Data Structures

Lab Assignment # 11

Name: Ahmed Kasteer

Roll Number: 20F-0336

#include<iostream>

using namespace std;

class AVL\_TREE

{

public:

int data;

struct AVL\_TREE\* left;

struct AVL\_TREE\* right;

}\*root;

class AVLTREE

{

public:

int Recursive\_Height(AVL\_TREE\* temp)

{

int height = 0;

if (temp != NULL)

{

int leftheight = Recursive\_Height(temp->left);

int rightheight = Recursive\_Height(temp->right);

int maximumheight = max(leftheight, rightheight);

height = maximumheight + 1;

}

return height;

}

AVL\_TREE\* Balancing\_Factor(AVL\_TREE\* temp)

{

int balancingfactor = difference(temp);

if (balancingfactor > 1)

{

if (difference(temp->left) > 0)

temp = Recursive\_LL(temp);

else

temp = Recursive\_LR(temp);

}

else if (balancingfactor < -1)

{

if (difference(temp->right) > 0)

temp = Recursive\_LR(temp);

else

temp = Recursive\_RR(temp);

}

return temp;

}

AVL\_TREE\* Recursive\_RR(AVL\_TREE\* parent)

{

AVL\_TREE\* temp;

temp = parent->right;

parent->right = temp->left;

temp->left = parent;

return temp;

}

AVL\_TREE\* Recursive\_LL(AVL\_TREE\* parent)

{

AVL\_TREE\* temp;

temp = parent->left;

parent->left = temp->right;

temp->right = parent;

return temp;

}

AVL\_TREE\* Recursive\_RL(AVL\_TREE\* parent)

{

AVL\_TREE\* temp;

temp = parent->right;

parent->right = Recursive\_LL(temp);

return Recursive\_RR(parent);

}

AVL\_TREE\* Recursive\_LR(AVL\_TREE\* parent)

{

AVL\_TREE\* temp;

temp = parent->left;

parent->left = Recursive\_RR(temp);

return Recursive\_LL(parent);

}

AVL\_TREE\* BST\_deletion(AVL\_TREE\* t, int x)

{

AVL\_TREE\* temp;

if (t == NULL)

{

return NULL;

}

else if (x < t->data)

{

t->left = BST\_deletion(t->left, x);

}

else if (x > t->data)

{

t->right = BST\_deletion(t->right, x);

}

else if (t->left && t->right)

{

temp = Minimum(t->right);

t->data = temp->data;

t->right = BST\_deletion(t->right, t->data);

}

else {

temp = t;

if (t->left == NULL) t = t->right;

else if (t->right == NULL) t = t->left;

delete temp;

}

if (t == NULL) return t;

t = Balancing\_Factor(t);

}

AVL\_TREE\* BST\_insertion(AVL\_TREE\* root, int value)

{

if (root == NULL) {

root = new AVL\_TREE;

root->data = value;

root->left = NULL;

root->right = NULL;

return root;

}

else if (value < root->data)

{

root->left = BST\_insertion(root->left, value);

root = Balancing\_Factor(root);

}

else if (value >= root->data)

{

root->right = BST\_insertion(root->right, value);

root = Balancing\_Factor(root);

}

return root;

}

void Display\_Nodes(AVL\_TREE\* ptr, int level)

{

if (ptr != NULL)

{

Display\_Nodes(ptr->right, level + 1);

cout << endl;

if (ptr == root)

{

cout << " ";

}

else

for (int i = 0; i < level && ptr != root; i++)

cout << " ";

cout << ptr->data;

cout << endl;

Display\_Nodes(ptr->left, level + 1);

}

}

AVL\_TREE\* Minimum(AVL\_TREE\* t)

{

if (t == NULL) return NULL;

else if (t->left == NULL) return t;

else return Minimum(t->left);

}

AVL\_TREE\* Maximum(AVL\_TREE\* t)

{

if (t == NULL) return NULL;

else if (t->right == NULL) return t;

else return Maximum(t->right);

}

int difference(AVL\_TREE\* temp)

{

int leftheight = Recursive\_Height(temp->left);

int rightheight = Recursive\_Height(temp->right);

int balancingfactor = leftheight - rightheight;

return balancingfactor;

}

AVLTREE()

{

root = NULL;

}

};

int main()

{

int option;

int element;

AVLTREE avl;

while (1)

{

cout << "Press 1 for inserting the element" << endl;

cout << "Press 2 for deleting the element" << endl;

cout << "Press 3 for display" << endl;

cout << "Press 4 for exit" << endl;

cout << "Please enter the option = ";

cin >> option;

switch (option)

{

case 1:

cout << "Please enter value to insert = ";

cin >> element;

root = avl.BST\_insertion(root, element);

system("cls");

break;

case 2:

cout << "Please enter value to delete = ";

cin >> element;

root = avl.BST\_deletion(root, element);

system("cls");

break;

case 3:

if (root == NULL) {

cout << "Tree is Empty" << endl;

continue;

}

system("cls");

cout << "AVL Tree " << endl;

avl.Display\_Nodes(root, 1);

break;

case 4:

exit(1);

break;

default:

cout << "Invalid Input!" << endl;

}

}

return 0;

}