**BST Help**

#include <iostream>

#include <cstdlib>

using namespace std;

class BinarySearchTree

{

private:

struct tree\_node {

tree\_node\* left;

tree\_node\* right;

int data;

};

public:

tree\_node\* root;

BinarySearchTree() {

root = nullptr;

}

bool isEmpty() const { return root == NULL; }

void print\_inorder();

void inorder(tree\_node\*);

void print\_preorder();

void preorder(tree\_node\*);

void print\_postorder();

void postorder(tree\_node\*);

void insert(int);

tree\_node\* remove(int d, tree\_node\* node) {

//First locate the element, if the element is not found, simply generate an error message

// Then there will be three cases in removing d. Code accordingly

// 1. Removing a leaf node

// 2. Removing a node with a single child

// 3. Removing a node with 2 children, then adjust its parents and grand children links

if (node == nullptr)

return nullptr;

else if (d < node->data)

{

node->left = remove(d, node->left);

return node;

}

else if (d > node->data)

{

node->right = remove(d, node->right);

return node;

}

else if (d == node->data)

{

// Write your code here

}

}

void BinarySearchTree::insert(int d) {

tree\_node\* t = new tree\_node;

tree\_node\* parent = root;

t->data = d;

t->left = nullptr;

t->right = nullptr;

if (root == nullptr) {

root = t;

return;

}

// Smaller elements should go to the left, whereas larger elements should go to the right

else {

// Write your code here

}

}

void BinarySearchTree::print\_inorder() {

inorder(root);

cout << endl;

}

void BinarySearchTree::inorder(tree\_node\* p) {

if (p != nullptr)

{

inorder(p->left);

cout << p->data << "\t";

inorder(p->right);

}

}

void BinarySearchTree::print\_preorder() {

preorder(root);

cout << endl;

}

void BinarySearchTree::preorder(tree\_node\* p) {

// Write your code here

}

void BinarySearchTree::print\_postorder() {

postorder(root);

cout << endl;

}

void BinarySearchTree::postorder(tree\_node\* p) {

// Write your code here

}

void insertIntoTree(BinarySearchTree\* b) {

cout << "How many elements do you want to insert?" << endl;

int n;

cin >> n;

int val;

for (int i = 0; i < n; i++)

{

cout << "Enter element # " << i << ":" << "\t";

cin >> val;

b->insert(val);

}

}

int main()

{

BinarySearchTree b;

while (true) {

int menuOption = 0;

while (menuOption > 6 || menuOption < 1) //This loop can exit program

{

cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "\t\* Select Option:\t\t\t\*" << endl;

cout << "\t\*\t1. Insert New Data\t\t\*" << endl;

cout << "\t\*\t2. In-Order Traversal\t\t\*" << endl;

cout << "\t\*\t3. Pre-Order Traversal\t\t\*" << endl;

cout << "\t\*\t4. Post-Order Traversal\t\t\*" << endl;

cout << "\t\*\t5. Delete an item\t\t\*" << endl;

cout << "\t\*\t6. Exit\t\t\t\t\*" << endl;

cout << "\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "> ";

cin >> menuOption;

if (menuOption == 6) {

return 0; //Exiting

}

switch (menuOption)

{

case 1:

{

insertIntoTree(&b);

break;

}

case 2:

{

b.print\_inorder();

break;

}

case 3:

{

b.print\_preorder();

break;

}

case 4:

{

b.print\_postorder();

break;

}

case 5:

{

cout << "Enter element to remove:" << endl;

int elem;

cin >> elem;

b.remove(elem, b.root);

}

}

}

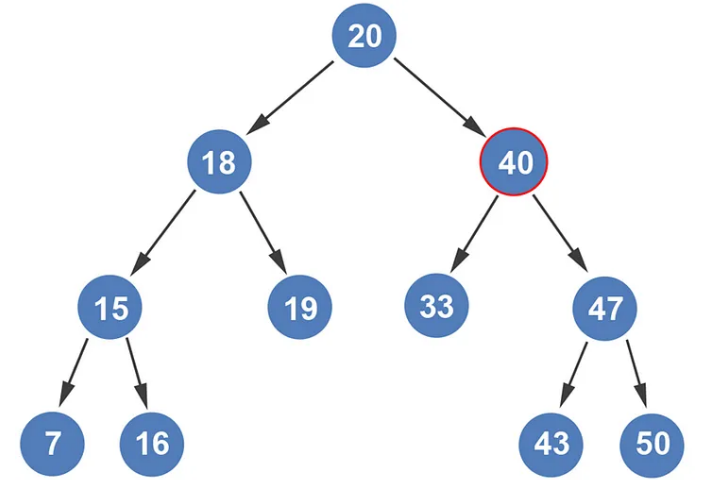
}

getchar();

