

**DATA STRUCTURES AND ALGORITHMS – LAB**

**FINAL PROJECT**

**AVL IMPLEMENTATION**

**GROUP MEMBERS:**

* HASSAN JAMSHAID (L1F22BSCS0422)
* ABDULLAH SADIQ (L1F22BSCS0425)
* TALHA BADAR (L1F22BSCS0406)
* SALMAN HAMEED (L1F22BSCS0444)

**TreeNode.h:**

#pragma once

#include<iostream>

using namespace std;

struct Treenode

{

public:

int data;

Treenode\* left;

Treenode\* right;

int Height;

Treenode(int d)

{

this->data = d;

this->left = NULL;

this->right = NULL;

this->Height = 1;

}

};

**AVL.h:**

#pragma once

#include<iostream>

#include"TreeNode.h"

using namespace std;

class AVLBST

{

public:

Treenode\* root;

AVLBST()

{

root = NULL;

}

//<------------------------------------------------Height of tree------------------------------------------------------>//

int height(Treenode\* n)

{

if (n == NULL)

{

return 0;

}

return n->Height;

}

//<--------------------------------------------MAX Function--------------------------------------------------->//

int max(int x, int y)

{

if (x > y)

{

return x;

}

else

{

return y;

}

}

//<--------------------------------------------Balance Factor------------------------------------------------>//

int getbalance(Treenode\* n)

{

if (n == NULL)

{

return 0;

}

return height(n->left) - height(n->right);

}

// <-----------------------------------------------INSERTION------------------------------------------------->//

Treenode\* insert(Treenode\* n, int d)

{

if (n == nullptr)

{

return new Treenode(d);

}

if (d < n->data)

{

n->left = insert(n->left, d);

}

else if (d > n->data)

{

n->right = insert(n->right, d);

}

else

{

return n;

}

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

int b = getbalance(n); //Balance cases -1,0,1 // Balance factor calculated in getbalance function

if (b > 1 && d < n->left->data)

{

return RR(n);

}

if (b < -1 && d > n->right->data)

{

return RL(n);

}

//<-------------------------------------------------------Left Right Rotation Case---------------------------------------->//

if (b > 1 && d > n->left->data)

{

return LRR(n);

}

//<-------------------------------------------------------Right Left Rotation Case------------------------------------------>//

if (b < -1 && d < n->right->data)

{

return RLR(n);

}

return n;

}

//<-----------------------------------------------------------DELETION------------------------------------------------------->//

Treenode\* deletetion(Treenode\* root, int k)

{

if (!root)

{

return root;

}

if (k < root->data)

{

root->left = deletetion(root->left, k);

}

else if (k > root->data) {

root->right = deletetion(root->right, k);

}

else

{

if ((!root->left) || (!root->right))

{

Treenode\* temp;

if (root->left)

{

temp = root->left;

}

else

{

temp = root->right;

}

if (!temp)

{

temp = root;

root = nullptr;

}

else

{

\*root = \*temp;

}

delete temp;

}

else

{

Treenode\* temp = MinNode(root->right);

root->data = temp->data;

root->right = deletetion(root->right, temp->data);

}

}

if (!root)

return root;

root->Height = max(height(root->left), height(root->right)) + 1; //<----------Height calculation again after deletion--------->//

int b = getbalance(root); //<-----------------Balance factor again -1,0,1------------------>//

if (b > 1 && getbalance(root->left) >= 0)

{

return RR(root);

}

else if (b > 1 && getbalance(root->left) < 0)

{

return LRR(root);

}

else if (b < -1 && getbalance(root->right) <= 0)

{

return RL(root);

}

else if (b < -1 && getbalance(root->right) > 0)

{

return RLR(root);

}

return root;

}

// <-------------------------------- TRAVERSALS -------------------------------------------> //

void inorder(Treenode\* root)

{

if (root != NULL)

{

inorder(root->left);

cout << root->data << " ";

inorder(root->right);

