# AVL TREES

## BALANCED TREES (OR) HEIGHT BALANCED TREES

* Trees whose height in the worst case turns out to be O(log n) are called balanced trees. Example: AVL trees,2-3 trees, Red Black Trees, Splay trees.
* We can guarantee O(log n) performance for the search ,insert and delete operations of a search tree by ensuring that the search tree height is always O(log n).
* Since trees are balanced by working with their height, they are also known as height balanced trees.
* ***Definition:*** A height balanced tree T is a binary tree that may be empty. If non empty, then
  1. Its left and right sub trees must also be height balanced trees
  2. The height of left and right sub trees differ by at most 1 i.e |HL-HR|<=1
* ***Examples of Balanced trees:*** AVL trees, Splay trees, Red Black trees, B trees and B+ trees.

## AVL TREES

* An AVL tree is a binary search tree in which difference between the heights of the left and right sub trees will be either -1, 0 or 1.
* Therefore an AVL tree is a balanced/height balanced binary search tree.
* ***Definition:*** An empty binary tree is an AVL tree. If T is a non empty binary tree with TL and TR are its left and right sub trees, then T is an AVL tree if and only if

1. TL and TR are AVL trees and
2. |HL-HR|<=1 where HL and HR are the height of left sub trees(TL) and right sub tree (TR) respectively of T.

Balance factor (bf) is associated with every node is an AVL tree which may be either 0 or +1 or -1.

* ***Balance factor:*** The balance factor bf(u) of a node u is defined as the height of the left sub tree of u minus the height of the right sub tree of u.
* **bf(u)=(h(uL)-h(uR))** where h(uL) and h(uR) are the height of the left and right sub trees of the node u respectively.

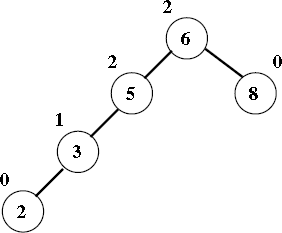
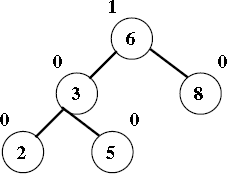
### Properties of AVL Tree:

1. The minimum height of an AVL tree with n elements is O(log n)
2. The maximum height of an AVL tree with n elements is 1.44logn

2

1. An n-element AVL tree can be searched in O(height)= O(log n) time.
2. A new element can be inserted into an n-element AVL search tree so that the result is an n+1 element AVL tree and such an insertion can be done in O(log n) time.
3. An element can be deleted from an n-element AVL search tree so that the result is an n-1 element AVL tree and such a deletion can be done in O(log n) time.

### Examples:



**AVL tree Not an AVL tree**

## OPERATIONS ON AN AVL TREES

The basic operations performed on an AVL tree are

1. Searching an element
2. Inserting a new element
3. Deleting an element

## INSERTION OPERATION ON AVL TREES

* In AVL Tree, new node is always inserted as a leaf node. The insertion operation is performed as follows....
  + **Step 1:** Insert the new element into the tree using Binary Search Tree insertion logic.
  + **Step 2:** After insertion, check the **Balance Factor** of every node.
  + **Step 3:** If the **Balance Factor** of every node is **0 or 1 or -1** then go for next operation.
  + **Step 4:** If after insertion the balance factor of any of the nodes turns out to be anything other than 0, +1 or -1, then the tree is said to be unbalanced.
* To balance the tree we perform rotations
* ***Rotations:*** Rotations are mechanisms which shift some of the sub trees of the unbalanced tree either to left or right to obtain a balanced tree.
* There are four rotations and they are classified into two types.

