LAB SESSION 05:

**AVL TREE ROTATIONS**

# Pre-Lab:

1. Construct an AVL tree having the following elements:

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

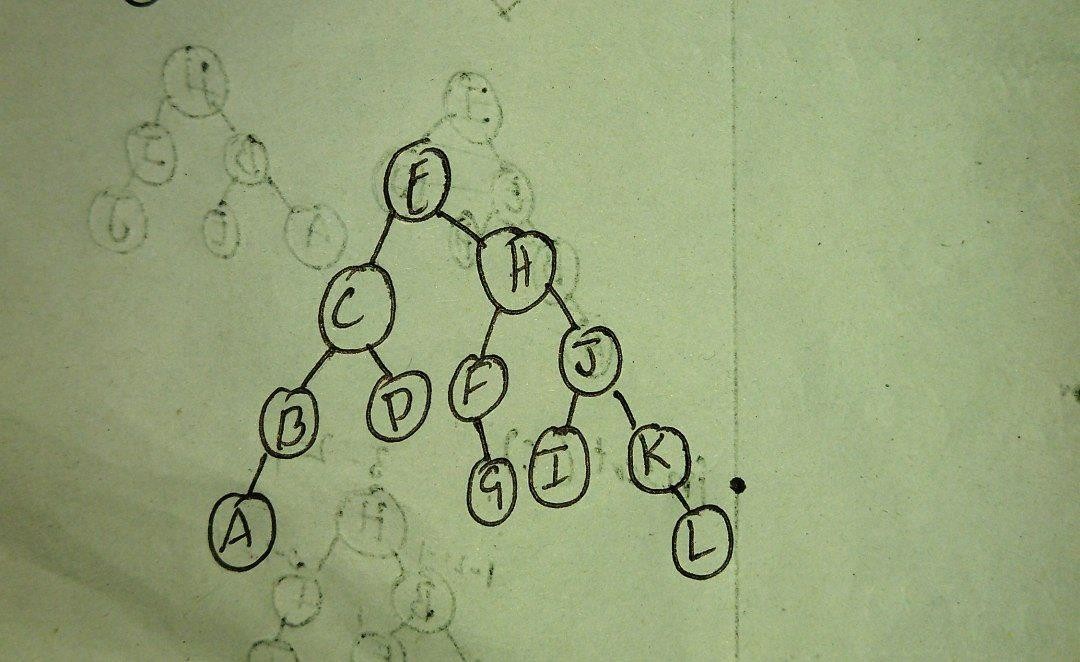
**Explain rotation in each step and write the final AVL tree.**

# Solution:

We use RL at the insertion of ‘B’ and ‘G’

We use LL at the insertion of ‘L’

Final AVL tree:

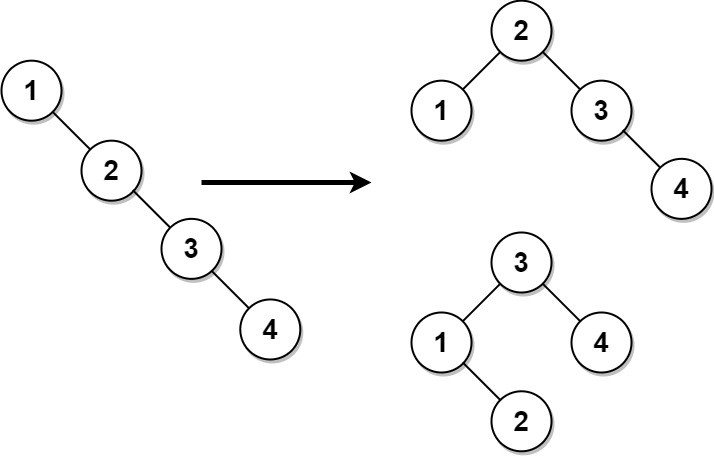


# In-Lab:

## Example :

1. Given the root of a binary search tree, return *a* ***balanced*** *binary search tree with the same node values*. If there is more than one answer, return **any of them**.

A binary search tree is **balanced** if the depth of the two subtrees of every node never differs by more than 1.



**Input:** root = [1,null,2,null,3,null,4,null,null]

**Output:** [2,1,3,null,null,null,4]

**Explanation:** This is not the only correct answer, [3,1,4,null,2] is also correct.

## Constraints:

* + The number of nodes in the tree is in the range [1, 104].
  + 1 <= Node.val <= 105

# Solution:

**package** INLAB\_5\_1; **import** java.util.\*; **public class** Main{

**public static void** main(String[] args) { Scanner s=**new** Scanner(System.***in***); **int** n=s.nextInt();

ArrayList<Integer>x=**new** ArrayList<>();

**for**(**int** i=0;i<n;i++)

{

x.add(s.nextInt())

}

ArrayList<Integer> x=**new** Solution().BSTconstructor(x, 0, n-1);

}

}

**class** TreeNode

{

TreeNode left; TreeNode right;

