

DATA structures

Lab-11



Name: Talib Husain

Roll# 21F-9070

**Task-1**

#include <iostream>

using namespace std;

class Node {

public:

int key;

Node \*left;

Node \*right;

int h;

};

class AVL{

Node \*root;

public:

void insert(int key) {

root = insertNode(root,key);

}

Node \*nodeWithMimumValue(Node \*node) {

Node \*current = node;

while (current->left != NULL)

current = current->left;

return current;

}

void display() {

printTree(root," ",1);

}

private:

int height(Node\* N) {

if (N == NULL)

return 0;

return N->h;

}

Node\* insertNode(Node \*r, int key){

Node\* newNode = new Node;

newNode->left = newNode->right = NULL;

newNode->key = key;

if (r == NULL) {

return newNode;

}

if (r->key > key)

r->left = insertNode(r->left, key);

else if (key > r->key)

r->right = insertNode(r->right, key);

else

return r;

r->h = 1 + max(height(r->left),

height(r->right));

int balanceFactor = getBalanceFactor(r);

if (balanceFactor > 1) {

if (key < r->left->key) {

return rightRotate(r);

} else if (key > r->left->key) {

r->left = leftRotate(r->left);

return rightRotate(r);

}

}

if (balanceFactor < -1) {

if (key > r->right->key) {

return leftRotate(r);

} else if (key < r->right->key) {

r->right = rightRotate(r->right);

return leftRotate(r);

}

}

return r;

}

Node \*leftRotate(Node \*x) {

Node \*y = x->right;

Node \*T2 = y->left;

y->left = x;

x->right = T2;

x->h = max(height(x->left),

height(x->right)) +

1;

y->h = max(height(y->left),

height(y->right)) +

1;

return y;

}

Node \*rightRotate(Node \*y) {

Node \*x = y->left;

Node \*T2 = x->right;

x->right = y;

y->left = T2;

y->h = max(height(y->left),

height(y->right)) +

1;

x->h = max(height(x->left),

height(x->right)) +

1;

return x;

}

int getBalanceFactor(Node \*N) {

if (N == NULL)

return 0;

return height(N->left) -

height(N->right);

}

void printTree(Node\* r, string indent, bool last) {

if (r != nullptr) {

cout << indent;

if (last) {

cout << "R----";

indent += " ";

}

else {

cout << "L----";

indent += "| ";

}

cout << r->key << endl;

printTree(r->left, indent, false);

printTree(r->right, indent, true);

}

}

};

int main() {

AVL t;

t.insert(12);

t.insert(1);

t.insert(2);

t.insert(19);

t.insert(17);

t.display();

}

Output:

A picture containing text

Description automatically generated