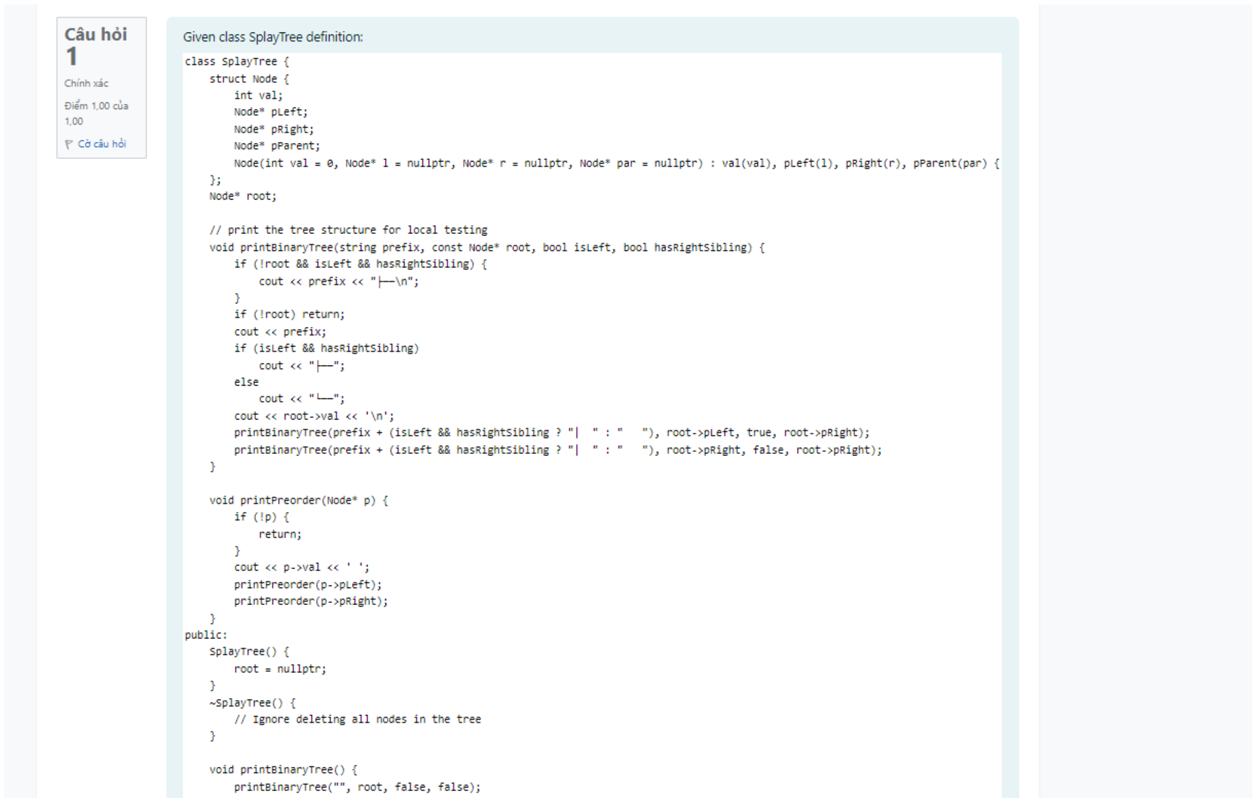


Câu hỏi **1**

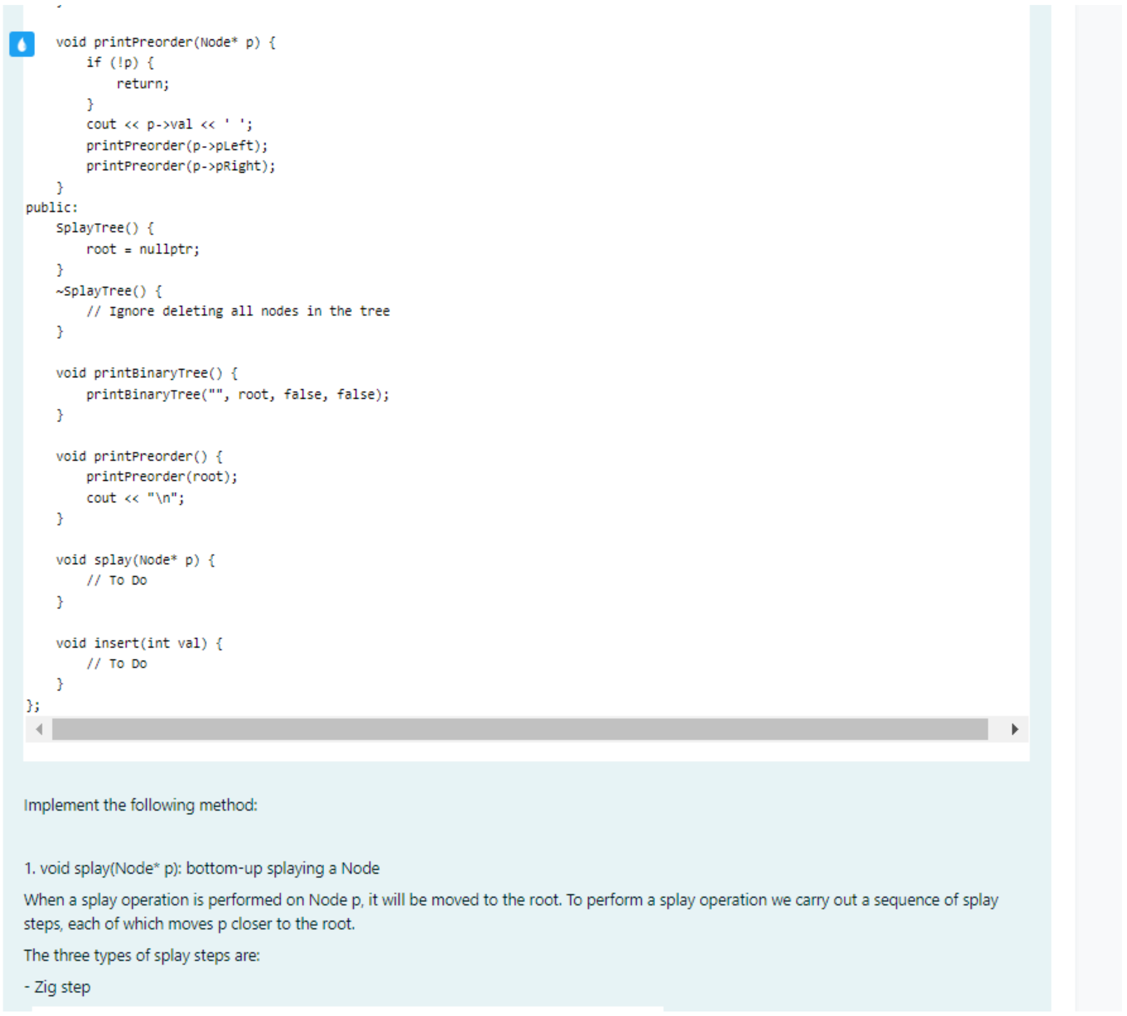
Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