Integer val;

}

**class** Solution { ArrayList<Integer> list;

**public void** traverse(TreeNode root){ **if**(root == **null**) **return**; traverse(root.left); list.add(root.val); traverse(root.right);

}

**public** TreeNode balanceBST(TreeNode root) { list = **new** ArrayList<>(); traverse(root);

**return** BSTconstructor(list, 0 , list.size() - 1);

}

**public** TreeNode BSTconstructor(ArrayList<Integer> list, **int** low, **int** high){

**if**(low > high) **return null**;

**int** mid = low + (high - low)/2;

**int** val = list.get(mid);

TreeNode lc = BSTconstructor(list , low , mid - 1); TreeNode rc = BSTconstructor(list , mid + 1 , high);

**return new** TreeNode(val,low,high);

}

}

1. Write a program for AVL tree having functions for the following operations:
   * Insert an element (no duplicates are allowed),
   * Delete an existing element,
   * Traverse the AVL (in-order, pre-order, and post-order)

**INPUT:**

Line 1 contains an integer NQ, the number of queries.

The next NQ lines contain queries and are of the form 'i x*x*' (Insert x*x* into an AVL tree) or 'd x*x*' (Delete x*x* from an AVL tree).

## OUTPUT:

For each query, print value of an unbalanced node (if any) at which rotation is being applied.

The last three lines print 'Preorder traversal', 'Inorder traversal', and 'Postorder traversal' of an AVL tree that results after the execution of all NQ queries.

## SAMPLE INPUT:

8

i 1

i 2

i 3

i 4

i 5

i 6

d 4

d 5

SAMPLE OUTPUT:

1

3

2

6

2 1 6 3

1 2 3 6

1 3 6 2

# Solution:

**package** Lab5\_in2; **import** java.util.\*; **class** AVLNode

{

AVLNode left, right;

**int** data;

**int** height;

**public** AVLNode()

{

left = **null**; right = **null**; data = 0;

height = 0;

}

**public** AVLNode(**int** n)

{

left = **null**; right = **null**; data = n; height = 0;

}

}

**class** AVLTree

{

**private** AVLNode root;

**public** AVLTree()

{

root = **null**;

}

**private int** height(AVLNode avlNode)

{

**return** avlNode == **null** ? -1 : avlNode.height;

}

**private int** max(**int** lHeight, **int** rHeight)

{

**return** (lHeight > rHeight) ? lHeight : rHeight;

}

**public void** insert(**int** data)

{

root = insert(data, root);

}

**private** AVLNode insert(**int** data, AVLNode avlNode)

{

**if** (avlNode == **null**)

avlNode = **new** AVLNode(data);

**else if** (data < avlNode.data)

{

avlNode.left = insert(data, avlNode.left);

**if** (height(avlNode.left) - height(avlNode.right) == 2)

**if** (data < avlNode.left.data) avlNode = leftRotation(avlNode);

### else

avlNode = leftRightRotation(avlNode);

} **else if** (data > avlNode.data)

{

avlNode.right = insert(data, avlNode.right);

**if** (height(avlNode.right) - height(avlNode.left) == 2)

**if** (data > avlNode.right.data) avlNode = rightRotation(avlNode);

} **else**

;

### else

avlNode = rightLeftRotation(avlNode);

avlNode.height = max(height(avlNode.left), height(avlNode.right)) + 1;

**return** avlNode;

}

**private** AVLNode leftRotation(AVLNode avlNode)

{

AVLNode k1 = avlNode.left; avlNode.left = k1.right; k1.right = avlNode;

avlNode.height = max(height(avlNode.left), height(avlNode.right)) + 1; k1.height = max(height(k1.left), avlNode.height) + 1;

**return** k1;

}

**private** AVLNode rightRotation(AVLNode avlNode)

{

AVLNode node = avlNode.right; avlNode.right = node.left; node.left = avlNode;

avlNode.height = max(height(avlNode.left), height(avlNode.right)) + 1; node.height = max(height(node.right), avlNode.height) + 1;

**return** node;

}

**private** AVLNode leftRightRotation(AVLNode avlNode)

{

avlNode.left = rightRotation(avlNode.left);

**return** leftRotation(avlNode);

}

**private** AVLNode rightLeftRotation(AVLNode avlNode)

{

avlNode.right = leftRotation(avlNode.right);

**return** rightRotation(avlNode);

}

**public int** countNodes()

{

**return** countNodes(root);

}

**private int** countNodes(AVLNode avlNode)

{

**if** (avlNode == **null**) **return** 0;

### else

{

**int** l = 1;

l += countNodes(avlNode.left); l += countNodes(avlNode.right); **return** l;

}

}

**public boolean** search(**int** data)

{

**return** search(root, data);

}

**private boolean** search(AVLNode avlNode, **int** data)

{

**boolean** found = **false**;

**while** ((avlNode != **null**) && !found)

{

**int** rval = avlNode.data;

**if** (data < rval)

avlNode = avlNode.left;

