Program-4

Write a program to perform insertion and deletion operations on AVL trees.

→ Insertion

```
Node *rightRotate(Node *node)
{
    Node *node1 = node->left;
    Node *node2 = node1->right;

    node1->right = node
    node->left = node2

    node->height = max(height(node->left),
                       height(node->right))+1
    node1->height = max(height(node1->left),
                        height(node1->right))+1
    return node1
}
Node *leftRotate(Node *node)
{
    Node *node1 = node->left right
    Node *node2 = node1->left

    node1->left = node
    node->right = node2

    node->height = max(height(node->left),
                       height(node->right))+1
    node1->height = max(height(node1->left),
                        height(node->right))+1
    return node1
}
```

node
node1
node2

SURYA Gold Date Page

Scanned with CamScanner

```
Node* insert(Node* node, int key)
{
    if(node == NULL)
        return(newNode(key));

    if(key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node;

    node->height = 1 + max(height(node->left),
                           height(node->right));

    int balance = getBalance(node);

    if(balance > 1 && key < node->left->key)
        return rightRotate(node);

    if(balance < -1 && key > node->right->key)
        return leftRotate(node);

    if(balance > 1 && key > node->left->key)
    {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if(balance < -1 && key < node->right->key)
    {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}
```

```
Node* delete (Node** node, int key)
{
    if (root == NULL)
        return root

    if(key < node->key)
        node->left = delete(node->left, key)

    else if(key > node->key)
        node->right = delete(node->right, key)

    else
    {
        if( (root->left == NULL) || (root->right == NULL)
        {
            Node *temp = root->left ? root->left :
                         root->right;

            if(temp == NULL)
            {
                temp = node;
                node = NULL;
            }
            else
            {*node = *temp
            free(temp);
        } }

        else
        {
            Node* temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = delete(root->right, temp->key);
        }
    }
```

```
node
if(root == NULL)
  return root;

root->ht
node->height = 1 + max(height(node->left),
                       height(node->right))

int balance = getBalance(root)
                         node

if(balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(node)

if(balance > 1 && getBalance(root->left) < 0)
  {  node                    node
     root->left = leftRotate(root->left);
     return rightRotate(node);
  }

if(balance < -1 && getBalance(node->right) <= 0)
    return leftRotate(node)

if(balance < -1 && getBalance(root->right) > 0)
  {  node
     root->right = rightRotate(node->right);
     return leftRotate(node);
  }
return root
}
```