14/10/20 ADS Dhanraj, K
IBM18CS027

```
// AVL tree
class Node{
    Public:
        int Key;
        Node *left;
        Node *right;
        int height;
};

// height
int height(Node *N){
    if (N==NULL) return 0;
    return N-> height;
}

//Rotation
Node *rightRotate(Node * y){                    // right rotation
    Node *x = y->left;
    Node * T2 = x->right;
    x-> right = y;
    y -> left = T2;
    y -> height = max(height(y->left) , height(y->right))+ 1 ;
    x-> height = max(height(x->left),height(x->right)) +1 ;
    return x;
}
Node *leftRotate (Node *x) {                    // left rotation
    Node *y = x->right;
    Node *T2 = y-> left ;
    y-> left = x;
    x-> right = T2 ;
    x-> height = max( height (x->left), height(x->right)) +1 ;
    y -> height = max(height (y-> left) ,height( y-> right)) +1;
    return y;
}
```

```
// balance factor => height(N->left) - height(N->right);

// To insert a node

Node * insertNode ( Node *node , int key) {
        if(node == NULL)
            return( newNode(key));
        if (key < node-> key)
            node-> left = insert Node(node-> left , key);
        else if (key > node ->key)
            node -> right = insertNode (node -> right , key);
        else
            return node;

        // balance the tree
        node -> height = 1+ max(height (node-> left), heigh(node-> right));
        int bf = getBalanceFactor(node);
        if ( ~~balance~~ bf >1) {
                if (key< node -> left -> key) {
                        return rightRotate (node );
                } else if ( key > node -> left -> key) {
                node -> left = leftRotate (node-> left);
                return rightRotate(node);
                }
        }
        if (balanceFactor <-1) {
                if (key > node-> right -> key ) {
                        return left Rotate(node);
        } else if (key < node -> right ->key)  {
            node -> right = right Rotate (node -> right);
            return leftRotate (node );
            }
        }
        return node;
}
```

1BM18CS027
Dharmaraj, K.

ADS

AVL tree deletion

```
Node * deleteNode(Node *root , int Key ) {

        //delete
        if ( root == NULL)
            return root;
        if (Key < root -> Key )
            root -> left = deleteNode(root -> left ,Key);
        else if (Key > root->Key)
            root -> right = deleteNode(root -> right, Key);
        else {
            if(( root -> left == NULL) || (root -> right == NULL)){
                    Node *temp = root -> left ? root -> left : root -> right
                    if (temp == NULL) {
                            temp = root;
                            root = NULL;
                    }
                    else
                        *root = *temp;
                    free(temp);
            } else {
                Node *temp = nodeWithMinimumValue(root -> right);
                root -> Key = temp -> Key;
                root -> right = deleteNode(root -> right, temp -> Key);
            }
        }
        if (root == NULL) return root;
        // balancing
        root -> height = 1 + max(height (root -> left), height(root -> right));
        int bf = getBalanceFactor(root);
        if (bf > 1){

         // this part of balancing the tree is same as balancing
            after inserting a node.

        }
        if (bf < -1) {
                // Same as insertion
        }
        return root;
}
```

Scanned with CamScanner