



# **SNS COLLEGE OF ENGINEERING**

Kurumbapalayam (Po), Coimbatore - 641 107

**An Autonomous Institution**

Accredited by NBA - AICTE and Accredited by NAAC - UGC with 'A' Grade  
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai



## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING-IOT Including CS&BCT**

**COURSE NAME : 19CS307- DATA STRUCTURES**

**II YEAR / III SEMESTER**

**Unit III- NON LINEAR DATA STRUCTURES - Tree**

**Topic 7 : AVL Trees**



## Problem



- Construct an AVL tree having the following elements  
**H, I, J, B, A, E, C, F, D, G, K, L**



# AVL tree



- AVL tree is a binary search tree in which the difference of heights of left and right subtrees of any node is less than or equal to one. The technique of balancing the height of binary trees was developed by Adelson, Velskii, and Landi and hence given the short form as AVL tree or Balanced Binary Tree.



## AVL Tree-Cont

An AVL tree can be defined as follows:

Let  $T$  be a non-empty binary tree with  $T_L$  and  $T_R$  as its left and right subtrees. The tree is height balanced if:

$T_L$  and  $T_R$  are height balanced

$h_L - h_R \leq 1$ , where  $h_L - h_R$  are the heights of  $T_L$  and  $T_R$

The Balance factor of a node in a binary tree can have value 1, -1, 0, depending on whether the height of its left subtree is greater, less than or equal to the height of the right subtree.



## AVL Tree -Cont..



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.

$$\text{Balance Factor (k)} = \text{height (left(k))} - \text{height (right(k))}$$

### NOTE:

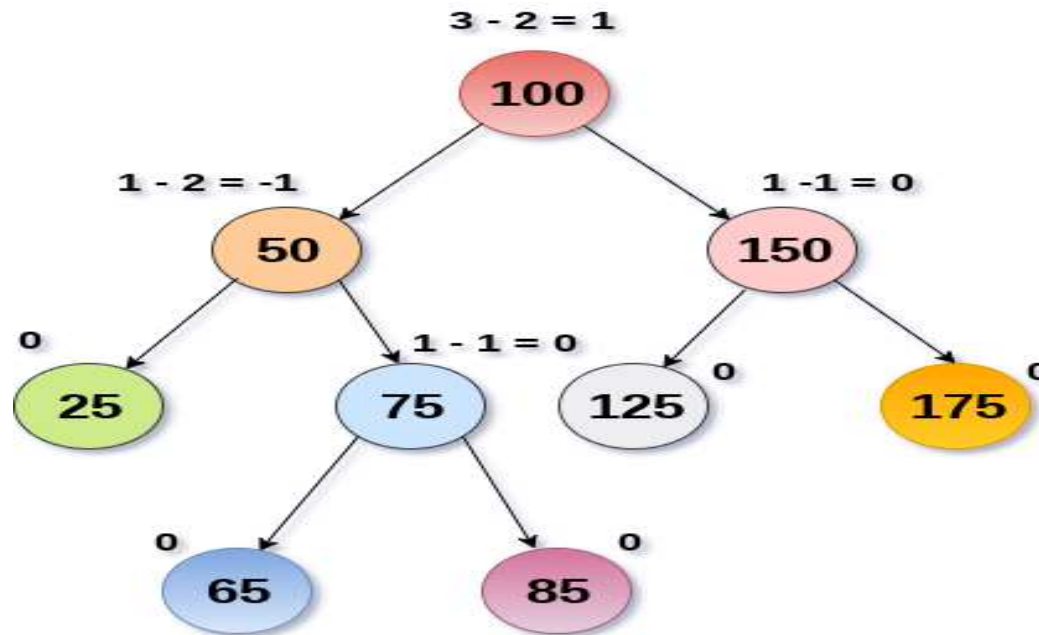
1. If balance factor of any node is 1, it means that the left sub-tree is one level higher than the right sub-tree.
2. If balance factor of any node is 0, it means that the left sub-tree and right sub-tree contain equal height.
3. If balance factor of any node is -1, it means that the left sub-tree is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



## AVL Tree -Cont..



An AVL tree is given in the following figure. We can see that, balance factor associated with each node is in between -1 and +1. therefore, it is an example of AVL tree.



AVL Tree



## AVL Tree -Cont..



### AVL Rotations

We perform rotation 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.

