**VARUN KUMAR**

**2K19-IT-140**

**DS LAB**

19: Write a Algorithm to count the number of leaf nodes in a a AVL tree.

**Algorithm :-**

1) If the node is null return 0, this is also the base case of our recursive algorithm  
2) If a leaf node is encountered then return 1  
3) Repeat the process with left and right subtree  
4) Return the sum of leaf nodes from both left and right subtree

#include <bits/stdc++.h>

using namespace std;

class TreeNode

{ public:

int val;

class TreeNode\* left;

class TreeNode\* right;

};

int Count\_leaves(class TreeNode\* TreeNode)

{

if(TreeNode == NULL)

return 0;

if(TreeNode->left == NULL && TreeNode->right == NULL)

return 1;

else

return Count\_leaves(TreeNode->left)+

Count\_leaves(TreeNode->right);

}

class TreeNode\* newNode(int val)

{

class TreeNode\* TreeNode = (class TreeNode\*)

malloc(sizeof(class TreeNode));

TreeNode->val = val;

TreeNode->left = NULL;

TreeNode->right = NULL;

return(TreeNode);

}

int main()

{

class TreeNode \*root = newNode(5);

root->left = newNode(3);

root->right = newNode(6);

root->left->left = newNode(1);

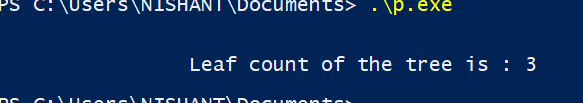
root->left->right = newNode(2);

cout << "\n\n\t\tLeaf count of the tree is : "<<

Count\_leaves(root) << endl<<endl;

return 0;

}



# 20: Write a program to Delete a key from the AVL tree.

#include<bits/stdc++.h>

using namespace std;

class TreeNode

{

public:

int val;

TreeNode \*left;

TreeNode \*right;

int height;

};

int height(TreeNode \*cur)

{

if (cur == NULL)

return 0;

return cur->height;

}

TreeNode\* New\_Node(int val)

{

TreeNode\* cur = new TreeNode();

cur->val = val;

cur->left = NULL;

cur->right = NULL;

cur->height = 1;

return(cur);

}

TreeNode \*Rotate\_Right(TreeNode \*y)

{

TreeNode \*x = y->left;

TreeNode \*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;

}

TreeNode \*Rotate\_left(TreeNode \*x)

{

TreeNode \*y = x->right;

TreeNode \*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;

}

int Balance\_get(TreeNode \*cur)

{

if (cur == NULL)

return 0;

return height(cur->left) -

height(cur->right);

}

TreeNode\* Insertion(TreeNode\* TreeNode, int val)

{

if (TreeNode == NULL)

return(New\_Node(val));

if (val < TreeNode->val)

TreeNode->left = Insertion(TreeNode->left, val);

else if (val > TreeNode->val)

TreeNode->right = Insertion(TreeNode->right, val);

else

return TreeNode;

TreeNode->height = 1 + max(height(TreeNode->left),

height(TreeNode->right));

int balance = Balance\_get(TreeNode);

if (balance > 1 && val < TreeNode->left->val)

return Rotate\_Right(TreeNode);

if (balance < -1 && val > TreeNode->right->val)

return Rotate\_left(TreeNode);

if (balance > 1 && val > TreeNode->left->val)

{

TreeNode->left = Rotate\_left(TreeNode->left);

return Rotate\_Right(TreeNode);

}

if (balance < -1 && val < TreeNode->right->val)

{

TreeNode->right = Rotate\_Right(TreeNode->right);

return Rotate\_left(TreeNode);

}

return TreeNode;

}

TreeNode \* Min\_Val\_Node(TreeNode\* c)

{

TreeNode\* cur = c;

while (cur->left != NULL)

cur = cur->left;

return cur;

}

TreeNode\* Delete\_Node(TreeNode\* root, int val)

{

if (root == NULL)

return root;

if ( val < root->val )

root->left = Delete\_Node(root->left, val);

else if( val > root->val )

root->right = Delete\_Node(root->right, val);

else

{

if( (root->left == NULL) ||

(root->right == NULL) )

{

TreeNode \*temp = root->left ?

root->left :

root->right;

if (temp == NULL)

{

temp = root;

root = NULL;

}

else

\*root = \*temp;

free(temp);

}

else

{

TreeNode\* temp = Min\_Val\_Node(root->right);

root->val = temp->val;

root->right = Delete\_Node(root->right,

temp->val);

}

}

if (root == NULL)

return root;

root->height = 1 + max(height(root->left),

height(root->right));

int balance = Balance\_get(root);

if (balance > 1 &&

Balance\_get(root->left) >= 0)

return Rotate\_Right(root);

if (balance > 1 &&

Balance\_get(root->left) < 0)

{

root->left = Rotate\_left(root->left);

return Rotate\_Right(root);

}

if (balance < -1 &&

Balance\_get(root->right) <= 0)

return Rotate\_left(root);

if (balance < -1 &&

Balance\_get(root->right) > 0)

{

root->right = Rotate\_Right(root->right);

return Rotate\_left(root);

}

return root;

}

void preOrder(TreeNode \*root)

{

if(root != NULL)

{

cout << root->val << " ";

preOrder(root->left);

preOrder(root->right);

}

}

int main()

{

TreeNode \*root = NULL;

root = Insertion(root, 9);

root = Insertion(root, 5);

root = Insertion(root, 10);

root = Insertion(root, 0);

root = Insertion(root, 6);

root = Insertion(root, 11);

root = Insertion(root, -1);

root = Insertion(root, 1);

root = Insertion(root, 2);

/\* The constructed AVL Tree would be

9

/ \

1 10

/ \ \

0 5 11

/ / \

-1 2 6

\*/

cout << "\n\n\tPreorder traversal of the "

"constructed AVL tree is:-- \n\n\t";

preOrder(root);

root = Delete\_Node(root, 10);

/\* The AVL Tree after deletion of 10

1

/ \

0 9

/ / \

-1 5 11

/ \

2 6

\*/

cout << "\n\n\tPreorder traversal after"

<< " deletion of 10:--- \n\n\t";

preOrder(root);

return 0;

}