**RR ROTATION**

**(Single Left Rotation)**

**SINGLE ROTATION**

**LL ROTATION**

**(Single Right Rotation)**

**RL ROTATION**

**(Right Left Rotation)**

**DOUBLE ROTATION**

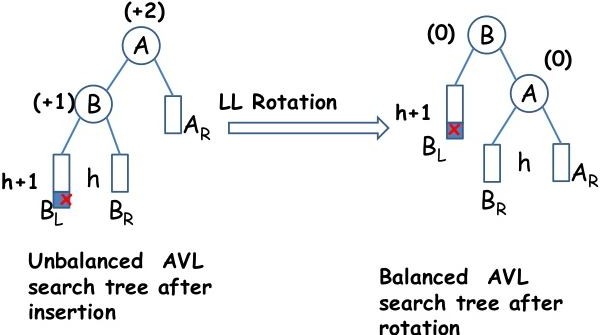
**LR ROTATION**

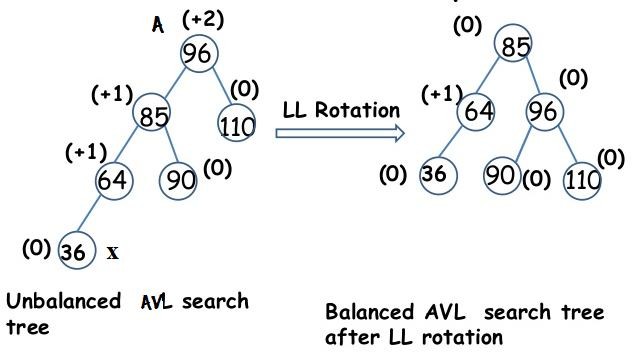
**(Left Right Rotation)**

## LL Rotation (Single Right Rotation)

* + The new node ‘x’ is inserted in the Left sub tree of left child of node A. As a result, the balance of A becomes +2.
  + To restore balance at A Single right rotation need to be applied at A.

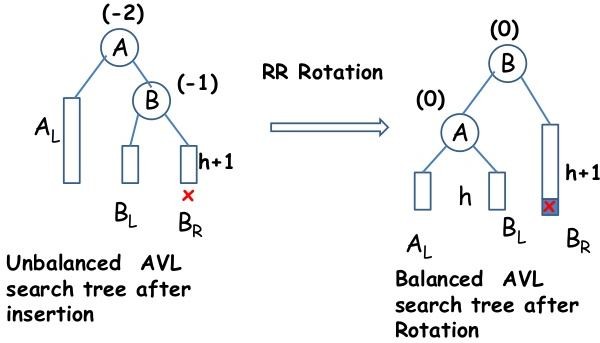
**General Notation:**



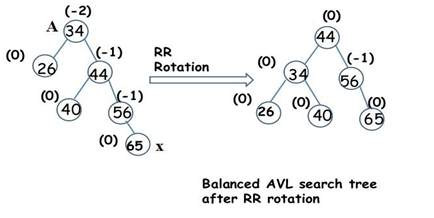
**Example:**

## RR Rotation (Single Left Rotation)

* + The new node ‘x’ is inserted in the Right sub tree of right child of node A. As a result, the balance of A becomes -2.
  + To restore balance at A Single left rotation need to be applied at A.

**General Notation:**

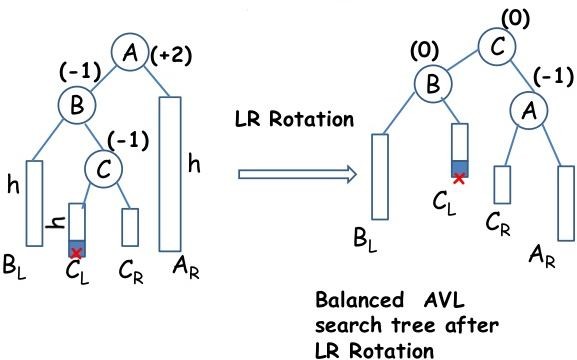
**Example:**

****

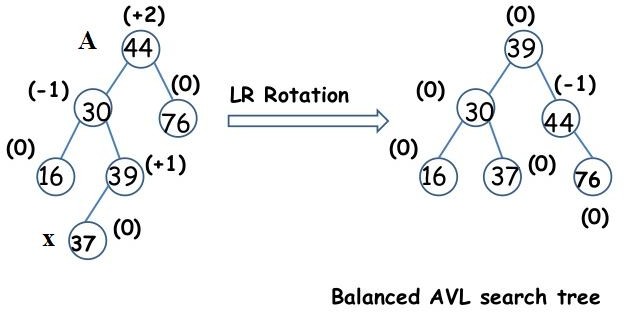
## LR Rotation (Left Right Rotation)

* + Imbalance occurred at A due to the insertion of node x in the right sub tree of left child of node A.
  + LR rotation involves sequence of two rotations

