Exercise 7 AVL TREE

7. Write a C++ Program for implementation of AVL tree performing following operations
a) Insertion b) Deletion

Objective: the objective of this exercise is to enable you to program AVL tree and to perform operations on it.

Procedure and description:
An AVL tree is a self-balancing binary search tree, Insertions and deletions may require the tree to be rebalanced by one or more tree rotations. In other words AVL tree is a binary search tree where the height of the left subtree differs from the height of the right subtree by at most 1 level, if it exceeds 1 level then rebalancing occurs.

a) Insertion:
Inserting an element into an AVL search tree is very similar to that of the process used in a binary tree. But after insertion of the element, if the balance factor of any node in the tree is affected, we need to resolve this problem by applying the technique called rotation.

To perform rotations it is necessary to identify a specific node A whose balance factor(A) is neither 0, 1 or -1, and which is the nearest ancestor to the inserted node on the path from the inserted node to A will have their balance factors to be either 0, 1, or -1. The rebalancing rotations are classified as LL (Left-Left), LR (Left-Right), RR (Right-Right), and RL (Right-Left).

Algorithm: Let P be the root of the unbalanced subtree, with R and L denoting the right and left children of P respectively.

Right-Right case and Right-Left case:
Step 1: if the balance factor of P is -2 then the right subtree outweighs the left subtree of the given node, and the balance factor of the right child (R) must be checked. The left rotation with P as the root is necessary.

Step 2: if the balance factor of R is -1, a single left rotation (with P as the root) is needed (Right-Right case).

Step 3: if the balance factor of R is +1, two different rotations are needed. The first rotation is a right rotation with R as the root. The second is a left rotation with P as the root (Right-Left case).

Left-Left case and Left-Right case:

Step 1: if the balance factor of P is 2, then the left subtree outweighs the right subtree of the given node, and the balance factor of the left child (L) must be checked. The right rotation with P as the root is necessary.

Step 2: if the balance factor of L is +1, a single right rotation (with P as the root) is needed (Left-Left case).

Step 3: if the balance factor of L is -1, two different rotations are needed. The first rotation is a left rotation with L as the root. The second is a right rotation with P as the root (Left-Right case).

Expected Output: After execution of program enter choice as insertion operation.

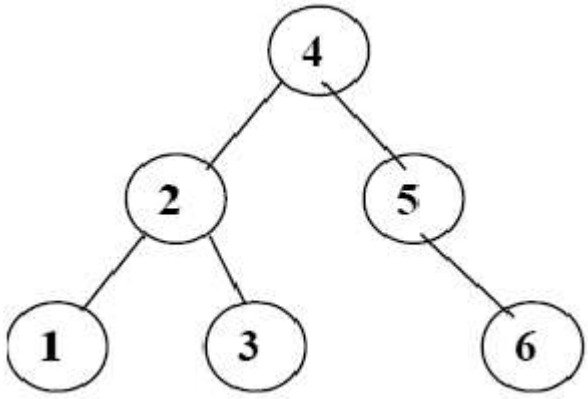

Fig: Initial AVL Tree before Insertion

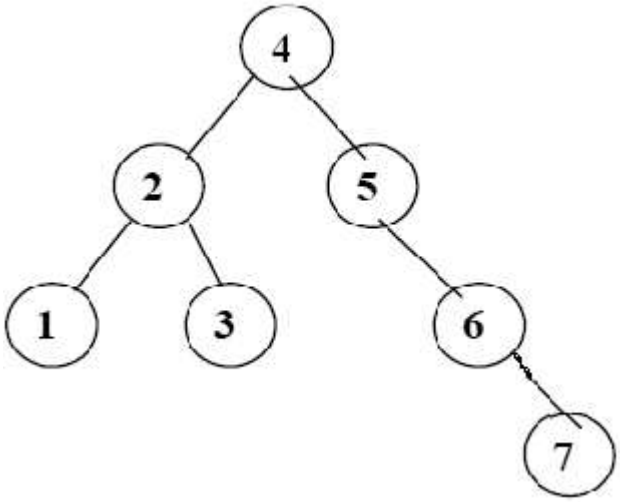

Fig: After inserting Node 7(It is non avl tree so perform rotation)

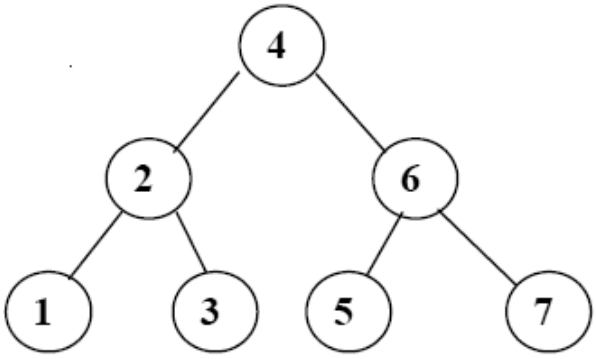

Fig: AVL Tree after performing single rotation

b) Deletion:

Algorithm:

Step 1: if the node is a leaf or has only one child, remove it.

Step 2: Otherwise, replace it with either the largest in its left subtree (in order predecessor) or the smallest in its right subtree (in order successor), and remove that node.

Step 3: The node that was found as a replacement has at most one sub tree. After deletion, retrace the path back up the tree (parent of the replacement) to the root, adjusting the balance factors as needed.

Expected Output: After execution of program enter choice as deletion operation.

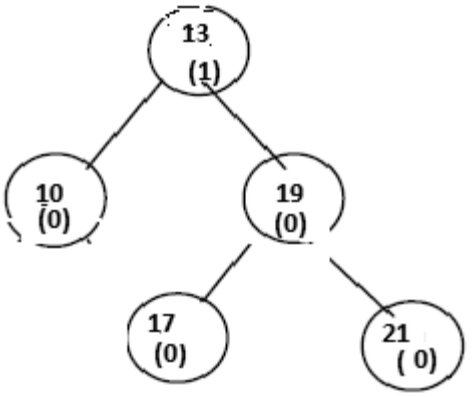

Fig: Initial AVL Tree before deletion

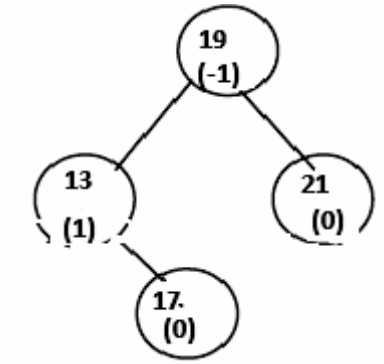

Fig: AVL Tree after deleting node j