Given class SplayTree definition:

```
class SplayTree {
    struct Node {
        int val;
        Node* pLeft;
        Node* pRight;
        Node* pParent;
        Node(int val = 0, Node* l = nullptr, Node* r = nullptr, Node* par = nullptr) : val(val), pLeft(l), pRight(r), pParent(par) {
    };
    Node* root;

    // print the tree structure for local testing
    void printBinaryTree(string prefix, const Node* root, bool isLeft, bool hasRightSibling) {
        if (!root && isLeft && hasRightSibling) {
            cout << prefix << "├──\n";
        }
        if (!root) return;
        cout << prefix;
        if (isLeft && hasRightSibling)
            cout << "├──";
        else
            cout << "└──";
        cout << root->val << '\n';
        printBinaryTree(prefix + (isLeft && hasRightSibling ? "|   " : "    "), root->pLeft, true, root->pRight);
        printBinaryTree(prefix + (isLeft && hasRightSibling ? "|   " : "    "), root->pRight, false, root->pRight);
    }

    void printPreorder(Node* p) {
        if (!p) {
            return;
        }
        cout << p->val << ' ';
        printPreorder(p->pLeft);
        printPreorder(p->pRight);
    }
public:
    SplayTree() {
        root = nullptr;
    }
    ~SplayTree() {
        // Ignore deleting all nodes in the tree
    }

    void printBinaryTree() {
        printBinaryTree("", root, false, false);
```



```
    void printPreorder(Node* p) {
        if (!p) {
            return;
        }
        cout << p->val << ' ';
        printPreorder(p->pLeft);
        printPreorder(p->pRight);
    }
public:
    SplayTree() {
        root = nullptr;
    }
    ~SplayTree() {
        // Ignore deleting all nodes in the tree
    }

    void printBinaryTree() {
        printBinaryTree("", root, false, false);
    }

    void printPreorder() {
        printPreorder(root);
        cout << "\n";
    }

    void splay(Node* p) {
        // To Do
    }

    void insert(int val) {
        // To Do
    }
};
```

Implement the following method:

1. void splay(Node* p): bottom-up splaying a Node

When a splay operation is performed on Node p, it will be moved to the root. To perform a splay operation we carry out a sequence of splay steps, each of which moves p closer to the root.

The three types of splay steps are:

- Zig step

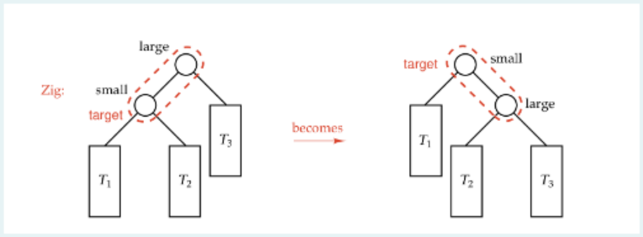
Implement the following method:

1. void splay(Node* p): bottom-up splaying a Node

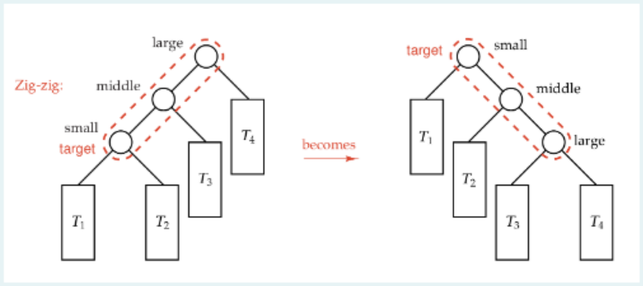
When a splay operation is performed on Node p, it will be moved to the root. To perform a splay operation we carry out a sequence of splay steps, each of which moves p closer to the root.

