# Mahad Ghauri

233523

Lab Task 12

#include <iostream>

#include <string>

using namespace std;

class Node

{

private:

int data, height;

Node \*left, \*right;

public:

Node()

{

// Default Constructor

}

Node(int x)

{

data = x;

height = 1;

left = right = NULL;

}

void setData(int value)

{

data = value;

}

int getData()

{

return data;

}

void setLeft(Node \*other)

{

left = other;

}

Node \*getLeft()

{

return left;

}

void setRight(Node \*other)

{

right = other;

}

Node \*getRight()

{

return right;

}

void setHeight(int x)

{

height = x;

}

int getHeight()

{

return height;

}

};

int getHeight(Node \*root)

{

if (!root)

{

return 0;

}

else

{

return root->getHeight();

}

}

int getBalancingFactor(Node \*root)

{

return (getHeight(root->getLeft()) - getHeight(root->getRight()));

}

Node \*rightRotation(Node \*root)

{

Node \*child = root->getLeft();

Node \*childRight = child->getRight();

child->setRight(root);

root->setLeft(childRight);

root->setHeight(1 + max(getHeight(root->getLeft()), getHeight(root->getRight())));

child->setHeight(1 + max(getHeight(child->getLeft()), getHeight(child->getRight())));

return child;

}

Node \*leftRotation(Node \*root)

{

Node \*child = root->getRight();

Node \*childLeft = child->getLeft();

child->setLeft(root);

root->setRight(childLeft);

root->setHeight(1 + max(getHeight(root->getLeft()), getHeight(root->getRight())));

child->setHeight(1 + max(getHeight(child->getLeft()), getHeight(child->getRight())));

return child;

}

Node \*insertNode(Node \*root, int value)

{

if (!root)

{

return new Node(value);

}

if (value < root->getData())

{

root->setLeft(insertNode(root->getLeft(), value));

}

else if (value > root->getData())

{

root->setRight(insertNode(root->getRight(), value));

}

else

{

return root;

}

root->setHeight(1 + max(getHeight(root->getLeft()), getHeight(root->getRight())));

int balancingFactor = getBalancingFactor(root);

// Left Left Case

if (balancingFactor > 1 && value < root->getLeft()->getData())

{

return rightRotation(root);

}

// Right Right Case

else if (balancingFactor < -1 && value > root->getRight()->getData())

{

return leftRotation(root);

}

// Left Right Case

else if (balancingFactor > 1 && value > root->getLeft()->getData())

{

root->setLeft(leftRotation(root->getLeft()));

return rightRotation(root);

}

// Right Left Case

else if (balancingFactor < -1 && value < root->getRight()->getData())

{

root->setRight(rightRotation(root->getRight()));

return leftRotation(root);

}

else

{

return root;

}

}

void Inorder(Node \*root)

{

if (root == NULL)

return;

Inorder(root->getLeft());

cout << root->getData() << " " ;

Inorder(root->getRight());

}

int main()

{

Node \*root = NULL;

int values[] = {10, 20, 30, 40, 50, 25};

for (int value : values)

{

root = insertNode(root, value);

}

cout << "Inorder traversal of the AVL tree:" << endl;

Inorder(root);

return 0;

return 0;

}

