papergrid

Date: / /

Khrithik S Anand
1BM18CS046

```
int height (Node *N)
{                                   //for height of AVL
                                                   tree
    if (N==NULL)
          return 0;
    return N->height;
}

int max (int a, int b)
{
      return (a>b) ? a : b;            //max of tow
                                                 no.
}
```

=> To perform Right rotate.

```
Node *rightRotate (Node *y)
{
      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
}
```

Khrithik S. Anand
1BM18CS046

papergrid
Date: / /

⇒ To rotate left rotate

```
Node *leftRotate (Node *x)
{
        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;
}
```

⇒ to get Balance

```
int getBalance (Node *N)
{
      if(N==NULL)
            return 0;
      return heigh (N->left) - heigh (N->right);
}
```

⇒ to insert a key (insertion)

Khrithik S. Anand
1BM18CS046

papergrid
Date: / /

```
Node * insert (Node * node ,int key)
{
    if (node ==NULL)
        return (new Node (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 left Rotate (node);
    if (balance >1 && key >node -> left -> key)
    {
        node -> left = leftRotate (node -> left);
        return ~~return Rota~~ rightRotate (node);

    if (balance < -1 && key <node -> right key)
    {
        node -> right = right Rotate (node -> right)
        return left Rotate (node);
    }
    return node;
```

Khrithik .S. Anand
1BM18CS046

papergrid
Date: / /

⇒ for deletion

```
Node *~~a~~ deleteNode(Node* root, int key)
{
    if (root == NULL)
        return root;
    if (key < root -> key)
        ~~ro~~ 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 = minValueNode(root -> right)
            root -> key = temp -> key
            root -> right = deleteNode (root -> right,
                                        temp -> key);
        }
    }
```

Khrithik S. Anand

1BM18CS046

papergrid

Date: / /

```
if (root = = NULL)
    return root;
    root -> height = 1+ max( height (root -> left),
                             height (root -> right ));
int balance = get balance (root);

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

if ( balance > 1 && getBalance (root -> left)< 0)
{
        root -> left = left Rotate (root -> left);
        return rightRotate (root);
}
if (balance <-1 &&
        getBalance ( root -> right) < =0)
        return left Rotate (Root );
if (balance <-1 &&
        getBalance ( root -> rigth) >0)
{
        root -> right = right Rotate (root -> right);
}
    return root;
}
```