Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Sáu, 22 tháng 3 2024, 10:15 PM
Kết thúc lúc	Thứ Sáu, 22 tháng 3 2024, 10:21 PM
Thời gian thực hiện	6 phút 7 giây

Câu hỏi 1

Đúng

Đạt điểm 1,00

In this question, you have to perform add **and delete on binary search tree**. Note that:

- When deleting a node which still have 2 children, **take the inorder successor** (smallest node of the right sub tree of that node) to replace it.

- When adding a node which has the same value as parent node, add it in the **left sub tree**.

Your task is to implement two functions: add and deleteNode. You could define one or more functions to achieve this task.

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"
template<class T>
class BinarySearchTree
{
public:
    class Node;
private:
    Node* root;
public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    //Helping function

    void add(T value){
        //TODO
    }

    void deleteNode(T value){
        //TODO
    }
    string inOrderRec(Node* root) {
        stringstream ss;
        if (root != nullptr) {
            ss << inOrderRec(root->pLeft);
            ss << root->value << " ";
            ss << inOrderRec(root->pRight);
        }
        return ss.str();
    }

    string inOrder(){
        return inOrderRec(this->root);
    }

    class Node
    {
    private:
        T value;
        Node* pLeft, * pRight;
        friend class BinarySearchTree<T>;
    public:
```

```
        Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
};
```

For example:

Test	Result

BinarySearchTree<int> bst;
bst.add(9);
bst.add(2);
bst.add(10);
bst.deleteNode(9);
cout << bst.inOrder();
	2 10

BinarySearchTree<int> bst;
bst.add(9);
bst.add(2);
bst.add(10);
bst.add(8);
cout << bst.inOrder()<<endl;
bst.add(11);
bst.deleteNode(9);
cout << bst.inOrder();
	2 8 9 10
	2 8 10 11

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
Node* addRec(Node*root, T value){
    if(root == NULL) return new Node(value);
    if(root->value >= value)
        root->pLeft =  addRec(root->pLeft, value);
    else if(root->value < value)
        root->pRight = addRec(root->pRight,value);
    return root;
}
void add(T value){
    if(root == NULL){
        root = new Node(value);
    }
    else addRec(root,value);
}

Node* deleteNodeRec(Node*root, T value){
    if(root == NULL) return NULL;
    if(root->value > value)
        root->pLeft =  deleteNodeRec(root->pLeft, value);
    else if(root->value < value)
```

```
    else if(root->value < value)
        root->pRight = deleteNodeRec(root->pRight,value);
    else{
        if(root->pLeft == NULL && root->pRight == NULL){
            delete root;
            return NULL;
        }
        else if(root->pLeft == NULL){
            Node* tmp = root->pRight;
            delete root;
            return tmp;
        }
        else if(root->pRight == NULL){
            Node* tmp = root->pLeft;
            delete root;
            return tmp;
        }
        else{
            Node* tmp = root->pLeft;
            while(tmp->pRight){
                tmp = tmp->pRight;
            }
            swap(root->value, tmp->value);
            root->pLeft = deleteNodeRec(root->pLeft, tmp->value);
        }
    }
    return root;
}
void deleteNode(T value){
    deleteNodeRec(root,value);
}
```

	Test	Expected	Got	

✓	`BinarySearchTree<int> bst;`
	`bst.add(9);`
	`bst.add(2);`
	`bst.add(10);`
	`bst.deleteNode(9);`
	`cout << bst.inOrder();`
		`2 10`	`2 10`	✓

✓	`BinarySearchTree<int> bst;`
	`bst.add(9);`
	`bst.add(2);`
	`bst.add(10);`
	`bst.add(8);`
	`cout << bst.inOrder()<<endl;`
	`bst.add(11);`
	`bst.deleteNode(9);`
	`cout << bst.inOrder();`
		`2 8 9 10`
		`2 8 10 11`
			`2 8 9 10`
			`2 8 10 11`
				✓

Passed all tests! ✓

Câu hỏi **2**

Đúng

Đạt điểm 1,00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

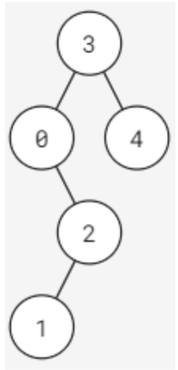
Request: Implement function:

```
vector<int> levelAlterTraverse(BSTNode* root);
```

Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the values of the nodes in each level, alternating from going left-to-right and right-to-left..

