Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Ba, 6 tháng 5 2025, 3:07 PM
Kết thúc lúc	Thứ Tư, 7 tháng 5 2025, 1:14 AM
Thời gian thực hiện	10 giờ 6 phút
Điểm	3,00/3,00
Điểm	**10,00** trên 10,00 (**100**%)

Câu hỏi 1

Đúng

Đạt điểm 1,00 trên 1,00

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

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

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

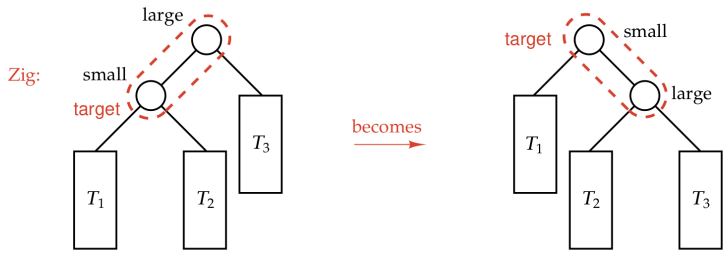
Implement the following method:

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

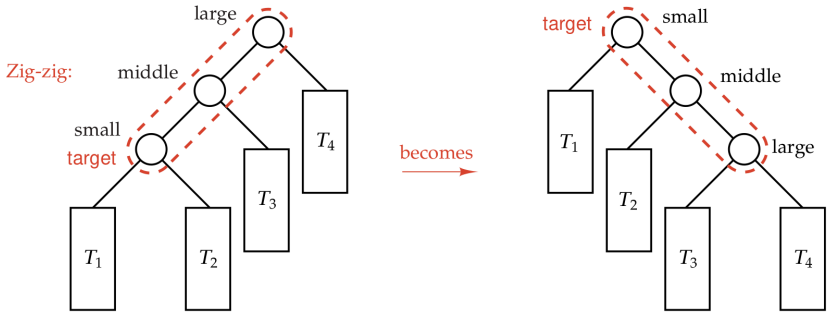
When a splay operation is performed on Node p, it will be moved to the root. To perform a splay operation we carry out a sequence of splay steps, each of which moves p closer to the root.

The three types of splay steps are:

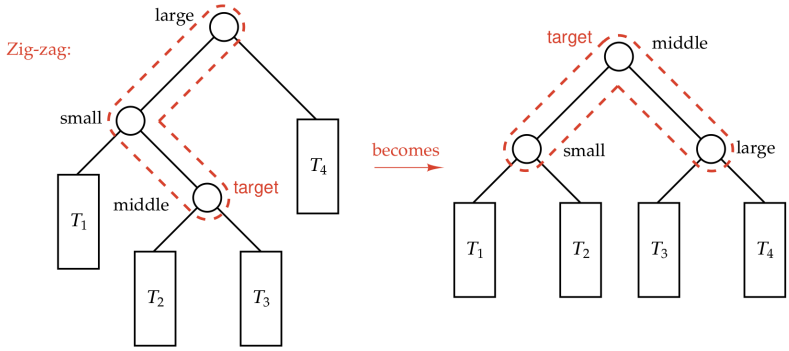
- Zig step



- 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:

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

Answer: (penalty regime: 0 %)

Reset answer

```
// Function Helping
/*
     * Xoay phải node p
     * Trước khi xoay:        Sau khi xoay:
     *      P                      L
     *     / \                    / \
     *    L   R     --->         X   P
     *   / \                        / \
     *  X   Y                      Y   R
     */
    void rightRotate(Node* p) {
        // Không thể xoay nếu p hoặc con trái của p là nullptr
        if (!p || !p->pLeft) return;

        // Lưu tham chiếu đến con trái của p
        Node* l = p->pLeft;

        // Cập nhật con trái của p thành con phải của l
        p->pLeft = l->pRight;
        if (l->pRight) {
            l->pRight->pParent = p;
        }

        // Cập nhật cha của l
        l->pParent = p->pParent;

        // Nếu p là root
        if (!p->pParent) {
            root = l;
        }
        // Nếu p là con trái của cha nó
        else if (p == p->pParent->pLeft) {
            p->pParent->pLeft = l;
        }
        // Nếu p là con phải của cha nó
        else {
            p->pParent->pRight = l;
        }

        // Đặt p là con phải của l và cập nhật cha của p
```

```
41          l->pRight = p;
42          p->pParent = l;
43      }
44
45      /*
46       * Xoay trái node p
47       * Trước khi xoay:        Sau khi xoay:
48       *      P                      R
49       *     / \                    / \
50       *    L   R     --->         P   Y
51       *       / \                / \
52       *      X   Y              L   X
```

	Test	Input	Expected	Got	

✓	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
				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
				95 95 80 200 100 200
				└─95
				├─95
				│ └─80
				└─200
				└─100
				├─
				└─200
					✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **2**

Đúng

Đạt điểm 1,00 trên 1,00

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() {
        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

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 %)

Reset answer

```
/*
     * Phương thức search tìm kiếm một giá trị trong cây Splay
     * 1. Tìm kiếm như trong BST thông thường
     * 2. Nếu tìm thấy, splay node đó lên làm gốc và trả về true
     * 3. Nếu không tìm thấy, splay node cuối cùng được truy cập lên làm gốc và trả về false
     */
    bool search(int val) {
        // Nếu cây rỗng, trả về false
```

```
        if (!root) {
            return false;
        }

        Node* curr = root;
        Node* lastAccessed = nullptr;  // Lưu node cuối cùng được truy cập

        // Tìm kiếm như trong BST thông thường
        while (curr) {
            lastAccessed = curr;  // Cập nhật node cuối cùng được truy cập

            // Nếu tìm thấy giá trị
            if (val == curr->val) {
                splay(curr);  // Splay node này lên làm gốc
                return true;  // Trả về true
            }

            // Nếu giá trị cần tìm nhỏ hơn node hiện tại, đi sang cây con trái
            if (val < curr->val) {
                curr = curr->pLeft;
            }
            // Nếu giá trị cần tìm lớn hơn node hiện tại, đi sang cây con phải
            else {
                curr = curr->pRight;
            }
        }

        // Nếu không tìm thấy, splay node cuối cùng được truy cập lên làm gốc
        splay(lastAccessed);
        return false;  // Trả về false
    }
```