The first two rotations LL and RR are single rotations and the next two rotations LR and RL are double rotations. For a tree to be unbalanced, minimum height must be at least 2, Let us understand each rotation

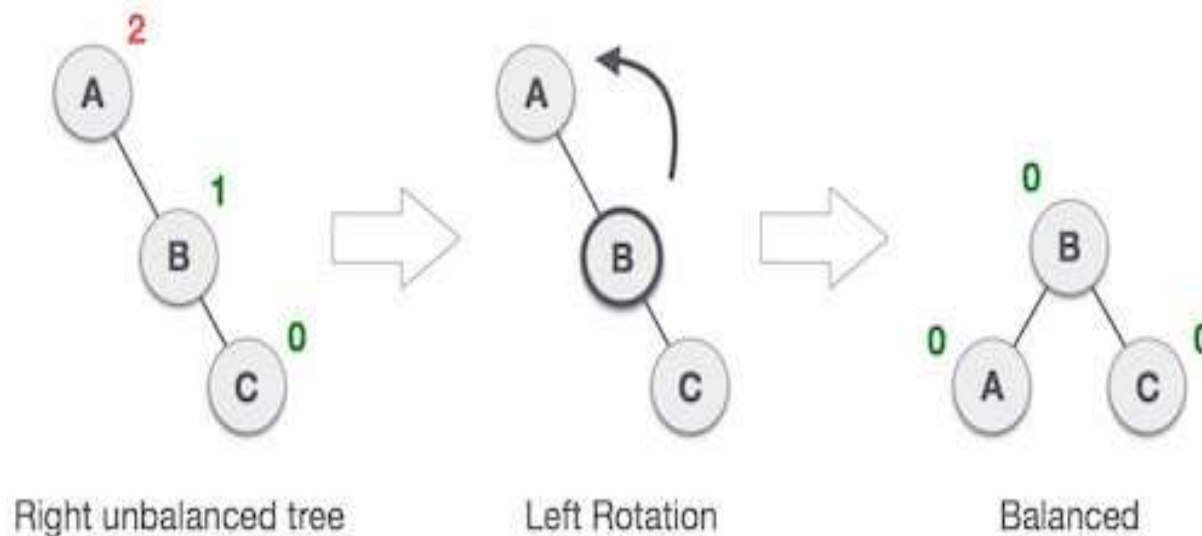


## AVL Tree -Cont..

### 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.





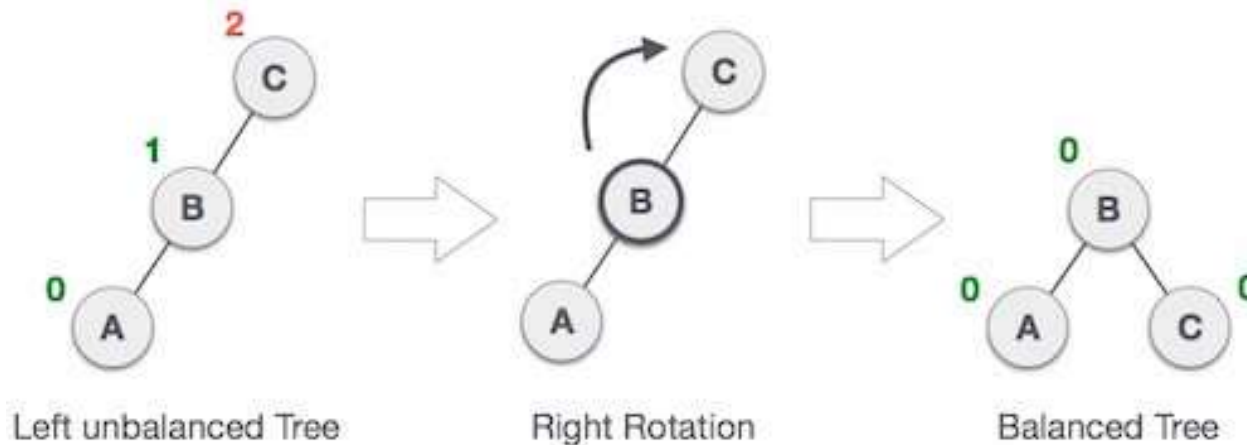


## AVL Tree -Cont..



### 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 A.



## AVL Tree -Cont..



### LR Rotation

Double rotations are bit tougher than single rotation which has already explained above. 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.



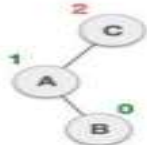
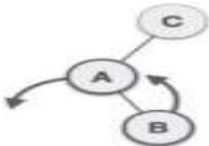
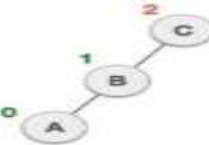
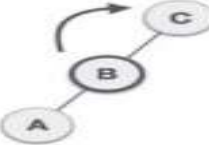
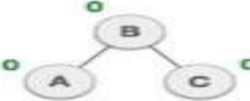
# AVL TREE -Cont..