The three types of splay steps are:

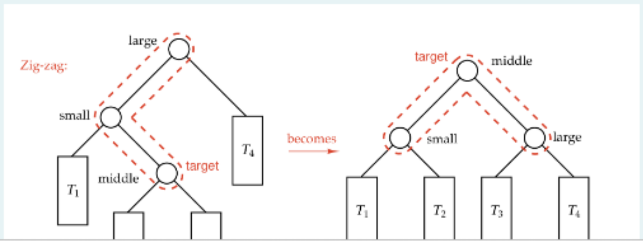
- Zig step



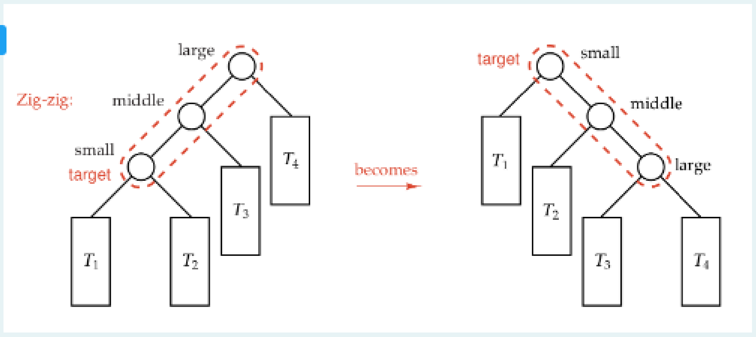
- Zig-zig step:



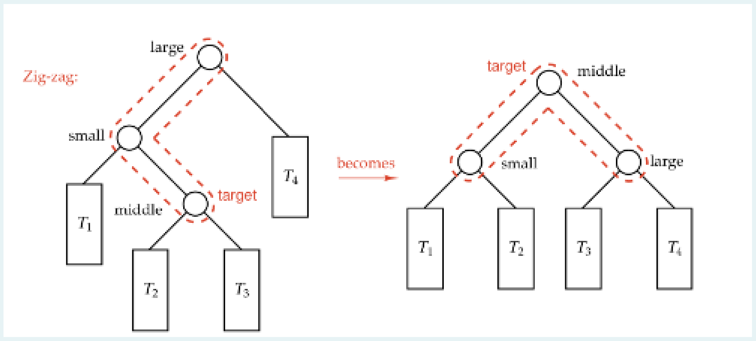
- Zig-zag step:



- Zig-zig step:



- Zig-zag step:



Note: there are also zag, zag-zag and zag-zig step but we don't show them here

2. void insert(int val):

To insert a value val into a splay tree:

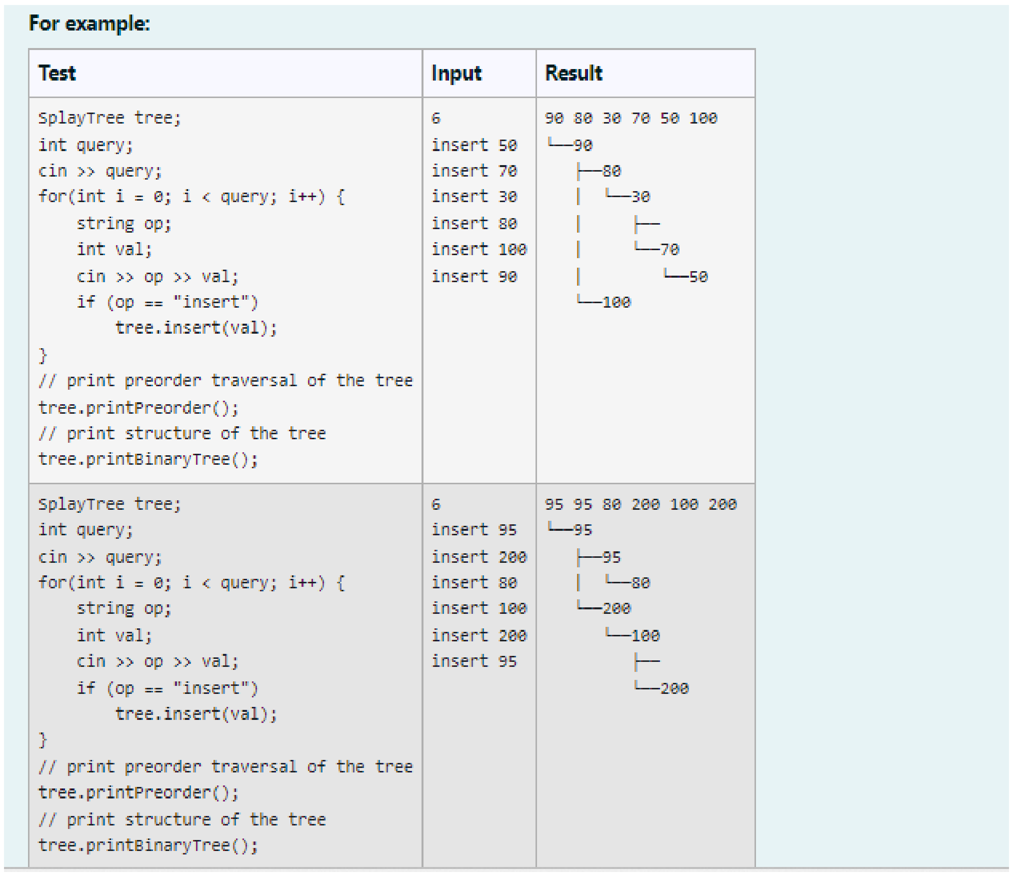
+ Insert val as with a normal binary search tree.
+ When the new value is inserted, a splay operation is performed. As a result, the newly inserted node becomes the root of the tree.

Note: In a splay tree, the values the in left subtree <= root's value <= the values in the right subtree. In this exercise, when inserting a duplicate value, you have to insert it to the right subtree to pass the testcases.

