classmate
Date
Page

PROGRAM-04

Arpana M Ramaswamy
1BM18CS147
14/10/20 Arpana

Q) WAP to perform insertion & deletion operation on AVL Trees

```
//Node declaration
class Node
{ public:
  int key;
  Node* left, right;
  int height;
}

// function to assign new node
~~Node* NewNode~~
 Node* NewNode(int key)
 {
   Node* node = new Node();
   node-> key = key;
   node-> left = NULL;
   node-> right = NULL;
   ~~height~~
   node-> height=1;  // initially when node is not added to tree
   return(node);
 }
```

classmate

Date
Page

Arpana M R
1BM18CS147
Arpana

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

//fun^c to calculate Balance Factor
int Balance(Node *N)
{
 if (N==NULL)
   return 0;
 return height(N->left) - height(N->right);
}

// Right Rotation fun^c
Node* RightRotation (Node *y)
{
 Node *x = y->left;
 Node *T = x->right;
 x->right=y;
 y->left=T;
 y->height=max(height(y->left), height(y->right))
           +1;
x->height=max(height(x->left), height(x->right))+1;
```

classmate

Date ______

Page ______

Arpana MR
1BM18CS147
Arpana

```
    return x;
}

// left Rotation func
Node* LeftRotate(Node *x)
{
    Node *y = x->right;
    Node *z = y->left;
    y->left = x;
    x->right = z;
    x->height = max(height(x->left), height(x->right))+1;
    y->height = max(height(y->left), height(y->right))+1;
    return y;
}

// insertion Func
Node* insert(Node* node, int key)
{
    //Normal BST insertion
    if(node == NULL)
        return(NewNode(key));
    if( key < node->key)
        node->left = insert(node->left, key);
    else if( key > node->key)
```

Arpana M R
1BM18CS147
Arpana

```
        node->right = insert(node->right, key);
    else
        return node;  // if value already present, skip it

    // Update height of ancestor nodes
    node->height = 1 + max(height(node->left),
                           height(node->right));

    int BF = Balance(node);  // get balancing Factor

    if (BF > 1 && key < node->left->key)
        return RightRotation(node);

    if (BF < -1 && Key > node->right->key)
        return LeftRotation(node);

    if (BF > 1 && Key > node->left->key)
    {
        node->left = LeftRotation(node->left);
        return RightRotation(node);
    }

    if (BF < -1 && Key < node->right->key)
    {   node->right = RightRotation(node->right);
```

Anpana M R
1BM18CS147
Anpana

```
        return LeftRotation(node);
    }
    return node;
}

//Finding minvalue of a tree & returning the node

Node* minNode(Node* node)
{
    Node * curr = node;

    // find minimum node
    while (curr->left != NULL)
        curr = curr->left;
    return curr;
}

// Deletion Func
Node* delete(Node * root, int key)
{   // Normal BST deletion
    if(root == NULL)
        return root;

    if (key < root->key)
        root-> left = delete(root-> left, key);
```

Arpana M R
1BM18CS147
Arpana

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

else
{
    if((node->left==NULL) ||(node->right==NULL))
{
    Node *temp = node->left ? node->left : node->right;
    //no child
    if(temp==NULL)
    {
        temp=node;
        node=NULL;
    }
    else        //one child
    *node=*temp;
    free(temp);
}

else    //two children
{
    Node *temp =minNode(node->right);
    node->key= temp->key;

    node->right = delete(node->right, temp->key)
```

classmate
Date
Page

Arpana M R
1BM18CS147
Arpana

```
    }
}

if (node==NULL)
return node;

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

int balance = Balance(node);

if( balance >1 && Balance(node->left) >=0)
    return RightRotation(node);
}

if( balance >1 && Balance(node->left)<0 )
{
    node->left = LeftRotation(node->left);
    return RightRotation(node);
}

if (balance <-1 && Balance(node->right)<=0)
    return LeftRotation(node);
if (balance <-1 && Balance(node->right)>0)
{ node->right = RightRotation(node->right);
  return LeftRotation(node)
}
```