Example:

Given a binary search tree in the following:



In the first level, we should traverse from left to right (order: `3`) and in the second level, we traverse from right to left (order: `4, 0`). After traversing all the nodes, the result should be `[3, 4, 0, 2, 1]`.

Note: In this exercise, the libraries `iostream`, `vector`, `stack`, `queue`, `algorithm` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`int arr[] = {0, 3, 5, 1, 2, 4};`
`BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));`
`printVector(levelAlterTraverse(root));`
`BSTNode::deleteTree(root);`
	`[0, 3, 1, 5, 4, 2]`

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
vector<int> levelAlterTraverse(BSTNode* root) {
    if(root == NULL) return {};
    queue<BSTNode*> q;
    vector<vector<int>> vv;
    vector<int> result;
    q.push(root);
    q.push(NULL);
    int i = 0;
    while(q.size()){
        BSTNode* tmp = q.front();
        q.pop();
        if(tmp == NULL){
            vv.push_back(result);
            result.clear();
            i++;
            if(q.size()) q.push(NULL);
            continue;
        }
        result.push_back(tmp->val);
        if(tmp->left) q.push(tmp->left);
```

```
        if(cmp->right) q.push(cmp->right);
    }
    int len = vv.size();
    for(int i = 0; i < len; i++){
        int length = vv[i].size();
        if(i % 2 == 0)
            for(int j = 0; j < length; j++) result.push_back(vv[i][j]);
        else
            for(int j = length - 1; j >= 0; j--) result.push_back(vv[i][j]);
    }
    return result;
}
```

	Test	Expected	Got	
✓	int arr[] = {0, 3, 5, 1, 2, 4}; BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int)); printVector(levelAlterTraverse(root)); BSTNode::deleteTree(root);	[0, 3, 1, 5, 4, 2]	[0, 3, 1, 5, 4, 2]	✓

Passed all tests! ✓

Câu hỏi **3**

Đúng

Đạt điểm 1,00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

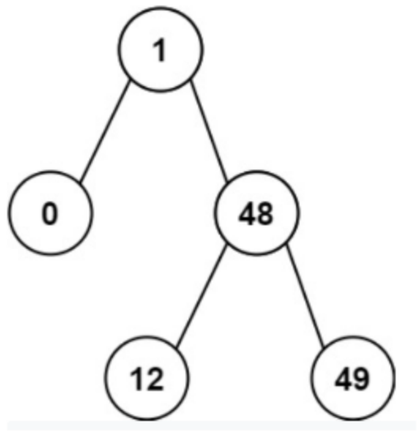
Request: Implement function:

```
int kthSmallest(BSTNode* root, int k);
```

Where `root` is the root node of given binary search tree (this tree has `n` elements) and `k` satisfy: `1 <= k <= n <= 100000`. This function returns the `k`-th smallest value in the tree.

Example:

Given a binary search tree in the following:



With `k = 2`, the result should be `1`.

Note: In this exercise, the libraries `iostream`, `vector`, `stack`, `queue`, `algorithm`, `climits` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`int arr[] = {6, 9, 2, 13, 0, 20};`
`int k = 2;`
`BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));`
`cout << kthSmallest(root, k);`
`BSTNode::deleteTree(root);`
	2

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
void  smallest(BSTNode* root, vector<int>& result){
    if(root == NULL) return;
    smallest(root->left,result);
    result.push_back(root->val);
    smallest(root->right,result);
}
int kthSmallest(BSTNode* root, int k) {
    vector<int> result;
    smallest(root,result);
    return result[k - 1];
}
```

	Test	Expected	Got	

✓	int arr[] = {6, 9, 2, 13, 0, 20};
	int k = 2;
	BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));
	cout << kthSmallest(root, k);
	BSTNode::deleteTree(root);
		2	2	✓

Passed all tests! ✓

Câu hỏi **4**

Đúng

Đạt điểm 1,00

Class **BTNode** is used to store a node in binary search tree, described on the following:

```
class BTNode {
    public:
        int val;
        BTNode *left;
        BTNode *right;
        BTNode() {
            this->left = this->right = NULL;
        }
        BTNode(int val) {
            this->val = val;
            this->left = this->right = NULL;
        }
        BTNode(int val, BTNode*& left, BTNode*& right) {
            this->val = val;
            this->left = left;
            this->right = right;
        }
};
```

Where `val` is the value of node (non-negative integer), `left` and `right` are the pointers to the left node and right node of it, respectively.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

Request: Implement function:

```
int rangeCount(BTNode* root, int lo, int hi);
```

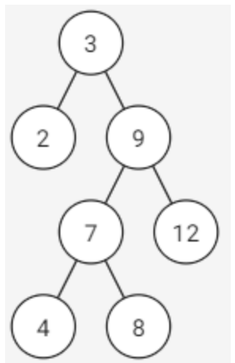
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements), `lo` and `hi` are 2 positives integer and $lo \leq hi$. This function returns the number of all nodes whose values are between `[lo, hi]` in this binary search tree.

