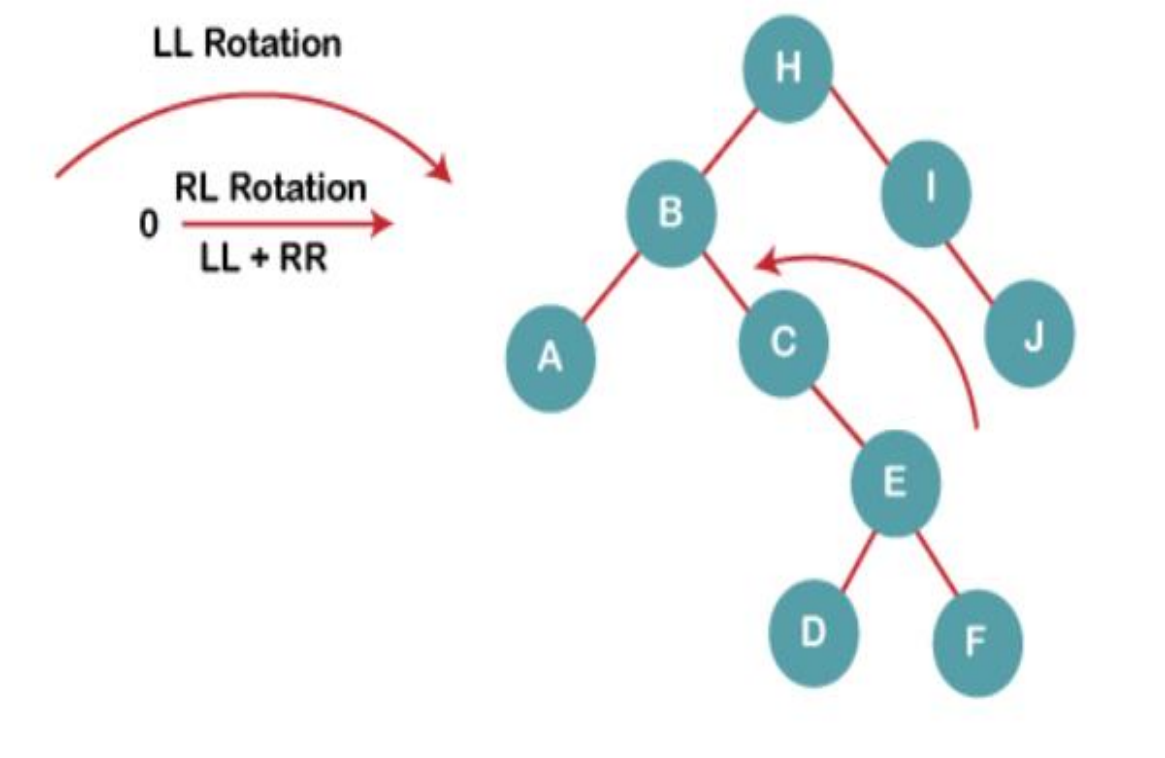
4. Insert C, F, D



On inserting C, F, D, BST becomes unbalanced as the Balance Factor of B and H is -2, since if we travel from D to B we find that it is inserted in the right subtree of left subtree of B, we will perform RL Rotation on node I. RL = LL + RR rotation.

