**Pre-order, Post-order, and In-order Traversal of Binary Tree**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

void preOrder(Node\* root) {

if (root == nullptr) return;

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

preOrder(root->left);

preOrder(root->right);

}

void postOrder(Node\* root) {

if (root == nullptr) return;

postOrder(root->left);

postOrder(root->right);

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

}

void inOrder(Node\* root) {

if (root == nullptr) return;

inOrder(root->left);

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

inOrder(root->right);

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

cout << "Pre-order Traversal: ";

preOrder(root);

cout << "\nPost-order Traversal: ";

postOrder(root);

cout << "\nIn-order Traversal: ";

inOrder(root);

return 0;

}

**AVL Tree Implementation**

#include <iostream>

#include <algorithm>

using namespace std;

struct Node {

int data, height;

Node\* left;

Node\* right;

Node(int val) : data(val), height(1), left(nullptr), right(nullptr) {}

};

int getHeight(Node\* node) {

return node ? node->height : 0;

}

int getBalance(Node\* node) {

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

}

Node\* rotateRight(Node\* y) {

Node\* x = y->left;

Node\* T2 = x->right;

x->right = y;

y->left = T2;

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

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

return x;

}

Node\* rotateLeft(Node\* x) {

Node\* y = x->right;

Node\* T2 = y->left;

y->left = x;

x->right = T2;

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

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

return y;

}

Node\* insert(Node\* node, int key) {

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

if (key < node->data)

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

else if (key > node->data)

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

else

return node;

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

int balance = getBalance(node);

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

return rotateRight(node);

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

return rotateLeft(node);

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

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

return rotateRight(node);

}

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

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

return rotateLeft(node);

}

return node;

}

void inOrder(Node\* root) {

if (root) {

inOrder(root->left);

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

inOrder(root->right);

}

}

int main() {

Node\* root = nullptr;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 40);

root = insert(root, 50);

cout << "In-order Traversal of AVL Tree: ";

inOrder(root);

return 0;

}

**Binary Search Tree Traversal**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

Node\* insert(Node\* root, int key) {

if (!root) return new Node(key);

if (key < root->data)

root->left = insert(root->left, key);

else

root->right = insert(root->right, key);

return root;

}

void inOrder(Node\* root) {

if (root) {

inOrder(root->left);

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

inOrder(root->right);

}

}

int main() {

Node\* root = nullptr;

root = insert(root, 50);

root = insert(root, 30);

root = insert(root, 20);

root = insert(root, 40);

root = insert(root, 70);

cout << "In-order Traversal of BST: ";

inOrder(root);

return 0;

}

**Converting Binary Tree to Binary Search Tree**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int val) : data(val), left(nullptr), right(nullptr) {}

};

void storeInOrder(Node\* root, vector<int>& values) {

if (!root) return;

storeInOrder(root->left, values);

values.push\_back(root->data);

storeInOrder(root->right, values);

}

void arrayToBST(Node\* root, vector<int>& values, int& index) {

if (!root) return;

arrayToBST(root->left, values, index);

root->data = values[index++];

arrayToBST(root->right, values, index);

}

Node\* binaryToBST(Node\* root) {

vector<int> values;

storeInOrder(root, values);

sort(values.begin(), values.end());

int index = 0;

arrayToBST(root, values, index);

return root;

}

void inOrder(Node\* root) {

if (!root) return;

inOrder(root->left);

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

inOrder(root->right);

}

int main() {

Node\* root = new Node(10);

root->left = new Node(30);

root->right = new Node(15);

root->left->left = new Node(20);

root->right->right = new Node(5);

root = binaryToBST(root);

cout << "In-order Traversal of Converted BST: ";

inOrder(root);

return 0;

}