## LR Rotation



### Left-Right Rotation

Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

State	Action
	A node has been inserted into the right subtree of the left subtree. This makes C an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of C. This makes A, the left subtree of B.
	Node C is still unbalanced, however now, it is because of the left-subtree of the left-subtree.
	We shall now right-rotate the tree, making B the new root node of this subtree. C now becomes the right subtree of its own left subtree.
	The tree is now balanced.



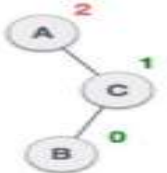
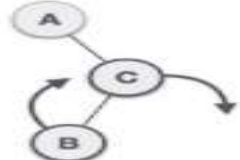
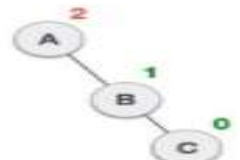
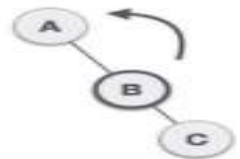
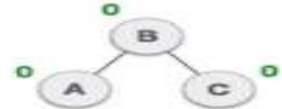
# AVL TREE -Cont..

## RR Rotation



### Right-Left Rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

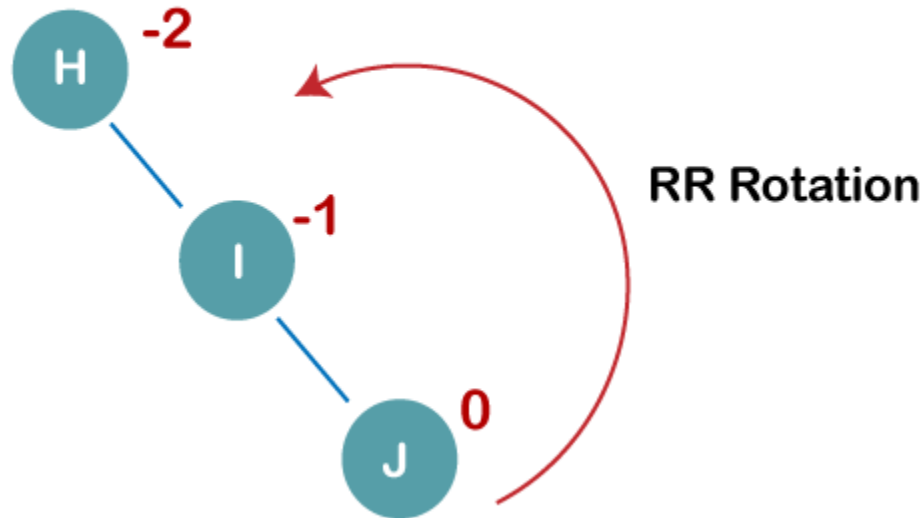
State	Action
	A node has been inserted into the left subtree of the right subtree. This makes A, an unbalanced node with balance factor 2.
	First, we perform the right rotation along C node, making C the right subtree of its own left subtree B. Now, B becomes the right subtree of A.
	Node A is still unbalanced because of the right subtree of its right subtree and requires a left rotation.
	A left rotation is performed by making B the new root node of the subtree. A becomes the left subtree of its right subtree B.
	The tree is now balanced.



## AVL Tree -Cont..



### 1. Insert H, I, J



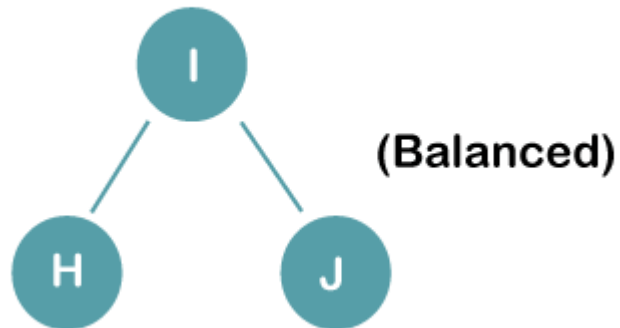
On inserting the above elements, especially in the case of H, the BST becomes unbalanced as the Balance Factor of H is -2. Since the BST is right-skewed, we will perform RR Rotation on node H.