More information:

- If a node has `val` which is equal to its ancestor's, it is in the right subtree of its ancestor.

Example:

Given a binary search tree in the following:



With `lo=5, hi=10`, all the nodes satisfied are node `9, 7, 8`; there fore, the result is `3`.

Note: In this exercise, the libraries `iostream`, `stack`, `queue`, `utility` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`int value[] = {3,2,9,7,12,4,8};`
`int lo = 5, hi = 10;`
`BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));`
`cout << rangeCount(root, lo, hi);`
	3

`int value[] = {1167,2381,577,2568,124,1519,234,1679,2696,2359};`
`int lo = 500, hi = 2000;`
`BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));`
`cout << rangeCount(root, lo, hi);`
	4

Answer: (penalty regime: 0 %)

Reset answer

```
int rangeCount(BTNode* root, int lo, int hi) {
    if(root == NULL) return 0;
    else if(root->val >= lo && root->val <= hi)
        return 1 + rangeCount(root->left,lo,hi) + rangeCount(root->right,lo,hi);
    return rangeCount(root->left,lo,hi) + rangeCount(root->right,lo,hi);
}
```

	Test	Expected	Got	

✓	int value[] = {3,2,9,7,12,4,8};
	int lo = 5, hi = 10;
	BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));
	cout << rangeCount(root, lo, hi);
		3	3	✓

	Test	Expected	Got	

✓	`int value[] = {1167,2381,577,2568,124,1519,234,1679,2696,2359};`
	`int lo = 500, hi = 2000;`
	`BTNode* root = BTNode::createBSTree(value, value + sizeof(value)/sizeof(int));`
	`cout << rangeCount(root, lo, hi);`
		4	4	✓

Passed all tests! ✓

Câu hỏi **5**

Đúng

Đạt điểm 1,00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

Request: Implement function:

```
int singleChild(BSTNode* root);
```

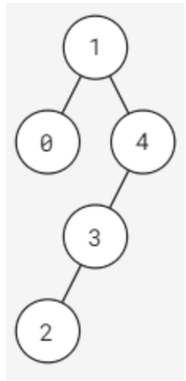
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the number of single children in the tree.

More information:

- A node is called a **single child** if its parent has only one child.

Example:

Given a binary search tree in the following:



There are 2 single children: node 2 and node 3.

Note: In this exercise, the libraries `iostream` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`int arr[] = {0, 3, 5, 1, 2, 4};`
`BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));`
`cout << singleChild(root);`
`BSTNode::deleteTree(root);`
	3

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
int singleChild(BSTNode* root) {
    if(root == NULL) return 0;
    else if(root->left == NULL && root->right != NULL){
        return  1 + singleChild(root->right);
    }
    else if(root->right == NULL && root->left != NULL){
        return  1 + singleChild(root->left);
    }
    return singleChild(root->right) + singleChild(root->left);
}
```

	Test	Expected	Got	
✓	int arr[] = {0, 3, 5, 1, 2, 4}; BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int)); cout << singleChild(root); BSTNode::deleteTree(root);	3	3	✓

Passed all tests! ✓

Câu hỏi **6**

Đúng

Đạt điểm 1,00

Class `BSTNode` is used to store a node in binary search tree, described on the following:

```
class BSTNode {
public:
    int val;
    BSTNode *left;
    BSTNode *right;
    BSTNode() {
        this->left = this->right = nullptr;
    }
    BSTNode(int val) {
        this->val = val;
        this->left = this->right = nullptr;
    }
    BSTNode(int val, BSTNode*& left, BSTNode*& right) {
        this->val = val;
        this->left = left;
        this->right = right;
    }
};
```

Where `val` is the value of node, `left` and `right` are the pointers to the left node and right node of it, respectively. If a repeated value is inserted to the tree, it will be inserted to the left subtree.

Also, a static method named `createBSTree` is used to create the binary search tree, by iterating the argument array left-to-right and repeatedly calling `addNode` method on the root node to insert the value into the correct position. For example:

```
int arr[] = {0, 10, 20, 30};
auto root = BSTNode::createBSTree(arr, arr + 4);
```

is equivalent to