}

}

void Preorder(Treenode\* n)

{

if (n != NULL)

{

cout << n->data << " ";

Preorder(n->left);

Preorder(n->right);

}

}

void Postorder(Treenode\* n)

{

if (n != NULL)

{

Postorder(n->left);

Postorder(n->right);

cout << n->data << " ";

}

}

//<------------------------------------------------Rotations------------------------------------------------>//

Treenode\* RR(Treenode\* n)

{

Treenode\* x = n->left;

Treenode\* y = x->right;

x->right = n;

n->left = y;

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

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

return x;

}

Treenode\* RL(Treenode\* n)

{

Treenode\* x = n->right;

Treenode\* y = x->left;

x->left = n;

n->right = y;

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

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

return x;

}

Treenode\* LRR(Treenode\* n)

{

n->left = RL(n->left);

return RR(n);

}

Treenode\* RLR(Treenode\* n)

{

n->right = RR(n->right);

return RL(n);

}

//<-----------------------------------------------SEARCH------------------------------------------------->//

bool search(Treenode\* n, int k)

{

if (n == NULL)

{

return false;

}

if (n->data == k)

{

return true;

}

else if (k < n->data)

{

return search(n->left, k);

}

else

{

return search(n->right, k);

}

}

//<---------------------------------------------MAximum and Minimum------------------------------------------->//

Treenode\* MaxNode(Treenode\* n)

{

Treenode\* temp = n;

while (temp->right != NULL)

{

temp = temp->right;

}

return temp;

}

int Maximum()

{

if (root == NULL)

{

cout << "Tree is empty" << endl;

return -1;

}

Treenode\* temp = MaxNode(root);

return temp->data;

}

Treenode\* MinNode(Treenode\* n)

{

Treenode\* temp = n;

while (temp->left != NULL)

{

temp = temp->left;

}

return temp;

}

int Minimum()

{

if (root == NULL)

{

cout << "Tree is empty" << endl;

return -1;

}

Treenode\* temp = MinNode(root);

return temp->data;

}

//<-----------------------------------------------Diameter----------------------------------------------->//

int dm(Treenode\* n)

{

if (n == NULL)

{

return 0;

}

int lHeight = height(n->left); //height of left subtree

int rHeight = height(n->right); // height of right subtree

int ldiameter = dm(n->left);

int rdiameter = dm(n->right);

return max(lHeight + rHeight + 1, max(ldiameter , rdiameter));

}

int diamter()

{

return dm(root);

}

//<---------------------------------------Sucessor and Predecessor------------------------------------------>//

void pred(Treenode\* n, int k, Treenode\*& predecessor)

{

if (n == nullptr)

{

return;

}

if (n->data== k)

{

if (n->left != nullptr)

{

predecessor = MaxNode(n->left);

}

}

else if (n->data > k)

{

pred(n->left, k, predecessor);

}

else

{

predecessor = n;

pred(n->right, k, predecessor);

}

}

void suc(Treenode\* n, int k, Treenode\*& successor)

{

if (n == nullptr)

{

return;

}

if (n->data == k)

{

if (n->right != nullptr)

{

successor = MinNode(n->right);

}

}

else if (n->data < k)

{

suc(n->right, k, successor);

}

else {

successor = n;

suc(n->left, k, successor);

}

}

int predecessor(int k)

{

Treenode\* temp = NULL;

pred(root, k, temp);

if (temp != nullptr)

{

return temp->data;

}

else

{

return -1;

}

}

int successor(int k)

{

Treenode\* temp = NULL;

suc(root, k, temp);

if (temp != nullptr)

{

return temp->data;

}

else

{

return -1;

}

}

};

**Source.cpp:**

#include<iostream>

#include"TreeNode.h"

#include"AVL.h"

using namespace std;

int main()

{

//Treenode\* root = NULL;