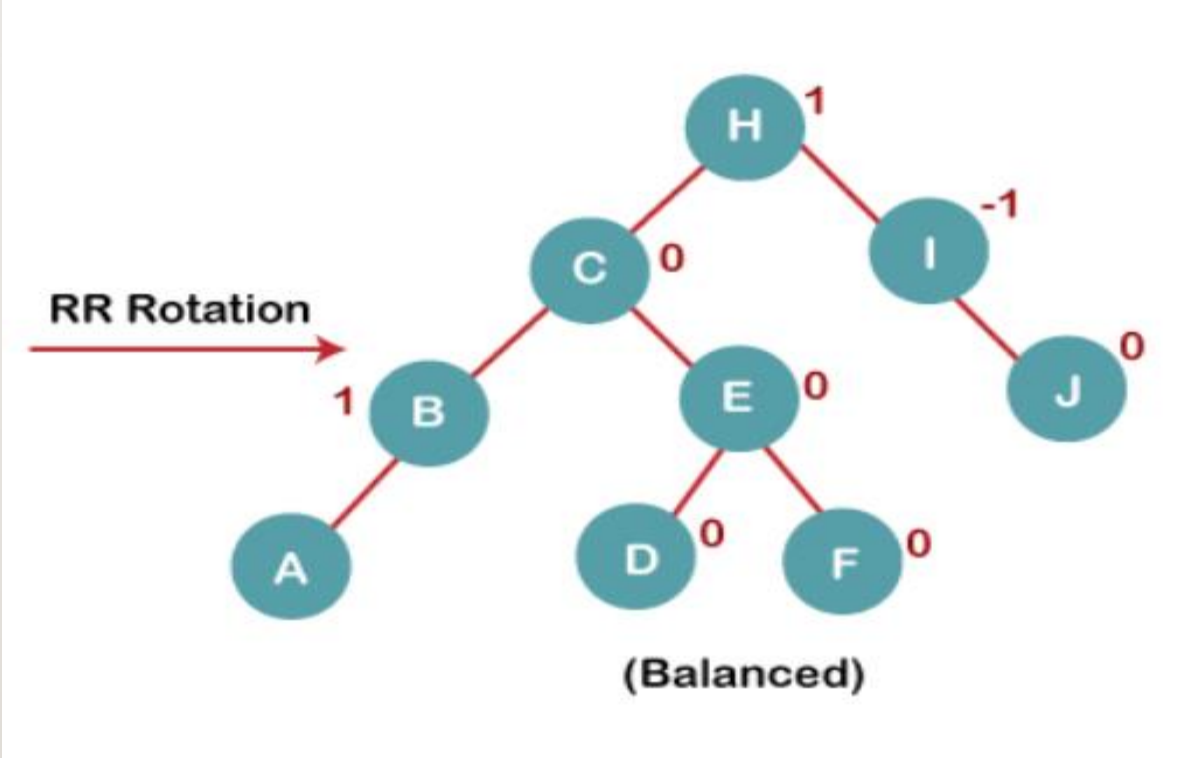
4a) We first perform LL rotation on node E

The resultant tree after LL rotation is:

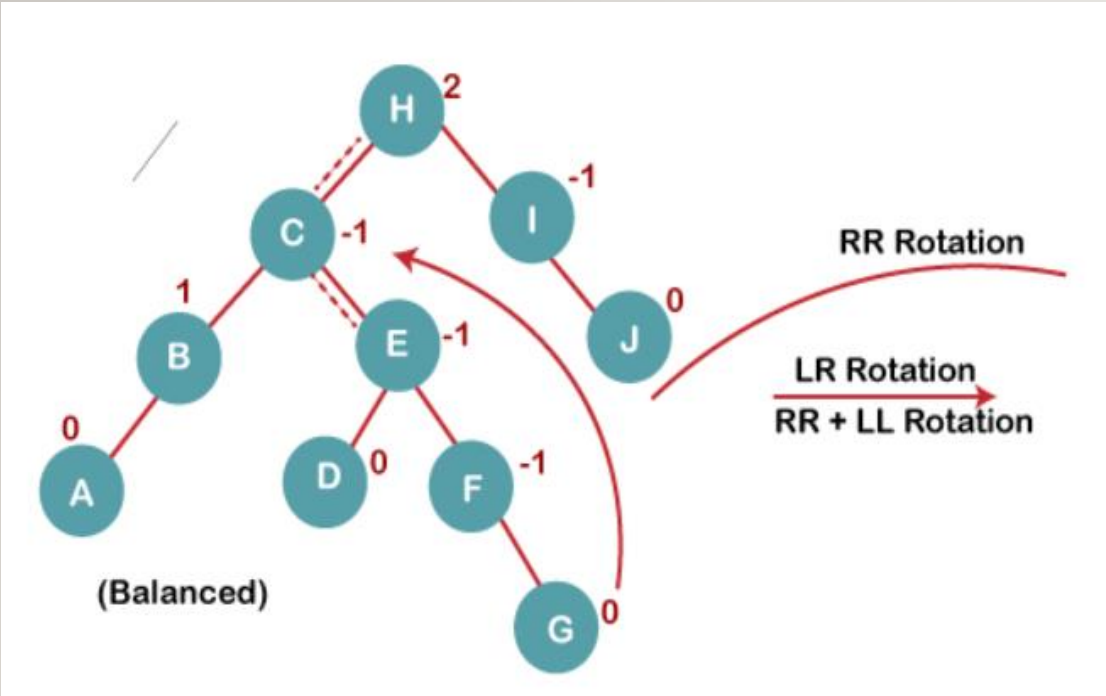


4b) We then perform RR rotation on node B

The resultant balanced tree after RR rotation is:



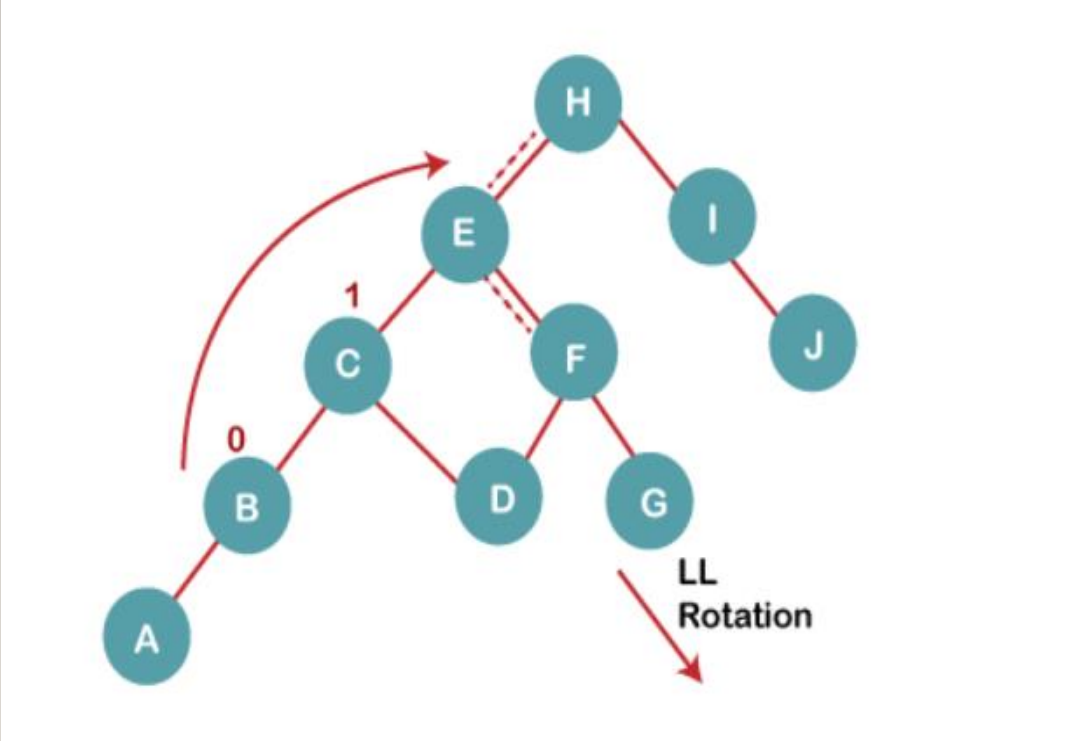
5. Insert G



On inserting G, BST become unbalanced as the Balance Factor of H is 2, since if we travel from G to H, we find that it is inserted in the right subtree of left subtree of H, we will perform LR Rotation on node I. LR = RR + LL rotation.