```
auto root = new BSTNode(0);
root->addNode(10);
root->addNode(20);
root->addNode(30);
```

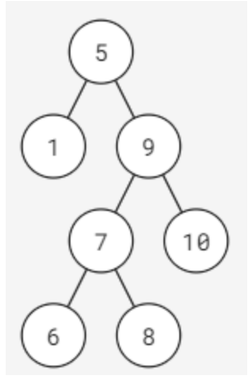
Request: Implement function:

```
BSTNode* subtreeWithRange(BSTNode* root, int lo, int hi);
```

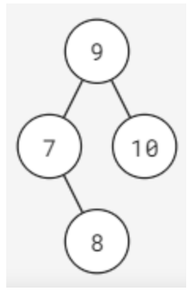
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the binary search tree after deleting all nodes whose values are outside the range `[lo, hi]` (inclusive).

Example:

Given a binary search tree in the following:



With `lo = 7` and `hi = 10`, the result should be:



Note: In this exercise, the libraries `iostream` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result

`int arr[] = {0, 3, 5, 1, 2, 4};`
`int lo = 1, hi = 3;`
`BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));`
`root = subtreeWithRange(root, lo, hi);`
`BSTNode::printPreorder(root);`
`BSTNode::deleteTree(root);`
	`3 1 2`

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
BSTNode* find(BSTNode* root, int lo, int hi){
    if(root == NULL) return NULL;
    BSTNode* tmp = root;
    while(tmp){
        if(tmp->val >= lo && tmp->val <= hi) break;
        else if(tmp->val > hi) tmp = tmp->left;
        else tmp = tmp->right;
    }
    return tmp;
}
```

```
}

BSTNode* del(BSTNode* root, int lo, int hi){
    if(root == NULL) return NULL;
    else if(root->val < lo) return del(root->right,lo,hi);
    else if(root->val > hi) return del(root->left,lo,hi);
    root->left = del(root->left,lo,hi);
    root->right= del(root->right,lo,hi);
    return root;
}


BSTNode* subtreeWithRange(BSTNode* root, int lo, int hi) {
     root = find(root, lo, hi);
     root = del(root,lo,hi);
    return root;
}
```

	Test	Expected	Got	

✓	int arr[] = {0, 3, 5, 1, 2, 4};
	int lo = 1, hi = 3;
	BSTNode* root = BSTNode::createBSTree(arr, arr + sizeof(arr)/sizeof(int));
	root = subtreeWithRange(root, lo, hi);
	BSTNode::printPreorder(root);
	BSTNode::deleteTree(root);
		3 1 2	3 1 2	✓

Passed all tests! ✓

Câu hỏi **7**

Đúng

Đạt điểm 1,00

Given class **BinarySearchTree**, you need to finish method **find(i)** to check whether value i is in the tree or not; method **sum(l,r)** to calculate sum of all all elements v in the tree that has value greater than or equal to l and less than or equal to r.

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

template<class T>
class BinarySearchTree
{
public:
    class Node;

private:
    Node* root;

public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }

    class Node
    {
    private:
        T value;
        Node* pLeft, * pRight;
        friend class BinarySearchTree<T>;

    public:
        Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
    Node* addRec(Node* root, T value);
    void add(T value) ;
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END
};
```

For example:

Test	Result

BinarySearchTree<int> bst;
for (int i = 0; i < 10; ++i) {
bst.add(i);
}
cout << bst.find(7) << endl;
cout << bst.sum(0, 4) << endl
	1
	10

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
bool find(T i) {
    // TODO: return true if value i is in the tree; otherwise, return false.
    Node* tmp = root;
    while(tmp){
        if(tmp->value > i) tmp = tmp->pLeft;
        else if(tmp->value < i) tmp = tmp->pRight;
        else return true;
    }
    return false;

}

T rangeCount(Node* root, T lo, T hi) {
    if(root == NULL) return 0;
    else if(root->value >= lo && root->value <= hi)
        return root->value + rangeCount(root->pLeft,lo,hi) + rangeCount(root->pRight,l
    return rangeCount(root->pLeft,lo,hi) + rangeCount(root->pRight,lo,hi);
}

T sum(T l, T r) {
    return rangeCount(root,l,r);

}
```

	Test	Expected	Got	

✓	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(i);
	}
	cout << bst.find(7) << endl;
	cout << bst.sum(0, 4) << endl
		1
		10
			1
			10
				✓