AVLBST h;

int in = 0;

cout << "\t\t\t\t<-----Welcome to AVL Tree Implemetation----->" << endl;

while (true)

{

cout << endl;

cout << "Please select from Following options:" << endl;

cout << endl;

cout << "1) Insertion " << endl;

cout << "2) Deletion " << endl;

cout << "3) Traversals " << endl;

cout << "4) Search " << endl;

cout << "5) Minimum And Maximum " << endl;

cout << "6) Height of Tree " << endl;

cout << "7) Balance Factor Calculation " << endl;

cout << "8) Predecessor and Sucessor " << endl;

cout << "9) Diameter Calculation " << endl;

cout << "10) Exit " << endl;

cout << endl << endl;

int x, a;

cout << "Enter your Choice (1-10) :" << endl;

cin >> x;

switch (x)

{

case 1://<---------------INSERTION-------------------------->//

int data;

cout << "Enter Data in Tree or press -1 to Terminate insertion:" << endl;

cin >> data;

while (data != -1)

{

h.root = h.insert(h.root, data);

input: cin >> data;

if (h.search(h.root, data))

{

cout << "\nNumber already present in the tree \n";

goto input;

}

}

break;

case 2://<-------------------DELETETION------------------->//

int y;

del:cout << "Enter Value which u want to delete:" << endl;

cin >> y;

if (!h.search(h.root, y))

{

cout << "Value Does not exists" << endl;

goto del;

}

h.deletetion(h.root, y);

break;

case 3: //<-------------------TRAVERSALS-------------------->//

int t;

cout << "1) In Order Traversal" << endl;

cout << "2) Pre Order Traversal" << endl;

cout << "3) Post Order Traversal" << endl;

cout << "4) ALl Traversals" << endl;

cout << endl << endl;

cout << "Enter Your choice :" << endl;

cin >> t;

cout << endl;

switch (t)

{

case 1:

cout << "In Order Traversal" << endl;

h.inorder(h.root);

break;

case 2:

cout << "Pre Order Traversal" << endl;

h.Preorder(h.root);

break;

case 3:

cout << "Post Order Traversal" << endl;

h.Postorder(h.root);

break;

case 4:

cout << "In Order Traversal" << endl;

h.inorder(h.root);

cout << endl;

cout << "Pre Order Traversal" << endl;

h.Preorder(h.root);

cout << endl;

cout << "Post Order Traversal" << endl;

h.Postorder(h.root);

break;

default:

cout << "Invalid Option" << endl;

}

break;

case 4://<-------------------SEARCH---------------->//

int k;

cout << "Enter Value u want to search:" << endl;

cin >> k;

if (h.search(h.root, k))

{

cout << "Number Searched" << endl;

}

else

{

cout << "Not Searched" << endl;

}

break;

case 5://<--------------MAX MIN--------------------------->//

cout << "Maximum Value In AVL Tree:" << h.Maximum() << endl;

cout << "Minimum Value In AVL Tree:" << h.Minimum() << endl;

break;

case 6: //<------------------HEIGHT-------------------->//

cout << "Height of Tree is :" << h.height(h.root) << endl;

break;

case 7://<----------------------BALANCE FACTOR------------------>//

cout << "Balance Factor of Current Tree :" << h.getbalance(h.root) << endl;

break;

case 8://<------------------SUCESSOR/PREDECESSOR---------------->//

int n;

cout << "Enter Value:" << endl;

cin >> n;

cout << "Sucessor of the node:" << h.successor(n) << endl;

cout << "Predecessor of the node:" << h.predecessor(n) << endl;

break;

case 9://<-----------------DIAMETER-------------------------->//

cout << "Diameter of current AVL Tree:" << h.diamter() << endl;

break;

case 10://<-------------------EXIT------------------------>//

goto a;

break;

default:

cout << "Invalid Choice" << endl;

}

}

a:

cout << endl;

cout << "Program Exited successfully !!!" << endl;

return 0;

}

















