AVL Insertion and Deletion

1BM18CS002
Abhijnya K.G

Insert

```
Node insert(node, key)
{
    if(node == NULL)
        return create_node(key)
    if(key < node->data)
        node->left = insert(node->left, key)
    if(key > node->data)
        node->right = insert(node->right, key)

    node->height = 1 + max(height(node->right), height(node->left));
// calculate balance after each insertion
    balance = balance(node)
    if(balance > 1 && key < node->left->data)
        return rightrotation(node);          // left rotation

    if(balance > 1 && key > node->left->data)
    {
        node->left = leftrotate(node->left)
        return rightrotate(node)             // Left right rotation
    }
```

```
// Right rotation
if (balance < -1 && key > node->right->data)
    return leftRotate(node);
// Right - left
if (balance < -1 && key < node->right->data)
{
    node->right = rightRotate(node->right);
    return leftRotate(node);
}
```

```
RightRotation(node x2)
{
    Node x1 = y->left
    Node x3 = x1->right
    x1->right = x2
    x2->left = x3
    update height
}
```

```
leftRotation(node x1)
{
    Node x2 = x1->right
    Node x3 = x2->left
    x2->left = x1
    x1->right = x3
    // update height
}
```

Deletion

```
Delete (root, key)
{
    if (root == NULL) return root
    if (key < root->data)
        root->left = delete(root->left, key)
    else if (key > root->data)
        root->right = delete(root->right, key)
    else
    {
        if (node with only 1 child or no child)
            temp = root->left ? root->left : root->right
            free (temp)
    }
    else
    {
        if (node with 2 children) root->right = delet (right, key)
            // Find inorder successor and delete it
    }
    // update height of current node
    balance = get_balance(root)
```

4 calls: Left rotate

```
    return right(root)
```

Right rotate

```
    return left(root)
```

left right

```
    root->left = left(root->left)
    right(root)
```

Right left

```
    root->right = right(root->right)
    left(root)
```