**else if** (data > rval) avlNode = avlNode.right;

### else

{

}

found = **true**; **break**;

found = search(avlNode, data);

}

**return** found;

}

**public void** inorder() { inorder(root);

}

**private void** inorder(AVLNode avlNode)

{

**if** (avlNode != **null**)

{

inorder(avlNode.left); System.***out***.print(avlNode.data + " "); inorder(avlNode.right);

}

}

AVLNode minValueNode(AVLNode node)

{

AVLNode temp;

**for**(temp=node;temp.left!=**null**;temp=temp.left);

**return** temp;

}

AVLNode deleteNode(AVLNode root, **int** key)

{

**if** (root == **null**)

**return** root;

**if** (key < root.data)

root.left = deleteNode(root.left, key);

**else if** (key > root.data)

root.right = deleteNode(root.right, key);

### else

{

**if** ((root.left == **null**) || (root.right == **null**))

{

AVLNode temp = **null**; **if** (temp == root.left)

temp = root.right;

### else

temp = root.left;

**if** (temp == **null**)

{

}

### else

{

}

### else

temp = root; root = **null**;

root = temp;

AVLNode temp = minValueNode(root.right);

root.data=temp.data; root.right=deleteNode(root.right, temp.data);

}

}

**if** (root == **null**)

**return** root;

root.height = Math.*max*(height(root.left), height(root.right)) + 1;

**int** balance =height(root);

**if** (balance > 1 && height(root.left) >= 0)

**return** rightRotation(root);

**if** (balance > 1 && height(root.left) < 0)

{

root.left = leftRotation(root.left);

**return** rightRotation(root);

}

**if** (balance < -1 && height(root.right) <= 0)

**return** leftRotation(root);

**if** (balance < -1 && height(root.right) > 0)

{

root.right = rightRotation(root.right);

**return** leftRotation(root);

}

**return** root;

}

}

**public class** Main

{

**public static void** main(String[] args)

{

Scanner s=**new** Scanner(System.***in***); AVLTree a=**new** AVLTree();

**int** i=0;

**while**(i!=1)

{

System.***out***.println("1.insert\t2.delete\t3.Inorder\nEnter Your Choice\n");

**int** n=s.nextInt();

**switch**(n)

{

**case** 1:

{

**int** k=s.nextInt();

**if**(a.search(k))

{

}

### else

{

}

### break;

}

System.***out***.println("AlreadyThere");

a.insert(k);

**case** 2:

{

a.deleteNode(**null**, s.nextInt());

### break;

}

**case** 3:

{

a.inorder();

}

}

}

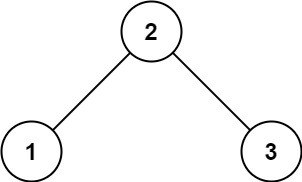
}

}

# Post-Lab:

1. Given the root of a binary tree, write a Java program to return the leftmost value in the last row of the tree.

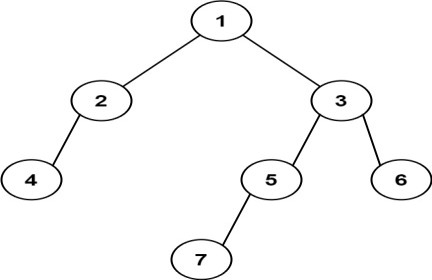
**Example 1:**



**Input:** root = [2,1,3]

## Output: 1

**Example 2:**



**Input:** root = [1,2,3,4,null,5,6,null,null,7]

**Output:** 7

# Solution:

**package** BST;

**import** java.util.Scanner;

**class** node

{

node(**int** data)

{

**this**.data=data; left=right=**null**;

}

**int** data; node left; node right;

}

**public class** Xys {

**public static void** main(String args[])

{

node root=**null**;

root=*insert*(root,2); root=*insert*(root,1); root=*insert*(root,3); root=*insert*(root,4);

System.***out***.println(*max*(root));

}

**public static** node insert(node root,**int** data)

{

**if**(root==**null**)

{

**return new** node(data);

}

**else if**(root.data>data)

{

root.left=*insert*(root.left,data);

}

**else if**(root.data<=data)

{

root.right=*insert*(root.right,data);

}

**return** root;

}

**public static int** max(node x)

{

**if**(x.right==**null**)

{

**return** x.data;

}

**return** *max*(x.right);

}

}

1. List out real-world applications of AVL trees.

# Solution:

A.

* AVL trees are mostly used for in-memory sorts of sets and dictionaries.
* AVL trees are also used extensively in database applications in which insertions and deletions are fewer but there are frequent lookups for data required
* It is used in applications that require improved searching apart from the database applications.
* AVL tree is a balanced binary search tree which employees rotation to maintain balance.
* It has application in story-line games as well.It is use mainly used in corporate sectors where the have to keep the information about the employees working there and their change in shifts.

**D. BALAJI REDDY**