ADS : Lab 4. Akanksha Laddha

USN : 1BM18CS007 Batch : B1.

Date:
P. No:

Ques : Implement insertion and deletion in AVL trees.

```
// structure node.
typedef struct node
{
    int data;
    struct node *left;
    struct node * right;
};
typedef node* NODE;

// class implementation

class Tree
{
    int getHeight (NODE);
    int getBalanceFactor (NODE);
    NODE leftRotate (NODE);
    NODE rightRotate (NODE);
    NODE rotation (NODE);
    void display (NODE);
    NODE insertElement (NODE, int);
    NODE deleteElement (NODE, int);
    NODE minValueNode (NODE);
};

// NODE Tree :: leftRotate (NODE p)
{
    // store right pointer of p in y.
```

Anamika Haddu

1BM18CS007

if y has store left pointer of y.

update left node of y as p.
&
update right of p as left of y.

return new root i.e y.

}

// NODE Tree :: rightRotate (NODE p)
{

y = left node of p.
T_2 = right node of y.

y update right node of y as p.
update left node of p as T_2.

return new root i.e y.

}

getBalanceFactor (NODE temp)
{

a recursive call
store height of left subtree in lHeight
store height of right subtree in rHeight

return diff of lHeight & rHeight.

}

Akanksha
Naddha

Date:
P. No:

1BM18CS007

```
// NODE Tree :: rotation (NODE p)
{
    get balance factor of node p

    if (balancefa > 1)
    {
        // if balance factor of p → left > 0
            rightRotate (p);   RRotation

        // else   LR.
        p → left = leftRotate (p → left)
        p = rightRotate (p);
    }
    else if (balancef. < -1)
    {
        // if balance factor of p → right > 0
            leftRotate (p);   left rotation

        // else   RL
        p → right = rightRotate (p → right);
        p = leftRotate (p)

        // else   left rotation.
            leftRotate (p)
    }
    return p

}
```

Akanksha
Laddha

1BM18CS007

```
// NODE Tree :: insertElement (NODE root, int value)
{
    if (root == NULL)
    {
        // create a new node
        & assign values

        return root;
    }
    else if ( value < root → data)
    {   // inserts in Left ST.
        recur. for root → left
        & ~~call~~ root = rotation (root);
    }
    else if (value >= root → data)
    {
        // inserts in Right ST
        recur for root → right
        & root = rotation (root)
    }
    return root
}

// NODE ~~det~~ Tree :: deleteElement (NODE p, int value)
{
    if (p == NULL)
        return p;
```

Akanksha
vaddva

IBM18CS007

// if value is less than p → data
recur for p → left &
p = rotation (p);

// else if value is greater than p → data
recur for p → right &
p = rotation (p);

// else
{ root node case.

// node where one child is present
NODE temp = root → left ? root → left
: root → right ;

```
if (temp == NULL)
{
    temp = p ;
    p = NULL ;
}
else
{
    p = temp ;
    p = rotation (p) ;
}
free temp
}
```

// NODE temp = minValueNODE (p → right);
make temp → data as p → data

Akanksha
Reddy

IBM18CS007

```
        p -> right = deleteElement (p -> right,
                                    temp -> data);
        p = rotation (p);
    }

    return p;

}

// NODE Tree :: minValueNode (NODE p)
{
    // finds the leftmost leaf

}
```