1. Single Left rotation/RR rotation
2. Single Right rotation/LL rotation

**General Notation:**

**Example:**

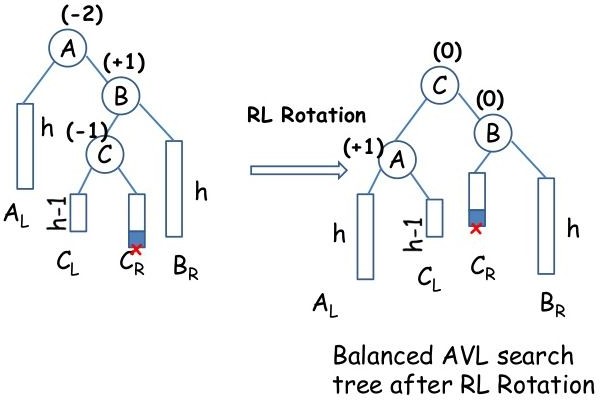


## RL Rotation (Right Left Rotation)

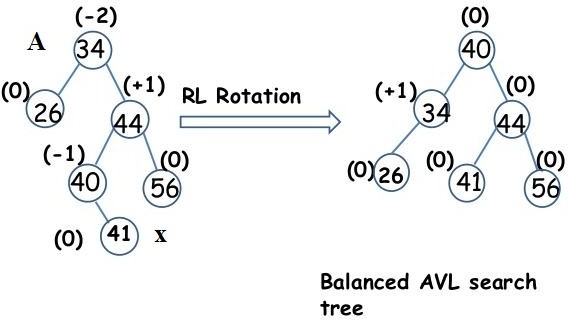
* + Imbalance occurred at A due to the insertion of node x in the Left sub tree of right child of node A.
  + RL rotation involves sequence of two rotations

1. Single Right rotation/LL rotation
2. Single Left rotation/RR rotation

**General Notation:**



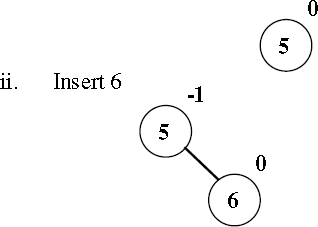
**Example:**

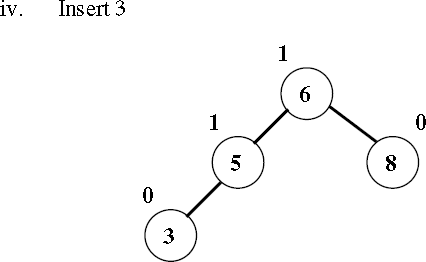
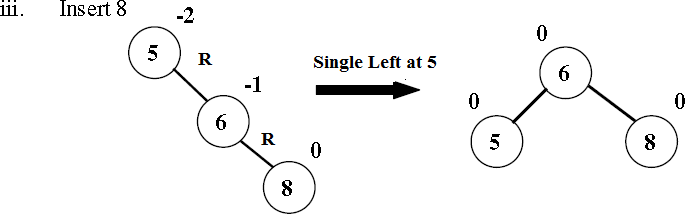