	Test	Expected	Got	

✓	`int values[] = { 66,60,84,67,21,45,62,1,80,35 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(5) << endl;`
	`cout << bst.sum(10, 40);`
		0
		56
			0
			56
				✓

✓	`int values[] = { 38,0,98,38,99,67,19,70,55,6 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(5) << endl;`
	`cout << bst.sum(10, 40);`
		0
		95
			0
			95
				✓

✓	`int values[] = { 34,81,73,48,66,91,19,84,78,79 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(5) << endl;`
	`cout << bst.sum(10, 40);`
		0
		53
			0
			53
				✓

✓	`int values[] = { 94,61,75,36,34,58,62,74,54,90 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 10; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		1
		70
			1
			70
				✓

✓	`int values[] = { 32,0,2,84,34,78,70,60,95,71,26,62,0,22,95 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		1
		114
			1
			114
				✓

	Test	Expected	Got	

✓	`int values[] = { 53,24,32,40,80,47,81,88,42,29,31,91,77,73,90 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		156
			0
			156
				✓

✓	`int values[] = { 32,19,23,33,76,1,37,53,18,89,28,1,77,52,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		207
			0
			207
				✓

✓	`int values[] = { 25,29,57,30,62,56,60,55,88,56,70,83,56,75,17 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		101
			0
			101
				✓

✓	`int values[] = { 75,13,83,83,30,40,10,86,17,21,45,22,22,72,63 };`
	`BinarySearchTree<int> bst;`
	`for (int i = 0; i < 15; ++i) {`
	`  bst.add(values[i]);`
	`}`
	
	`cout << bst.find(34) << endl;`
	`cout << bst.sum(10, 40);`
		0
		175
			0
			175
				✓

Passed all tests! ✓

Câu hỏi **8**

Đúng

Đạt điểm 1,00

Given class **BinarySearchTree**, you need to finish method getMin() and getMax() in this question.

```
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

template<class T>
class BinarySearchTree
{
public:
    class Node;

private:
    Node* root;

public:
    BinarySearchTree() : root(nullptr) {}
    ~BinarySearchTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }
    class Node
    {
    private:
        T value;
        Node* pLeft, * pRight;
        friend class BinarySearchTree<T>;

    public:
        Node(T value) : value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
    Node* addRec(Node* root, T value);
    void add(T value) ;
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END
};
```

For example:

Test	Result

BinarySearchTree<int> bst;
for (int i = 0; i < 10; ++i) {
  bst.add(i);
}
cout << bst.getMin() << endl;
cout << bst.getMax() << endl;
	0
	9

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
T getMin() {
    Node* tmp = root;
    while(tmp->pLeft){
        tmp = tmp->pLeft;
    }
    return tmp->value;

}

T getMax() {
     Node* tmp = root;
    while(tmp->pRight){
        tmp = tmp->pRight;
    }
    return tmp->value;   //TODO: return the maximum values of nodes in the tree.

}
```

	Test	Expected	Got	

✓	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	  bst.add(i);
	}
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		0
		9
			0
			9
				✓

	Test	Expected	Got	

✓	int values[] = { 66,60,84,67,21,45,62,1,80,35 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		1
		84
			1
			84
				✓

✓	int values[] = { 38,0,98,38,99,67,19,70,55,6 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		0
		99
			0
			99
				✓

✓	int values[] = { 34,81,73,48,66,91,19,84,78,79 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		19
		91
			19
			91
				✓

✓	int values[] = { 94,61,75,36,34,58,62,74,54,90 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 10; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		34
		94
			34
			94
				✓

✓	int values[] = { 32,0,2,84,34,78,70,60,95,71,26,62,0,22,95 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 15; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		0
		95
			0
			95
				✓

	Test	Expected	Got	

✓	int values[] = { 53,24,32,40,80,47,81,88,42,29,31,91,77,73,90 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 15; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		24
		91
			24
			91
				✓

✓	int values[] = { 32,19,23,33,76,1,37,53,18,89,28,1,77,52,17 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 15; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		1
		89
			1
			89
				✓

✓	int values[] = { 25,29,57,30,62,56,60,55,88,56,70,83,56,75,17 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 15; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		17
		88
			17
			88
				✓

✓	int values[] = { 75,13,83,83,30,40,10,86,17,21,45,22,22,72,63 };
	BinarySearchTree<int> bst;
	for (int i = 0; i < 15; ++i) {
	bst.add(values[i]);
	}
	
	cout << bst.getMin() << endl;
	cout << bst.getMax() << endl;
		10
		86
			10
			86
				✓

Passed all tests! ✓