}

**Deletion of 2 child:**

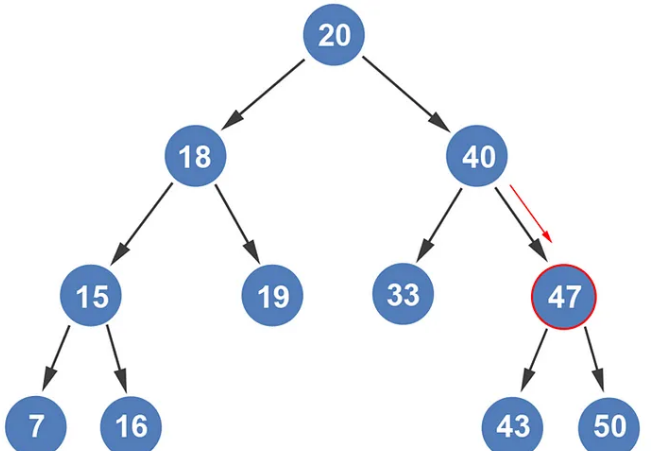
What would happen if we wanted to remove node 40 from the tree below?



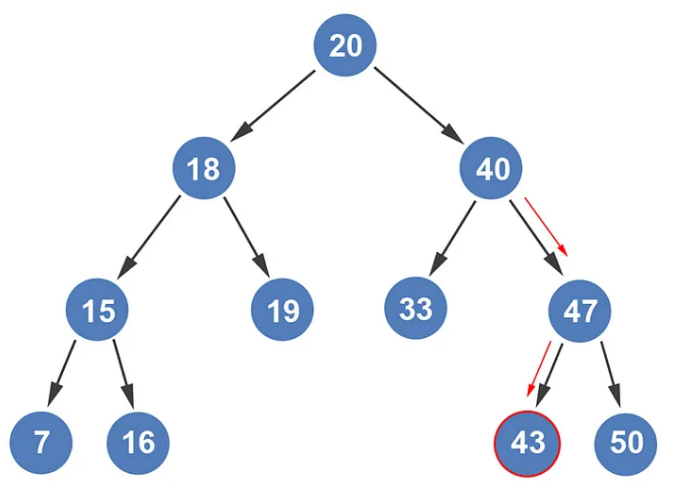
7 15 16 18 19 20 33 40 43 47 50

Since node 40 has two children, we can’t simply point to its child and hope it works. There’s an algorithmic approach to this problem too. The way we remove this node is by replacing it with the next largest value. How do we get the next largest value? By going into the right subtree, and then following the left subtree until we get to the leaf node.

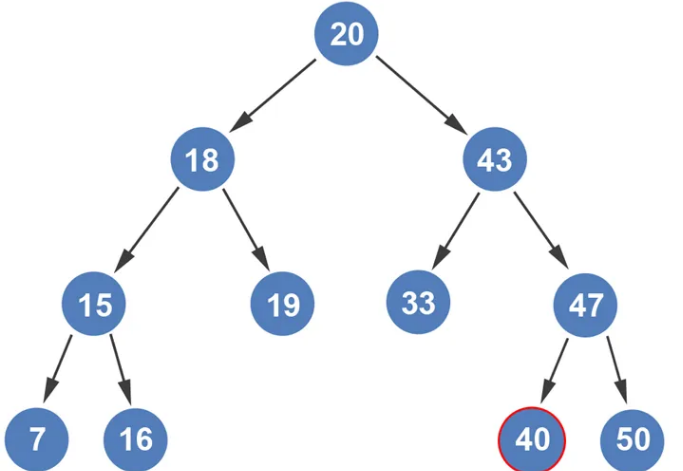
We enter node 40’s right subtree.



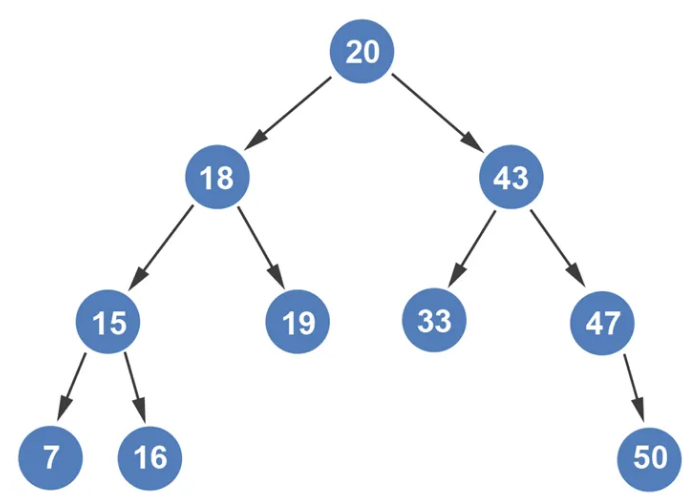
We then follow the left subtree path until we get to a leaf node. So, we go to node 43.



Since node 43 is a leaf node, we swap node 40 with node 43.



We can now remove node 40 from the tree.



7 15 16 18 19 20 33 40 43 47 50