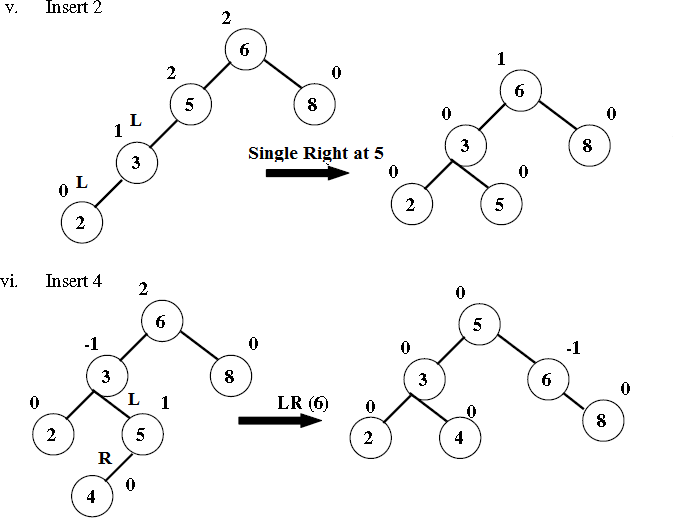
## CONSTRUCTION OF AN AVL TREE

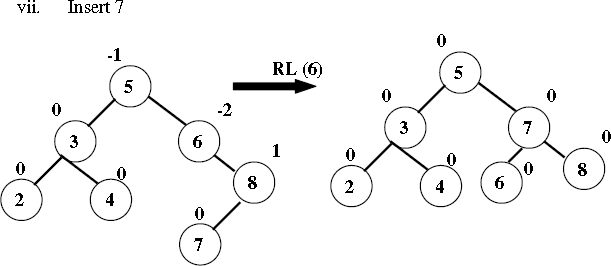
Construct AVL tree for the list by successive insertion: 5, 6, 8, 3, 2, 4, 7

1. Insert 5 into empty tree









## DELETION OPERATION ON AVL TREES

The sequence of steps to be followed in deletion are:

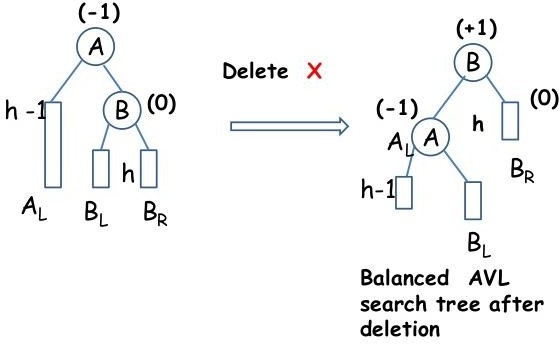
* 1. Initially, the AVL tree is searched to find the node to be deleted
  2. If the search is successful, delete the node. The following are the 3 possibilities for the node ‘x’ that is to be deleted
     1. x is a leaf – In this case the leaf node is discarded
     2. x has exactly one non empty sub tree – If x has no parent, the root of its sub tree becomes the new search tree root. If x has a parent P, then we change the pointer from parent p so that it points to x’s only child
     3. x has two non empty sub trees – x is replaced with either the largest element in the left sub tree or the smallest element in its right sub tree.
  3. After deletion of node, check the balance factor of each node
  4. Rebalance the tree if the tree is unbalanced. For this AVL tree deletion rotations are used.
* On deletion of a node x from the AVL tree. Let A be the closest ancestor node on the path from x to the root node, with a balance factor of +2 or -2. To restores balance at node A, we classify the type of imbalance as follows:

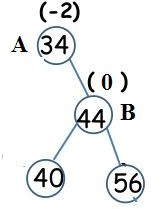
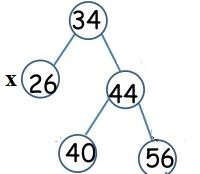
## L – type imbalance:

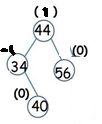
* + The imbalance is of type L if the deletion took place from A’s left sub tree.
  + The balance factor of A = **-2**
  + A has a right sub tree with root B
  + L-type imbalance is sub classified into types L0, L1 and L-1 depending on the balance factor of B

## L0 ROTATION

* + L0 imbalance occurs if the deletion takes place from the left sub tree of A and balance factor of B is **0**
  + L0 rotation is Single Left rotation, that is applied at node A

**General Notation:**

**Example:**

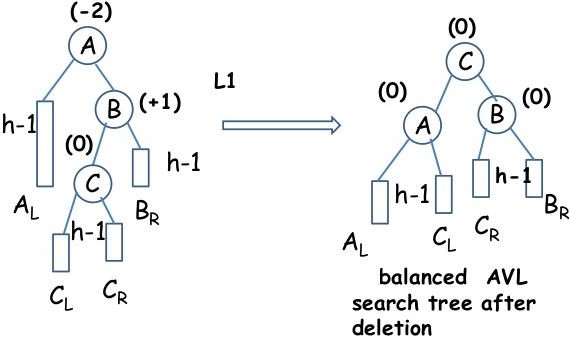
Before deletion After deleting 26 After L0 rotation