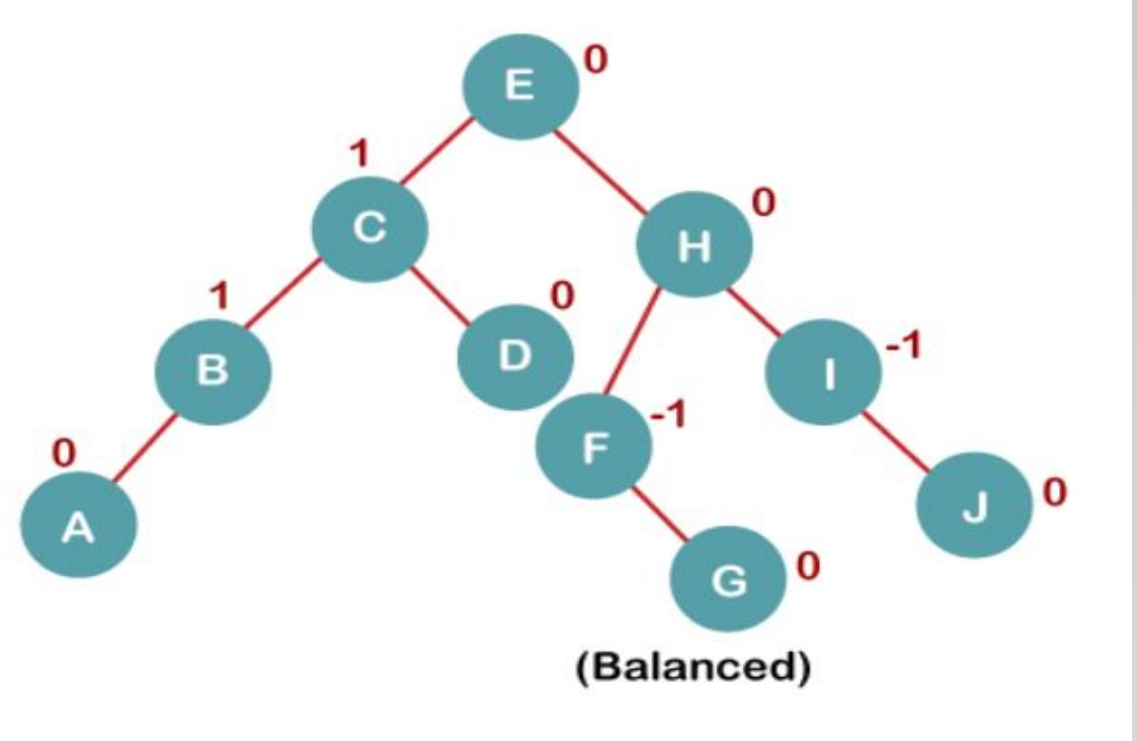
5 a) We first perform RR rotation on node C

The resultant tree after RR rotation is:

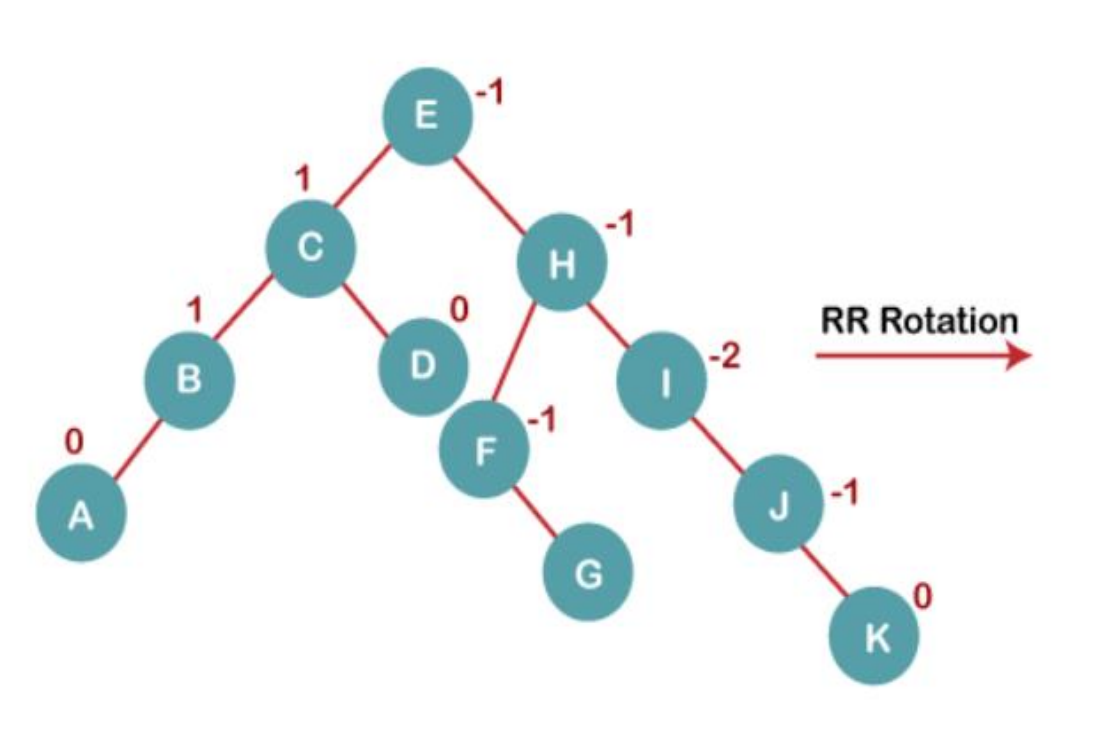


5 b) We then perform LL rotation on node H

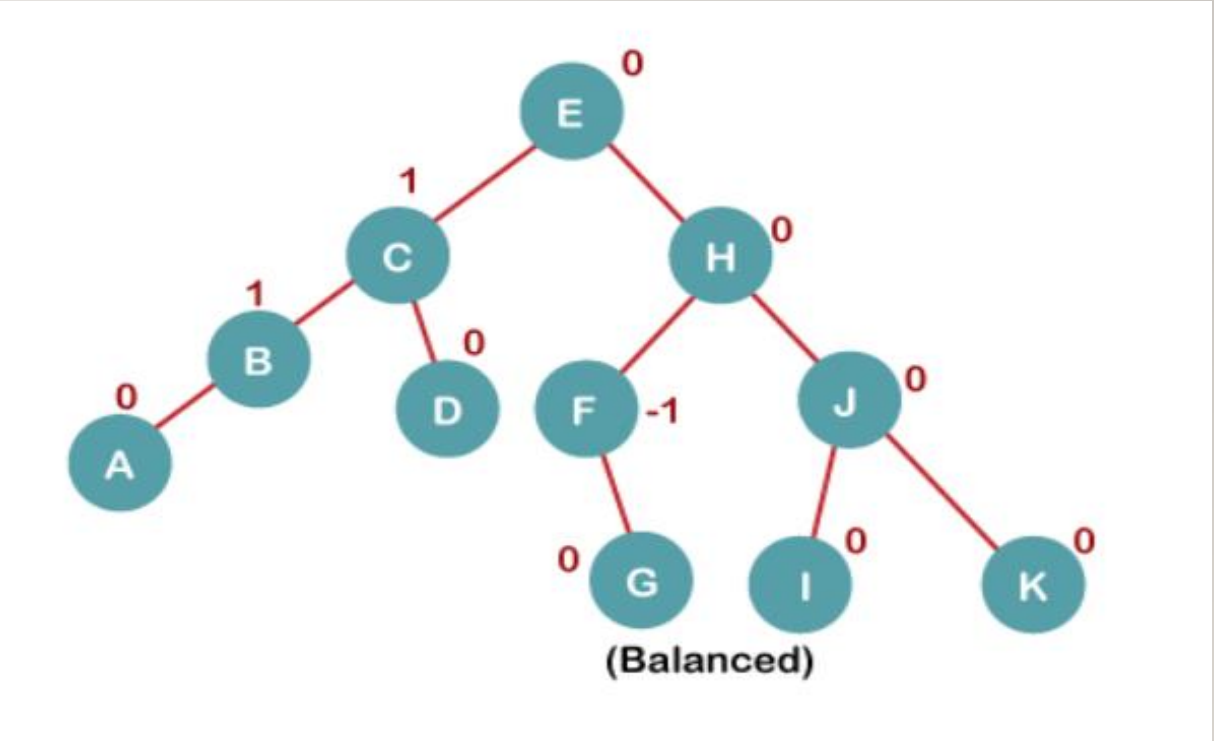
The resultant balanced tree after LL rotation is:



6. Insert K

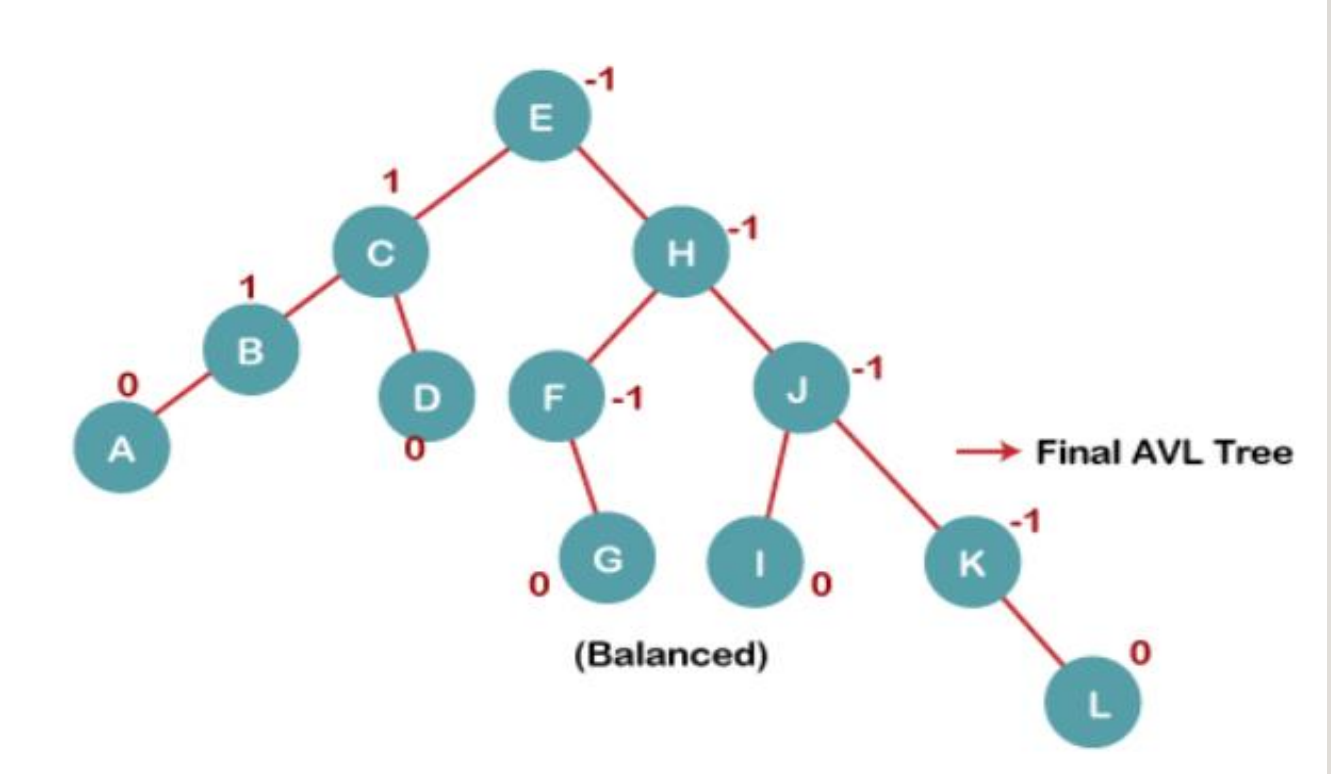


The resultant balanced tree after RR rotation is:



On inserting K, BST becomes unbalanced as the Balance Factor of I is -2. Since the BST is right-skewed from I to K, hence we will perform RR Rotation on the node I.

7. Insert L



On inserting the L tree is still balanced as the Balance Factor of each node is now either, -1, 0, +1. Hence the tree is a Balanced AVL tree

Construct an AVL tree by inserting the following elements in the giver order:

63, 9, 19, 27, 18, 108, 99, 81