Constraint of testcases:

+ number of operation <= 10^4
+ 1 <= val <= 10^5

For example:

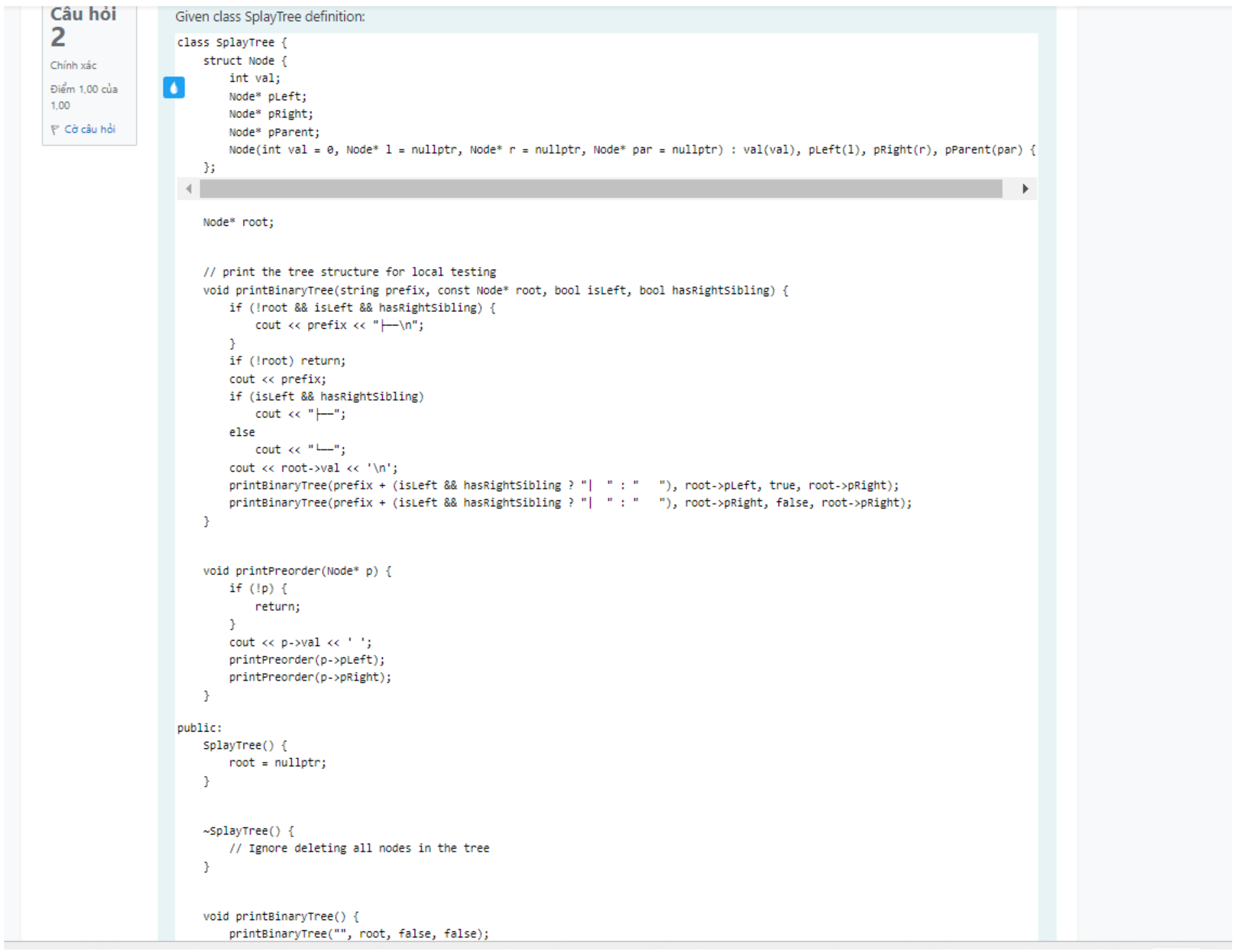


For example:

Test	Input	Result

SplayTree tree;
int query;
cin >> query;
for(int i = 0; i < query; i++) {
    string op;
    int val;
    cin >> op >> val;
    if (op == "insert")
        tree.insert(val);
}
// print preorder traversal of the tree
tree.printPreorder();
// print structure of the tree
tree.printBinaryTree();
	6
	insert 50
	insert 70
	insert 30
	insert 80
	insert 100
	insert 90
		90 80 30 70 50 100
		└──90
		    ├──80
		    | └──30
		    | ├──
		    | └──70
		    | └──50
		    └──100

SplayTree tree;
int query;
cin >> query;
for(int i = 0; i < query; i++) {
    string op;
    int val;
    cin >> op >> val;
    if (op == "insert")
        tree.insert(val);
}
// print preorder traversal of the tree
tree.printPreorder();
// print structure of the tree
tree.printBinaryTree();
	6
	insert 95
	insert 200
	insert 80
	insert 100
	insert 200
	insert 95
		95 95 80 200 100 200
		└──95
		    ├──95
		    | └──80
		    └──200
		        └──100
		            ├──
		            └──200