## L1 ROTATION

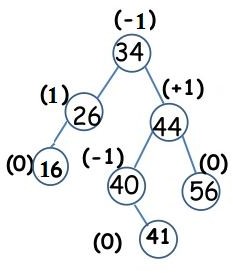
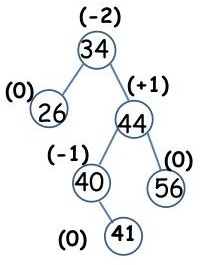
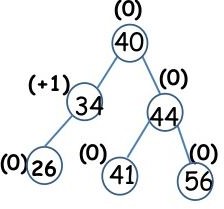
* + L1 imbalance occurs if the deletion takes place from the left sub tree of A and balance factor of B is **1**
  + L1 rotation is RL rotation, which involves 2 rotations:

1. Single Right
2. Single Left

**General Notation:**



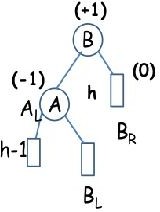
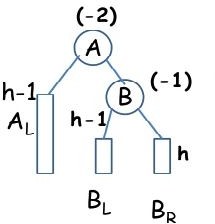
**Example:**

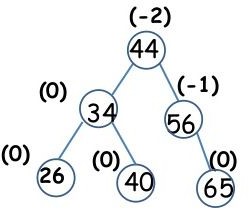
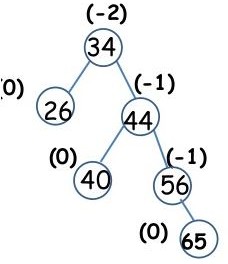
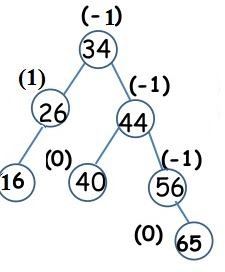
  

Before deletion After deleting 16 After L1 rotation

## L-1 ROTATION

* + L-1 imbalance occurs if the deletion takes place from the left sub tree of A and balance factor of B is **-1**
  + L-1 rotation is Single Left rotation, that is applied at node A

**General Notation:**

**Example:**

Before deletion After deleting 16 After L-1 rotation

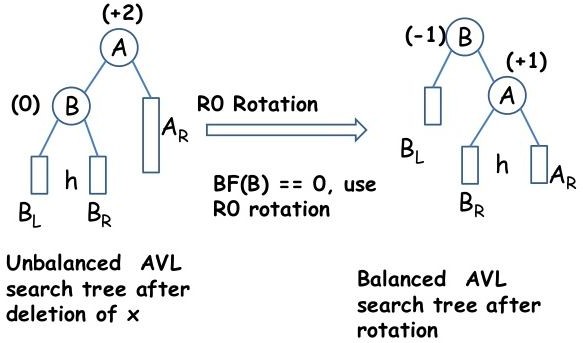
## R – type Imbalance:

* + - The imbalance is of type R if the deletion took place from A’s Right sub tree.
    - The balance factor of A = **2**
    - A has a left sub tree with root B
    - R-type imbalance is sub classified into types R0, R1 and R-1 depending on the balance factor of B

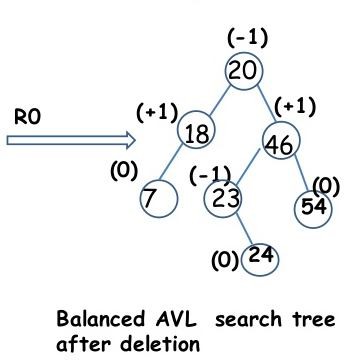
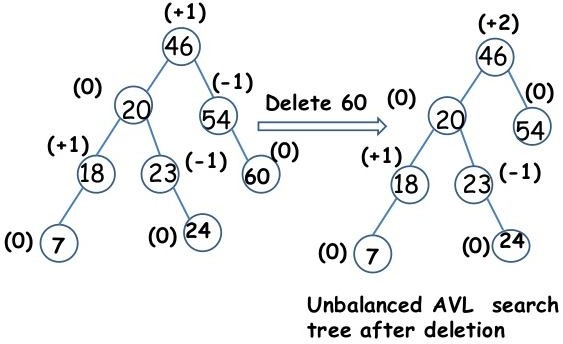
## R0 ROTATION

* + R0 imbalance occurs if the deletion takes place from the Right sub tree of A and balance factor of B is **0**
  + R0 rotation is Single Right rotation, that is applied at node A