	Test	Input	Expected	Got	

✓	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
				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
				not found
				└─300
				└─200
				└─200
				└─100
					✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **3**

Đúng

Đạt điểm 1,00 trên 1,00

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

```
    void splay(Node* p);
```

```
    void insert(int val);
```

```
    bool search(int val);
```

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

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.

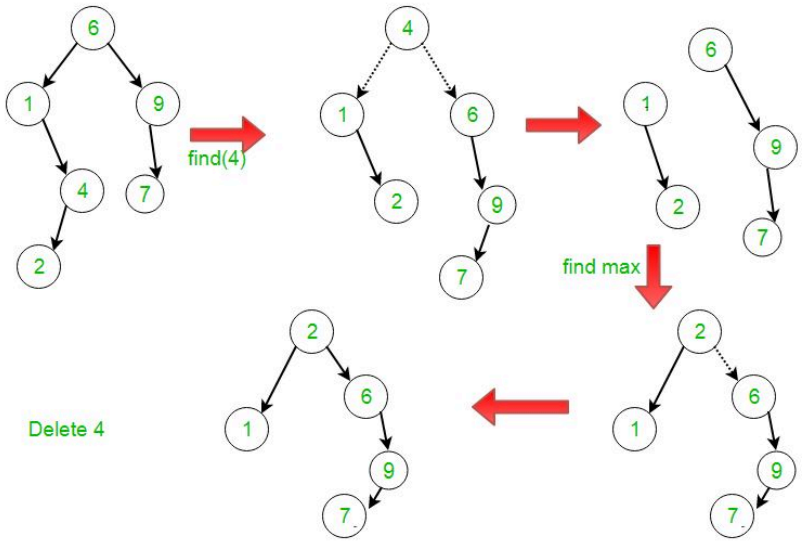
4. Else the value val is present, we remove root from the tree by the following steps:

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

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 == "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();
	7
	insert 100
	insert 300
	insert 200
	insert 50
	insert 250
	remove 200
	print 0
		removed
		100 50 250 300
		└─100
		├─50
		└─250
		├─
		└─300

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 == "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();
	7
	insert 900
	insert 1400
	insert 100
	insert 800
	insert 750
	remove 500
	print 0
		not found
		100 750 800 900 1400
		└─100
		├─
		└─750
		├─
		└─800
		├─
		└─900
		├─
		└─1400

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();
}
	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
		removed
		3
		removed
		13 3 3 9 5 5

Answer: (penalty regime: 0. %)

Reset answer

```
    /**3.
     * Tìm node có giá trị lớn nhất trong cây con
     * Được sử dụng trong quá trình xóa node
     *
     * @param p Node gốc của cây con
     * @return Node có giá trị lớn nhất trong cây con
     */
    Node *findMax(Node *p)
    {
        // Nếu cây con rỗng, trả về nullptr
        if (!p)
            return nullptr;

        // Node lớn nhất nằm ở phía ngoài cùng bên phải
        while (p->pRight)
        {
            p = p->pRight;
        }

        return p;
    }
/**6.
     * Phương thức xóa một node có giá trị cho trước khỏi cây Splay
     *
     * Thuật toán:
     * 1. Tìm kiếm node có giá trị cần xóa và splay nó lên làm gốc
     * 2. Nếu không tìm thấy, trả về nullptr
     * 3. Nếu tìm thấy, tách cây thành 2 cây con: cây con trái và cây con phải
     * 4. Nếu cây con trái rỗng, cây con phải trở thành cây mới
     * 5. Nếu cây con trái không rỗng, tìm node lớn nhất trong cây con trái và splay nó lên làm gốc của cây c
     * 6. Gắn cây con phải vào là con phải của gốc mới của cây con trái
     * 7. Trả về node đã xóa
     *
     * Độ phức tạp: O(h) với h là chiều cao của cây
```

```
 */
Node *remove(int val)
{
    // Trường hợp 1: Cây rỗng, không thể xóa
    if (!root)
    {
        return nullptr;
    }

    // Bước 1: Tìm kiếm node có giá trị cần xóa và splay nó lên làm gốc
    bool found = search(val);

    // Bước 2: Nếu không tìm thấy node có giá trị val, trả về nullptr
    if (!found || root->val != val)
    {
        return nullptr;
    }
```

	Test	Input	Expected	Got	

✓	`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();`
		7
		insert
		100
		insert
		300
		insert
		200
		insert 50
		insert
		250
		remove
		200
		print 0
			removed
			100 50 250 300
			└─100
			├─50
			└─250
			├─
			└─300
				removed
				100 50 250 300
				└─100
				├─50
				└─250
				├─
				└─300
					✓

✓	`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();`
		7
		insert
		900
		insert
		1400
		insert
		100
		insert
		800
		insert
		750
		remove
		500
		print 0
			not found
			100 750 800 900
			1400
			└─100
			├─
			└─750
			├─
			└─800
			├─
			└─900
			├─
			
			└─1400
				not found
				100 750 800 900
				1400
				└─100
				├─
				└─750
				├─
				└─800
				├─
				└─900
				├─
				
				└─1400
					✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.