Câu hỏi 2

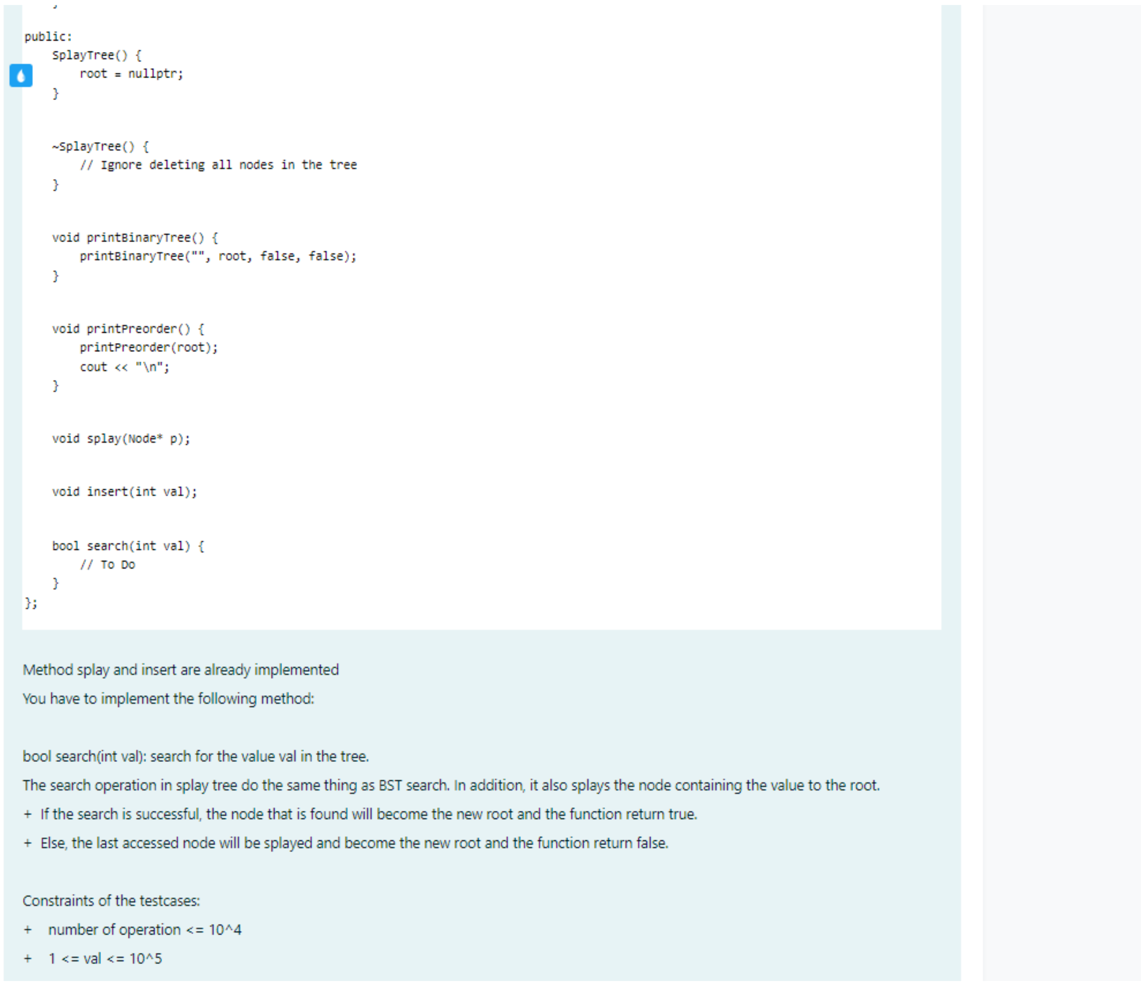
Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

Given class SplayTree definition:

```
class SplayTree {
    struct Node {
        int val;
        Node* pLeft;
        Node* pRight;
        Node* pParent;
        Node(int val = 0, Node* l = nullptr, Node* r = nullptr, Node* par = nullptr) : val(val), pLeft(l), pRight(r), pParent(par) {
    };

    Node* root;


    // print the tree structure for local testing
    void printBinaryTree(string prefix, const Node* root, bool isLeft, bool hasRightSibling) {
        if (!root && isLeft && hasRightSibling) {
            cout << prefix << "├──\n";
        }
        if (!root) return;
        cout << prefix;
        if (isLeft && hasRightSibling)
            cout << "├──";
        else
            cout << "└──";
        cout << root->val << '\n';
        printBinaryTree(prefix + (isLeft && hasRightSibling ? "|   " : "    "), root->pLeft, true, root->pRight);
        printBinaryTree(prefix + (isLeft && hasRightSibling ? "|   " : "    "), root->pRight, false, root->pRight);
    }


    void printPreorder(Node* p) {
        if (!p) {
            return;
        }
        cout << p->val << ' ';
        printPreorder(p->pLeft);
        printPreorder(p->pRight);
    }

public:
    SplayTree() {
        root = nullptr;
    }


    ~SplayTree() {
        // Ignore deleting all nodes in the tree
    }


    void printBinaryTree() {
        printBinaryTree("", root, false, false);
```



```
public:
    SplayTree() {
        root = nullptr;
    }

    ~SplayTree() {
        // Ignore deleting all nodes in the tree
    }

    void printBinaryTree() {
        printBinaryTree("", root, false, false);
    }

    void printPreorder() {
        printPreorder(root);
        cout << "\n";
    }

    void splay(Node* p);

    void insert(int val);

    bool search(int val) {
        // To Do
    }
};
```

Method splay and insert are already implemented

You have to implement the following method:

bool search(int val): search for the value val in the tree.

The search operation in splay tree do the same thing as BST search. In addition, it also splays the node containing the value to the root.