**General Notation:**

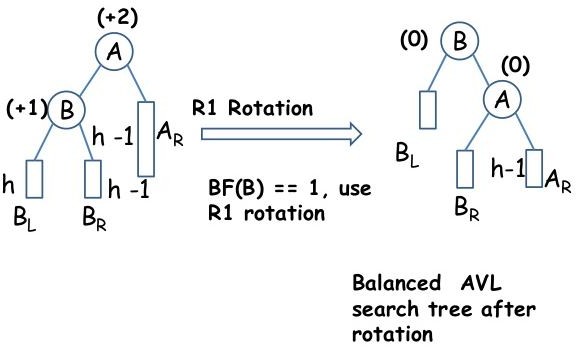


**Example:**

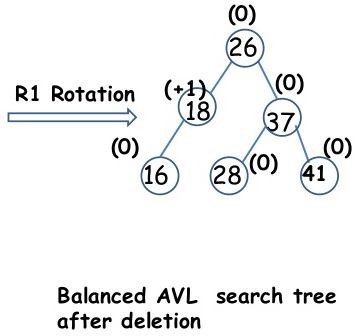
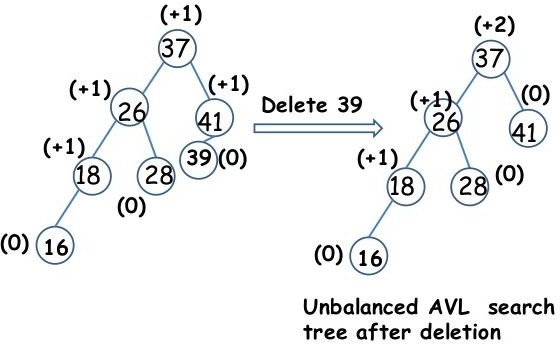


## R1 ROTATION

* + R1 imbalance occurs if the deletion takes place from the Right sub tree of A and balance factor of B is **1**
  + R1 rotation is Single Right rotation, that is applied at node A

**General Notation:**

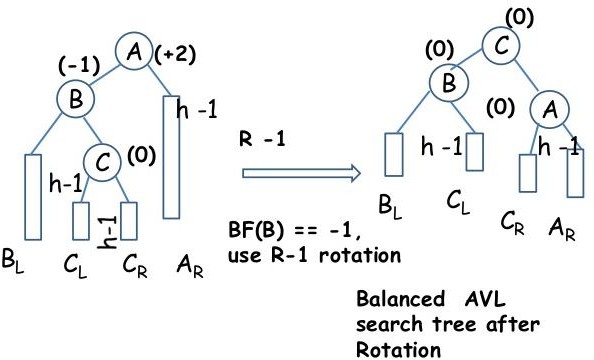
**Example:**

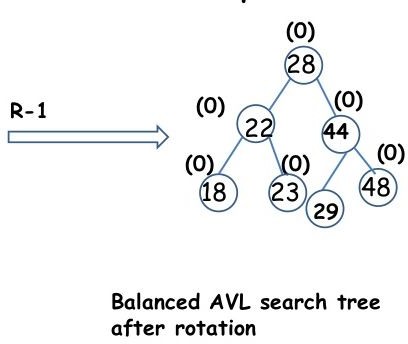
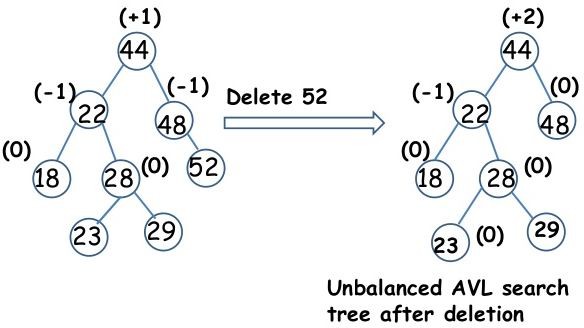


## R-1 ROTATION

* + R-1 imbalance occurs if the deletion takes place from the Right sub tree of A and balance factor of B is **-1**
  + R-1 rotation is LR rotation, which involves 2 rotations:

1. Single Left
2. Single Right

**General Notation:**

**Example:**