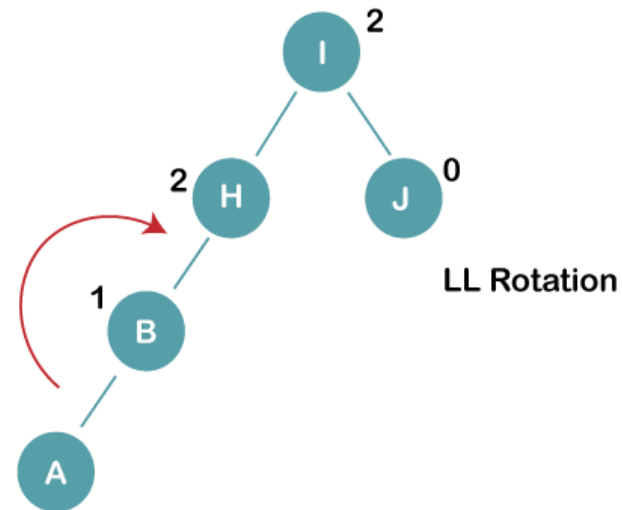
## AVL Tree -Cont..



### 1. The resultant balance tree is:



### 2. Insert B, A.

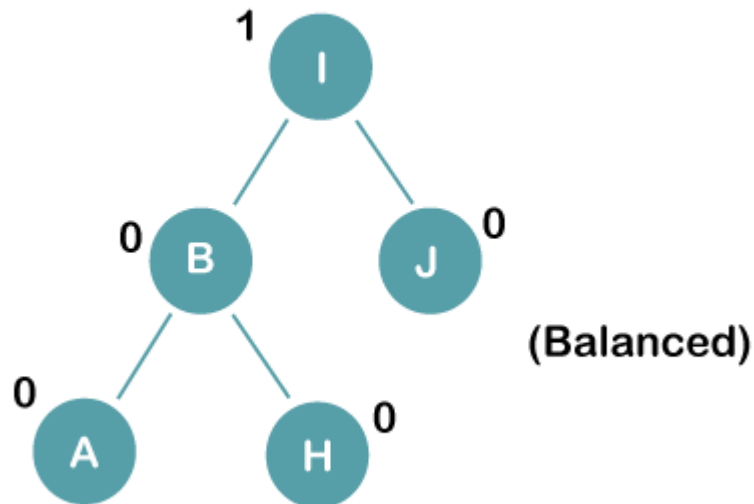


On inserting the above elements, especially in case of A, the BST becomes unbalanced as the Balance Factor of H and I is 2, we consider the first node from the last inserted node i.e. H. Since the BST from H is left-skewed, we will perform LL Rotation on node H.



## AVL Tree -Cont..

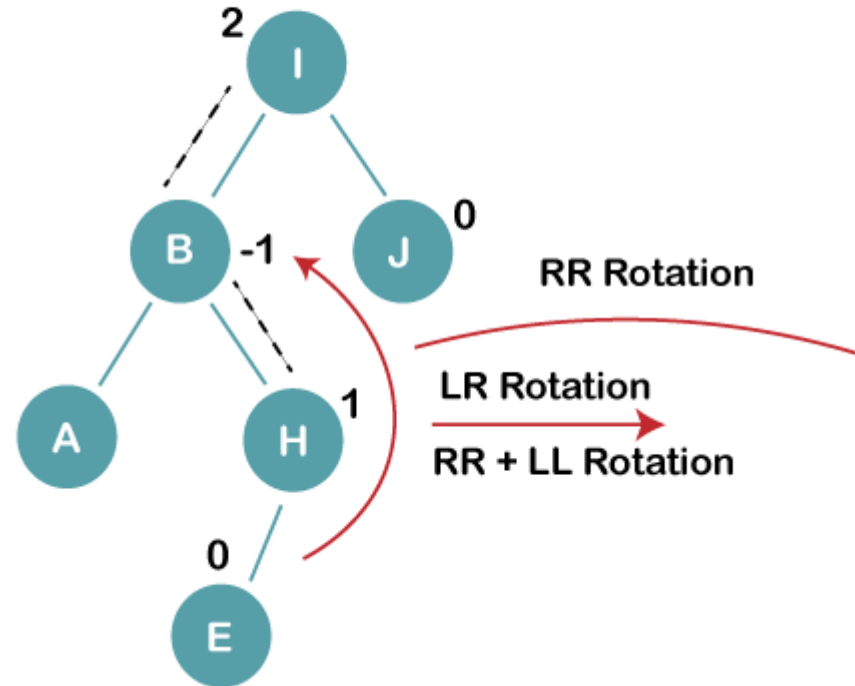
1. The resultant balance tree is:





## AVL Tree -Cont..

### 1. 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

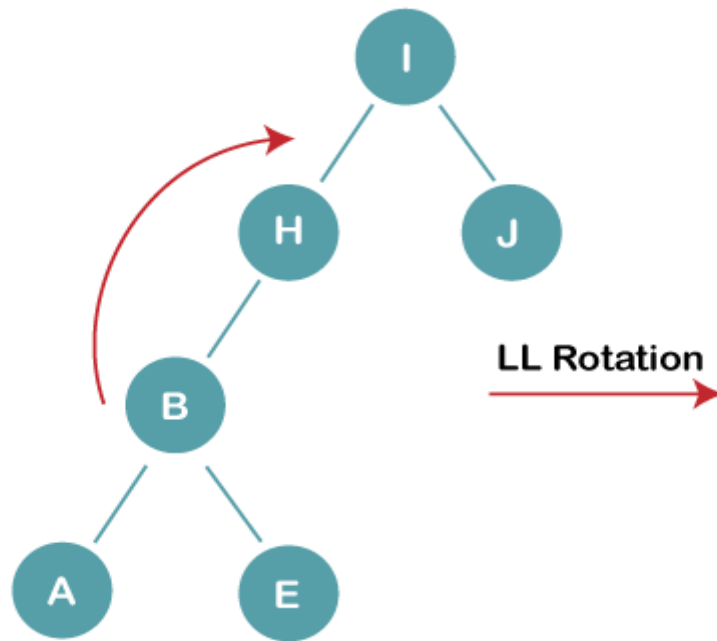




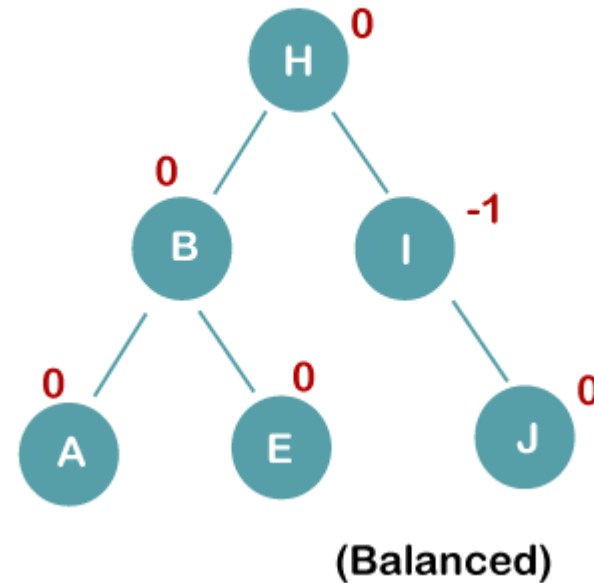
## AVL Tree -Cont..



### 1. We first perform RR rotation on node B



We first perform LL rotation on the node I  
The resultant balanced tree after LL rotation is:





# Activity



## MCQ

1. What is an AVL tree?
  - a) a tree which is balanced and is a height balanced tree
  - b) a tree which is unbalanced and is a height balanced tree
  - c) a tree with three children
  - d) a tree with atmost 3 children
2. Why to prefer red-black trees over AVL trees?
  - a) Because red-black is more rigidly balanced
  - b) AVL tree store balance factor in every node which costs space
  - c) AVL tree fails at scale
  - d) Red black is more efficient



# Advantages



- **Better search times** for key
- The height of the AVL tree is always balanced. The height never grows beyond  $\log N$ , where  $N$  is the total number of nodes in the tree.
- It gives better search time complexity when compared to simple Binary Search trees.
- AVL trees have self-balancing capabilities.



## Disadvantages



- Longer running times for the *insert* and *remove* operations(As we will see, the *insert* and *remove* operation must re-balance the AVL tree....)
- restructuring is an expensive operation



# Assessment 1



1. List out the advantages of AVL tree

- a) \_\_\_\_\_
- b) \_\_\_\_\_
- c) \_\_\_\_\_
- d) \_\_\_\_\_

2. Identify the disadvantages of AVL tree

- a) \_\_\_\_\_
- b) \_\_\_\_\_
- c) \_\_\_\_\_
- d) \_\_\_\_\_





# REFERENCES



1. M. A. Weiss, “Data Structures and Algorithm Analysis in C”, Pearson Education, 8<sup>th</sup> Edition, 2007. [Unit I, II, III, IV,V]
2. A. V. Aho, J. E. Hopcroft and J. D. Ullman, “Data Structures and Algorithms”, Pearson Education, 2<sup>nd</sup> Edition, 2007 [Unit IV].
3. A.M.Tenenbaum, Y. Langsam and M. J. Augenstein, “Data Structures using C”, Pearson Education, 1<sup>st</sup> Edition, 2003.(UNIT I,II,V)
4. <https://www.javatpoint.com/avl-tree>

## THANK YOU