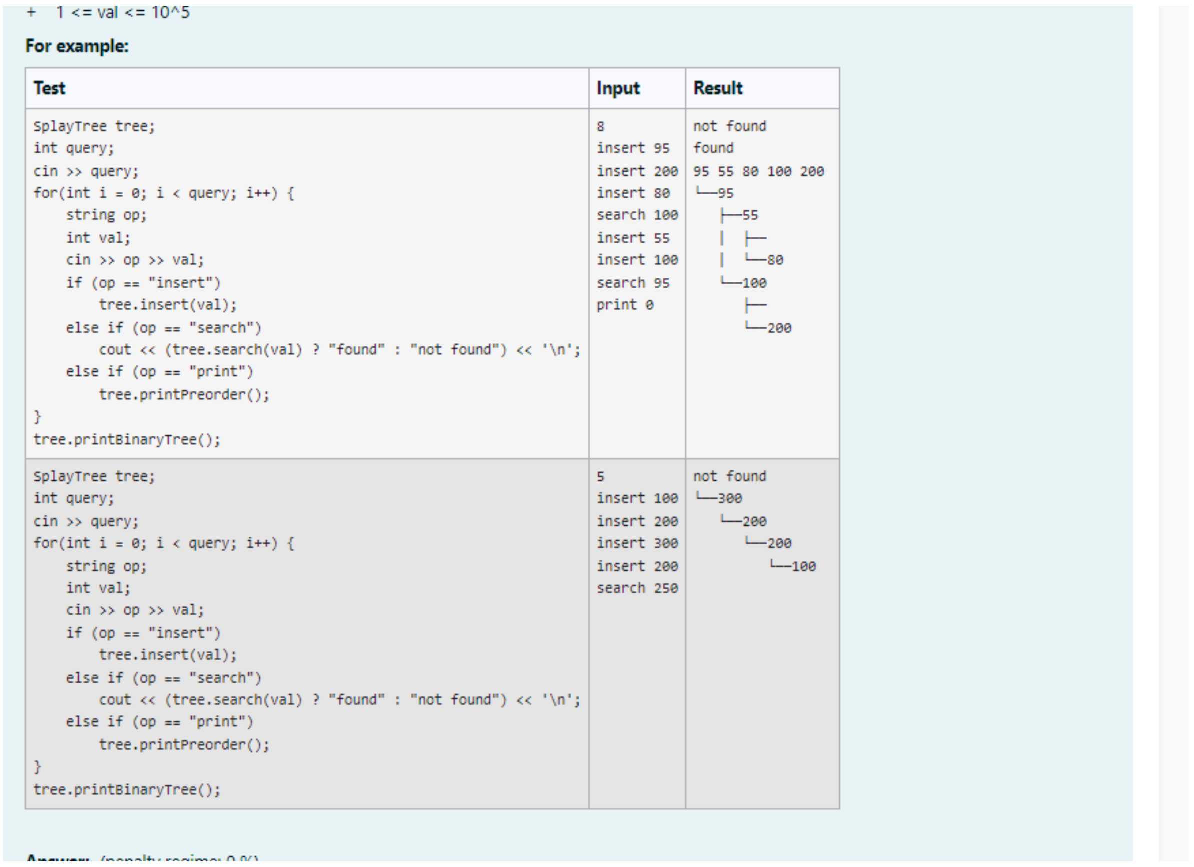
+ If the search is successful, the node that is found will become the new root and the function return true.

+ Else, the last accessed node will be splayed and become the new root and the function return false.

Constraints of the testcases:

+ number of operation <= 10^4

+ 1 <= val <= 10^5



+ 1 <= val <= 10^5

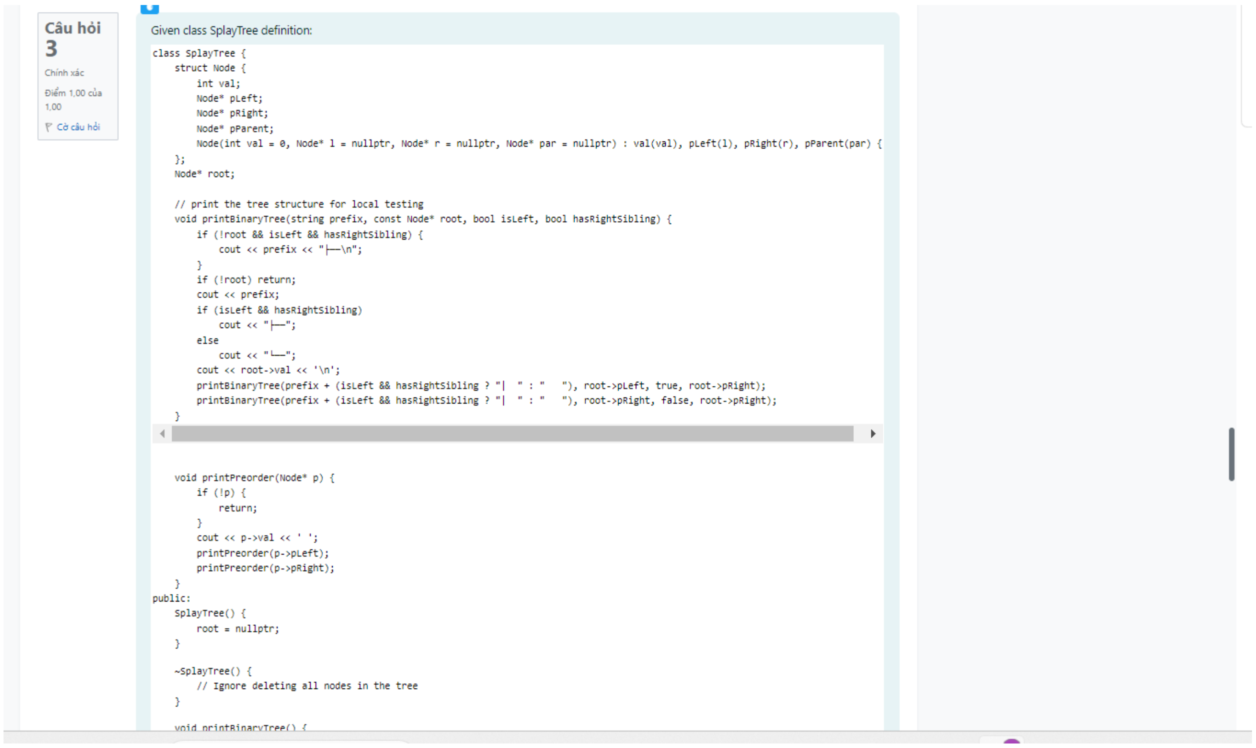
For example:

Test	Input	Result

`SplayTree tree;`
`int query;`
`cin >> query;`
`for(int i = 0; i < query; i++) {`
`    string op;`
`    int val;`
`    cin >> op >> val;`
`    if (op == "insert")`
`        tree.insert(val);`
`    else if (op == "search")`
`        cout << (tree.search(val) ? "found" : "not found") << '\n';`
`    else if (op == "print")`
`        tree.printPreorder();`
`}`
`tree.printBinaryTree();`
	8
	insert 95
	insert 200
	insert 80
	search 100
	insert 55
	insert 100
	search 95
	print 0
		not found
		found
		95 55 80 100 200
		└──95
		    ├──55
		    │ ├──
		    │ └──80
		    └──100
		        ├──
		        └──200

`SplayTree tree;`
`int query;`
`cin >> query;`
`for(int i = 0; i < query; i++) {`
`    string op;`
`    int val;`
`    cin >> op >> val;`
`    if (op == "insert")`
`        tree.insert(val);`
`    else if (op == "search")`
`        cout << (tree.search(val) ? "found" : "not found") << '\n';`
`    else if (op == "print")`
`        tree.printPreorder();`
`}`
`tree.printBinaryTree();`
	5
	insert 100
	insert 200
	insert 300
	insert 200
	search 250
		not found
		└──300
		    └──200
		        └──200
		            └──100

Answer: (penalty regime: 0 %)



Câu hỏi **3**

Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

Given class SplayTree definition:

```
class SplayTree {
    struct Node {
        int val;
        Node* pLeft;
        Node* pRight;
        Node* pParent;
        Node(int val = 0, Node* l = nullptr, Node* r = nullptr, Node* par = nullptr) : val(val), pLeft(l), pRight(r), pParent(par) {
    };
    Node* root;

    // print the tree structure for local testing
    void printBinaryTree(string prefix, const Node* root, bool isLeft, bool hasRightSibling) {
        if (!root && isLeft && hasRightSibling) {
            cout << prefix << "├──\n";
        }
        if (!root) return;
        cout << prefix;
        if (isLeft && hasRightSibling)
            cout << "├──";
        else
            cout << "└──";
        cout << root->val << '\n';
        printBinaryTree(prefix + (isLeft && hasRightSibling ? "|   " : "    "), root->pLeft, true, root->pRight);
        printBinaryTree(prefix + (isLeft && hasRightSibling ? "|   " : "    "), root->pRight, false, root->pRight);
    }


    void printPreorder(Node* p) {
        if (!p) {
            return;
        }
        cout << p->val << ' ';
        printPreorder(p->pLeft);
        printPreorder(p->pRight);
    }
public:
    SplayTree() {
        root = nullptr;
    }

    ~SplayTree() {
        // Ignore deleting all nodes in the tree
    }

    void printBinaryTree() {
```

```cpp
        cout << prefix;
        if (isLeft && hasRightSibling)
            cout << "├──";
        else
            cout << "└──";
        cout << root->val << '\n';
        printBinaryTree(prefix + (isLeft && hasRightSibling ? "|   " : "    "), root->pLeft, true, root->pRight);
        printBinaryTree(prefix + (isLeft && hasRightSibling ? "|   " : "    "), root->pRight, false, root->pRight);
    }

    void printPreorder(Node* p) {
        if (!p) {
            return;
        }
        cout << p->val << ' ';
        printPreorder(p->pLeft);
        printPreorder(p->pRight);
    }
public:
    SplayTree() {
        root = nullptr;
    }

    ~SplayTree() {
        // Ignore deleting all nodes in the tree
    }

    void printBinaryTree() {
        printBinaryTree("", root, false, false);
    }

    void printPreorder() {
        printPreorder(root);
        cout << "\n";
    }

    void splay(Node* p);

    void insert(int val);

    bool search(int val);

    Node* remove(int val) {
        // To Do
    }
};
```

```cpp
};
```

The methods splay, insert and search are already implemented.

Implement the following method:

Node* remove(int val): remove the first Node with value equal to val from the tree and return it.

To perform remove operation on splay tree:

1. If root is NULL, return the root

2. Search for the first node containing the given value val and splay it. If val is present, the found node will become the root. Else the last accessed leaf node becomes the root.

3. If new root's value is not equal to val, return NULL as val is not present.

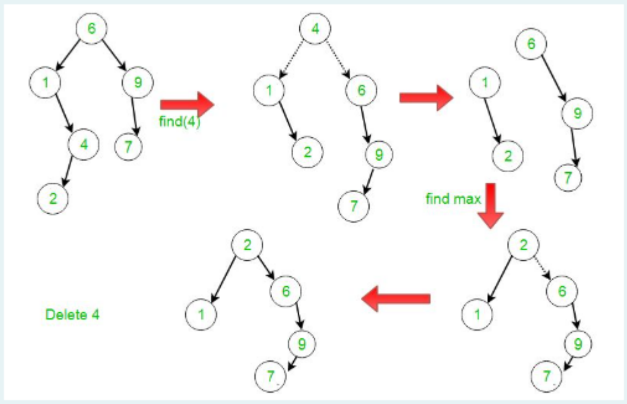
4. Else the value val is present, we remove root from the tree by the following steps:

    4.1 Split the tree into two tree: tree1 = root's left subtree and tree2 = root's right subtree

    4.2 If tree1 is NULL, tree2 is the new root

    4.3 Else, splay the leaf node with the largest value in tree1. tree1 will be a left skewed binary tree. Make tree2 the right subtree of tree1. tree1 becomes the new root

