Câu hỏi 1

Chính xác

Chấm điểm của 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

```
// Function to add a new value to the Binary Search Tree
void add(T value)
{
    // Call the recursive helper function to add the value
    this->root = addRec(this->root, value);
}

// Recursive helper function for 'add'
Node *addRec(Node *node, T value)
{
    // If the tree/sub-tree is empty, create a new node
    if (node == nullptr)
    {
        return new Node(value);
    }
    // If the value to be added is less than or equal to the current node's value,
    // call the function for the left sub-tree
    if (value <= node->value)
    {
        node->pLeft = addRec(node->pLeft, value);
    }
    else
    {
        // Else call the function for the right sub-tree
        node->pRight = addRec(node->pRight, value);
    }
    return node;
}

// Function to delete a node from the Binary Search Tree
void deleteNode(T value)
{
    // Call the recursive helper function to delete the value
    this->root = deleteNodeRec(this->root, value);
}

// Recursive helper function for 'deleteNode'
Node *deleteNodeRec(Node *node, T value)
{
    // Base case: If the tree is empty, return it
    if (node == nullptr)
    {
        return node;
    }
```

```
45      // If the value to be deleted is less than the node's value,
46      // then it lies in left sub-tree
47      if (value < node->value)
48      {
49          node->pLeft = deleteNodeRec(node->pLeft, value);
50      }
51      else if (value > node->value)
52      {
53          // Else if the value to be deleted is greater than the node's value,
54          // then it lies in right sub-tree
55          node->pRight = deleteNodeRec(node->pRight, value);
56      }
57      else
58      {
59          // If key is same as root's key, then this is the node to be deleted
60          if (node->pLeft == nullptr)
61          {
62              Node *temp = node->pRight;
```

Precheck Kiểm tra

	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! ✔

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn

Câu hỏi 2

Chính xác

Chấm điểm của 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);
```

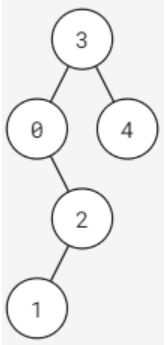
Request: Implement function:

```
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

```
1  // Function to traverse a binary search tree in an alternating level order
2  vector<int> levelAlterTraverse(BSTNode *root)
3  {
4      // Initialize an empty vector to store the result
5      vector<int> result;
6
7      // If the tree is empty, return the empty result
8      if (root == nullptr)
9      {
10         return result;
11     }
12
13     // Initialize two deques to store the nodes at the current and next levels
14     deque<BSTNode *> currentLevel;
15     deque<BSTNode *> nextLevel;
16
17     // Start with the root node at the current level
18     currentLevel.push_back(root);
19
20     // Initialize a boolean flag to indicate the direction of traversal
21     bool leftToRight = true;
22
```

Precheck Kiểm tra

	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! ✔

HCMUT

MyBK

BKSI

LIÊN HỆ

268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn

Câu hỏi 3

Chính xác

Chấm điểm của 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);
```

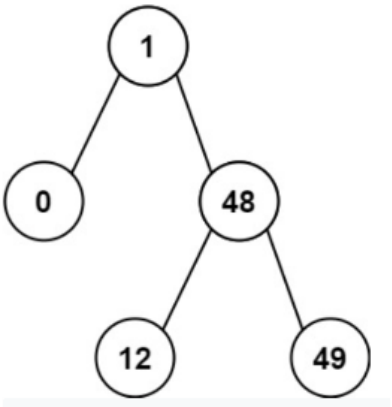
Request: Implement function:

```
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

```
1  // Function to find the k-th smallest element in a binary search tree
2  int kthSmallest(BSTNode *root, int k)
3  {
4      // Create a stack to keep track of nodes
5      std::stack<BSTNode *> stk;
6      // Start with the root node
7      BSTNode *curr = root;
8      // Initialize a counter to keep track of the number of nodes visited
9      int count = 0;
10
11     // While there are still nodes to visit
12     while (curr != nullptr || !stk.empty())
13     {
14         // Visit the left subtree first
15         while (curr != nullptr)
16         {
17             stk.push(curr);
18             curr = curr->left;
19         }
20         // Visit the current node
21         curr = stk.top();
22         stk.pop();
```

Precheck Kiểm tra

	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! ✔

WEBSITE

HCMUT

MyBK

BKSI

LIÊN HỆ

268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn

Câu hỏi 4

Chính xác

Chấm điểm của 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);
```

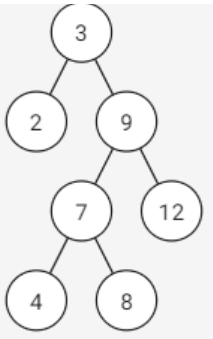
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements), `lo` and `hi` are 2 positives integer and `lo ≤ hi`. This function returns the number of all nodes whose values are between `[lo, hi]` in this binary search tree.

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

