

* ***Name*:**
* **Aqeel Abbas**
* ***Roll Number*:**
* **56**
* ***University*:**
* **Superior University**
* ***Submitted to***
* **: sir Rasikh sahab**

***Question no 1***

***.*** *Binary search tree*

#include <iostream>

using namespace std;

struct BSTNode {

int data;

BSTNode\* left;

BSTNode\* right;

BSTNode(int val) : data(val), left(NULL), right(NULL) {}

};

BSTNode\* insertBST(BSTNode\* root, int val) {

if (!root) return new BSTNode(val);

if (val < root->data)

root->left = insertBST(root->left, val);

else

root->right = insertBST(root->right, val);

return root;

}

void inorderBST(BSTNode\* root) {

if (root) {

inorderBST(root->left);

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

inorderBST(root->right);

}

}

int main() {

BSTNode\* root = NULL;

int values[] = {50, 30, 70, 20, 40, 60, 80};

for (int val : values)

root = insertBST(root, val);

cout << "Inorder Traversal of BST: ";

inorderBST(root);

return 0;

}

Avl tree.

#include <iostream>

using namespace std;

struct AVLNode {

int data, height;

AVLNode\* left;

AVLNode\* right;

AVLNode(int val) : data(val), height(1), left(NULL), right(NULL) {}

};

int height(AVLNode\* node) {

return node ? node->height : 0;

}

int getBalance(AVLNode\* node) {

return node ? height(node->left) - height(node->right) : 0;

}

AVLNode\* rightRotate(AVLNode\* y) {

AVLNode\* x = y->left;

AVLNode\* 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;

}

AVLNode\* leftRotate(AVLNode\* x) {

AVLNode\* y = x->right;

AVLNode\* 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;

}

AVLNode\* insertAVL(AVLNode\* node, int key) {

if (!node) return new AVLNode(key);

if (key < node->data)

node->left = insertAVL(node->left, key);

else if (key > node->data)

node->right = insertAVL(node->right, key);

else return node;

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

int balance = getBalance(node);

if (balance > 1 && key < node->left->data)

return rightRotate(node);

if (balance < -1 && key > node->right->data)

return leftRotate(node);

if (balance > 1 && key > node->left->data) {

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

return rightRotate(node);

}

if (balance < -1 && key < node->right->data) {

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

return leftRotate(node);

}

return node;

}

void inorderAVL(AVLNode\* root) {

if (root) {

inorderAVL(root->left);

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

inorderAVL(root->right);

}

}

int main() {

AVLNode\* root = NULL;

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

for (int val : values)

root = insertAVL(root, val);

cout << "Inorder Traversal of AVL: ";

inorderAVL(root);

return 0;

}