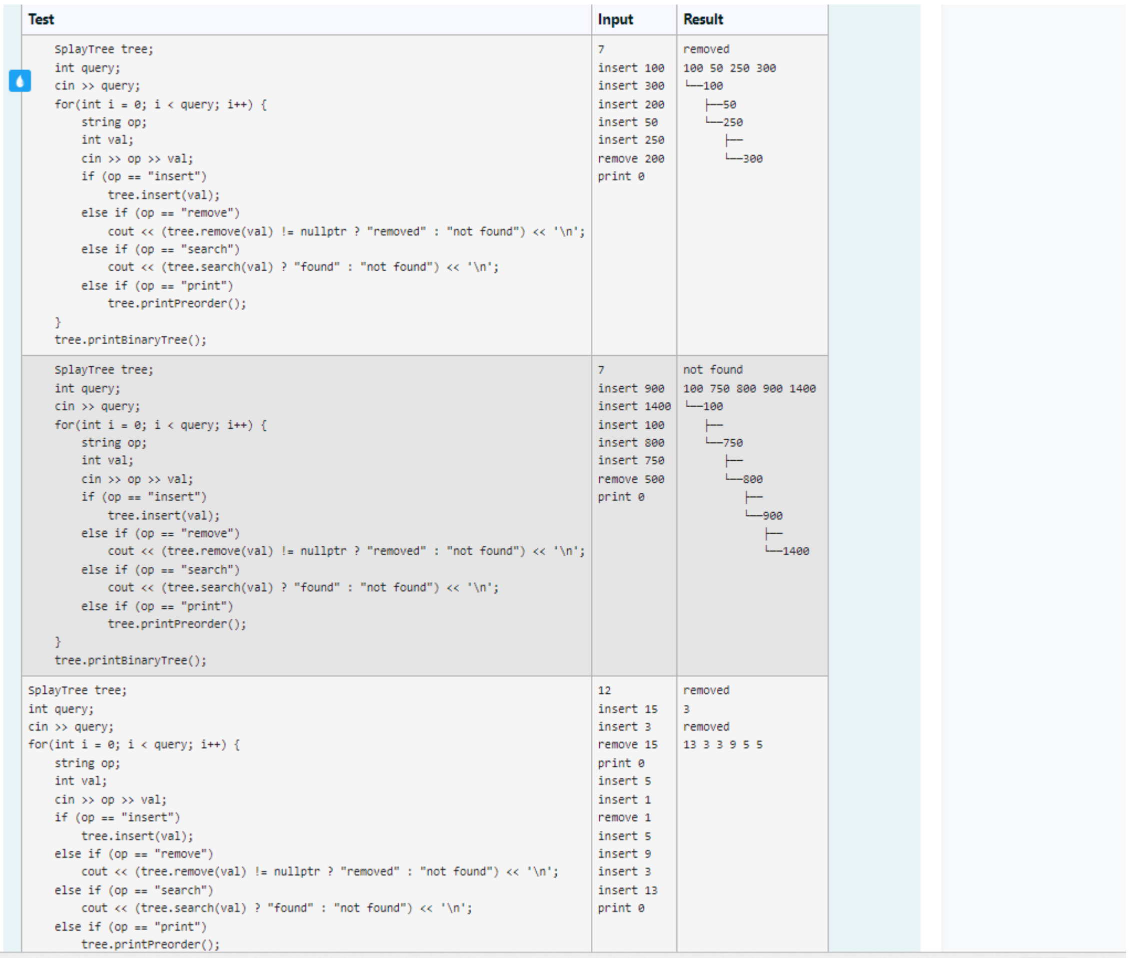
    4.4 Return the removed node.



Constraints of the testcases:

+ number of operations <= 10^4

+ 1 <= val <= 10^5



Test | **Input** | **Result**

Test
```
    SplayTree tree;
    int query;
    cin >> query;
    for(int i = 0; i < query; i++) {
        string op;
        int val;
        cin >> op >> val;
        if (op == "insert")
            tree.insert(val);
        else if (op == "remove")
            cout << (tree.remove(val) != nullptr ? "removed" : "not found") << '\n';
        else if (op == "search")
            cout << (tree.search(val) ? "found" : "not found") << '\n';
        else if (op == "print")
            tree.printPreorder();
    }
    tree.printBinaryTree();
```
Input
```
7
insert 100
insert 300
insert 200
insert 50
insert 250
remove 200
print 0
```
Result
```
removed
100 50 250 300
└──100
    ├──50
    └──250
        ├──
        └──300
```

Test
```
    SplayTree tree;
    int query;
    cin >> query;
    for(int i = 0; i < query; i++) {
        string op;
        int val;
        cin >> op >> val;
        if (op == "insert")
            tree.insert(val);
        else if (op == "remove")
            cout << (tree.remove(val) != nullptr ? "removed" : "not found") << '\n';
        else if (op == "search")
            cout << (tree.search(val) ? "found" : "not found") << '\n';
        else if (op == "print")
            tree.printPreorder();
    }
    tree.printBinaryTree();
```
Input
```
7
insert 900
insert 1400
insert 100
insert 800
insert 750
remove 500
print 0
```
Result
```
not found
100 750 800 900 1400
└──100
    ├──
    └──750
        ├──
        └──800
            ├──
            └──900
                ├──
                └──1400
```

Test
```
SplayTree tree;
int query;
cin >> query;
for(int i = 0; i < query; i++) {
    string op;
    int val;
    cin >> op >> val;
    if (op == "insert")
        tree.insert(val);
    else if (op == "remove")
        cout << (tree.remove(val) != nullptr ? "removed" : "not found") << '\n';
    else if (op == "search")
        cout << (tree.search(val) ? "found" : "not found") << '\n';
    else if (op == "print")
        tree.printPreorder();
```
Input
```
12
insert 15
insert 3
remove 15
print 0
insert 5
insert 1
remove 1
insert 5
insert 9
insert 3
insert 13
print 0
```
Result
```
removed
3
removed
13 3 3 9 5 5
```