```
// Function to count the number of nodes in a binary search tree whose values are within a given range
int rangeCount(BTNode *root, int lo, int hi)
{
    // If the node is null, return 0 because a null node doesn't have a value
    if (root == NULL)
    {
        return 0;
    }
    // If the node's value is less than the lower bound of the range,
    // skip this node and all nodes in its left subtree because they are also less than the lower bour
    // and recursively count nodes in its right subtree
    if (root->val < lo)
    {
        return rangeCount(root->right, lo, hi);
    }
    // If the node's value is greater than the upper bound of the range,
    // skip this node and all nodes in its right subtree because they are also greater than the upper
    // and recursively count nodes in its left subtree
    if (root->val > hi)
    {
        return rangeCount(root->left, lo, hi);
    }
    // If the node's value is within the range,
    // count this node and recursively count nodes in both its left and right subtrees
    return 1 + rangeCount(root->left, lo, hi) + rangeCount(root->right, lo, hi);
}

```

Precheck Kiểm tra

	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	✔

✔	`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

Chính xác

Chấm điểm của 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);
```

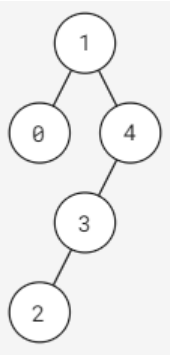
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the number of single children in the tree.

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

```
1  // Function to count the number of single children in a binary search tree
2  int singleChild(BSTNode *root)
3  {
4      // If the node is null, return 0
5      if (root == nullptr)
6      {
7          return 0;
8      }
9
10     // Initialize count to 0
11     int count = 0;
12
13     // If the node has either a left child or a right child (but not both), increment count
14     if ((root->left != nullptr && root->right == nullptr) ||
15         (root->left == nullptr && root->right != nullptr))
16     {
17         count = 1;
18     }
19
20     // Recursively call the function on the left and right children of the node,
21     // adding their results to count
22     return count + singleChild(root->left) + singleChild(root->right);
```

Precheck Kiểm tra

	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! ✔

HCMUT

MyBK

BKSI

LIÊN HỆ

268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn

Câu hỏi 6

Chính xác

Chấm điểm của 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);
```

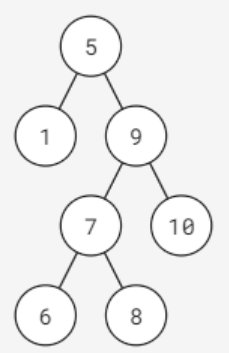
Request: Implement function:

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

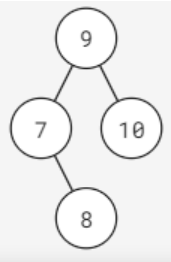
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the binary search tree after deleting all nodes whose values are outside the range `[lo, hi]` (inclusive).

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

```
1  BSTNode *subtreeWithRange(BSTNode *root, int lo, int hi)
2  {
3      // Base case: if the node is null, return null
4      if (root == nullptr)
5      {
6          return nullptr;
7      }
8
9      // If the node's value is less than lo, then the left subtree cannot have any nodes in range,
10     // so we discard the left subtree and check in the right subtree
11     if (root->val < lo)
12     {
13         BSTNode *rChild = subtreeWithRange(root->right, lo, hi);
14         delete root;
15         return rChild;
16     }
17
18     // If the node's value is more than hi, then the right subtree cannot have any nodes in range,
19     // so we discard the right subtree and check in the left subtree
20     if (root->val > hi)
21     {
22
```

Precheck Kiểm tra

	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! ✔

BÁCH KHOA E-LEARNING

WEBSITE

HCMUT

MyBK

BKSI

LIÊN HỆ

268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn

Câu hỏi 7

Chính xác

Chấm điểm của 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

```
1 // STUDENT ANSWER BEGIN
2 // You can define other functions here to help you.
```

```
bool findRec(Node *node, T i)
{
    if (node == NULL)
        return false;
    if (node->value == i)
        return true;
    if (node->value < i)
        return findRec(node->pRight, i);
    return findRec(node->pLeft, i);
}
T sumRec(Node *node, T l, T r)
{
    if (node == NULL)
        return 0;
    if (node->value < l)
        return sumRec(node->pRight, l, r);
    if (node->value > r)
        return sumRec(node->pLeft, l, r);
    return node->value + sumRec(node->pLeft, l, r) + sumRec(node->pRight, l, r);
}
```

Precheck Kiểm tra

	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
				✔

✔	`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
				✔

	Test	Expected	Got	

✔	`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
				✔

✔	`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! ✔

LIÊN HỆ

268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn

Câu hỏi 8

Chính xác

Chấm điểm của 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

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.

T getMin()
{
    Node *node = root;
    while (node->pLeft != NULL)
    {
        node = node->pLeft;
    }
    return node->value;
}

T getMax()
{
    Node *node = root;
    while (node->pRight != NULL)
    {
        node = node->pRight;
    }
    return node->value;
}
```

Precheck Kiểm tra

	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
				✔

✔	`  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
				✔

	Test	Expected	Got	

✔	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
				✔

✔	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! ✔

WEBSITE

HCMUT

MyBK

BKSI

LIÊN HỆ

268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

(028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

elearning@hcmut.edu.vn