Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Sáu, 22 tháng 11 2024, 9:34 PM
Kết thúc lúc	Thứ Sáu, 22 tháng 11 2024, 9:36 PM
Thời gian thực hiện	2 phút 20 giây
Điểm	8,00/8,00
Điểm	**10,00** trên 10,00 (**100**%)

Câu hỏi **1**

Đúng

Đạt điểm 1,00 trên 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`

Test	Result

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 node to the BST
    void add(T value) {
        root = addRec(root, value);
    }

    Node* addRec(Node* node, T value) {
        if (node == nullptr) {
            return new Node(value);
        }
        // Add the value in the left subtree if it's the same as the parent node's va
        if (value <= node->value) {
            node->pLeft = addRec(node->pLeft, value);
        } else {
            node->pRight = addRec(node->pRight, value);
        }
        return node;
    }

    // Function to delete a node from the BST
    void deleteNode(T value) {
        root = deleteRec(root, value);
    }

    Node* deleteRec(Node* node, T value) {
        if (node == nullptr) {
            return node;
        }
        // Navigate the tree to find the node to delete
        if (value < node->value) {
            node->pLeft = deleteRec(node->pLeft, value);
        } else if (value > node->value) {
            node->pRight = deleteRec(node->pRight, value);
        } else {
            // Node with only one child or no child
            if (node->pLeft == nullptr) {
                Node* temp = node->pRight;
                delete node;
                return temp;
            } else if (node->pRight == nullptr) {
                Node* temp = node->pLeft;
                delete node;
                return temp;
            }

            // Node with two children: get the inorder successor (smallest in the rig
            Node* temp = minValueNode(node->pRight);
            node->value = temp->value;
            node->pRight = deleteRec(node->pRight, temp->value);
        }
        return node;
    }

    // Helper function to find the minimum value node in a subtree
    Node* minValueNode(Node* node) {
        Node* current = node;
        while (current && current->pLeft != nullptr) {
            current = current->pLeft;
        }
        return current;
    }
```

	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! ✓

Đúng

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

Câu hỏi **2**

Đúng

Đạt điểm 1,00 trên 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() {
    if (!this->root) {
        throw runtime_error("Tree is empty, no minimum value.");
    }

    Node* current = this->root;
    while (current->pLeft) { // Traverse to the leftmost node
        current = current->pLeft;
    }
    return current->value; // The leftmost node contains the minimum value
}

T getMax() {
    if (!this->root) {
        throw runtime_error("Tree is empty, no maximum value.");
    }

    Node* current = this->root;
    while (current->pRight) { // Traverse to the rightmost node
        current = current->pRight;
    }
    return current->value; // The rightmost node contains the maximum value
}
```

	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
				✓

<table>
<tr><th></th><th>Test</th><th>Expected</th><th>Got</th><th></th></tr>
<tr><td>✓</td><td><pre>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;</pre></td><td><pre>34
94</pre></td><td><pre>34
94</pre></td><td>✓</td></tr>
<tr><td>✓</td><td><pre>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;</pre></td><td><pre>0
95</pre></td><td><pre>0
95</pre></td><td>✓</td></tr>
<tr><td>✓</td><td><pre>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;</pre></td><td><pre>24
91</pre></td><td><pre>24
91</pre></td><td>✓</td></tr>
<tr><td>✓</td><td><pre>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;</pre></td><td><pre>1
89</pre></td><td><pre>1
89</pre></td><td>✓</td></tr>
<tr><td>✓</td><td><pre>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;</pre></td><td><pre>17
88</pre></td><td><pre>17
88</pre></td><td>✓</td></tr>
<tr><td>✓</td><td><pre>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;</pre></td><td><pre>10
86</pre></td><td><pre>10
86</pre></td><td>✓</td></tr>
</table>

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 **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) {
    Node* current = this->root;
    Node* current = this->root;
    while (current) {
        if (i == current->value) return true;  // Found the value
        if (i < current->value) {
            current = current->pLeft;  // Move to left subtree
        } else {
            current = current->pRight;  // Move to right subtree
        }
    }
    return false;  // Value not found
}

T sumInRange(Node* node, T l, T r) {
    if (!node) return 0;  // Base case: null node contributes 0 to the sum

    T total = 0;

    // If node value is within range, include it in the sum
    if (node->value >= l && node->value <= r) {
        total += node->value;
    }

    // Traverse left subtree if there's a chance of values in range
    if (node->value > l) {
        total += sumInRange(node->pLeft, l, r);
    }

    // Traverse right subtree if there's a chance of values in range
    if (node->value < r) {
        total += sumInRange(node->pRight, l, r);
    }

    return total;
}

T sum(T l, T r) {
    return sumInRange(this->root, l, r);  // Start from the root
}
```

	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
				✓

	Test	Expected	Got	

✓	`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
				✓

✓	`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! ✓

Đúng

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

Câu hỏi **4**

Đúng

Đạt điểm 1,00 trên 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);
```

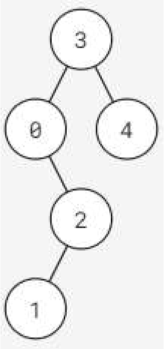
Request: Implement function:

`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) return {}; // If tree is empty, return an empty vector

    vector<int> result;
    deque<BSTNode*> dq;
    dq.push_back(root);

    bool leftToRight = true; // Start with left-to-right traversal

    while (!dq.empty()) {
        int levelSize = dq.size();
        vector<int> levelValues(levelSize);

        for (int i = 0; i < levelSize; ++i) {
            BSTNode* node = nullptr;

            // Pop nodes based on current direction
            if (leftToRight) {
                node = dq.front();
                dq.pop_front();
            } else {
                node = dq.back();
                dq.pop_back();
            }

            // Add the node's value to the level vector
            levelValues[i] = node->val;

            // Enqueue child nodes based on the current direction
            if (leftToRight) {
                if (node->left) dq.push_back(node->left);
                if (node->right) dq.push_back(node->right);
            } else {
                if (node->right) dq.push_front(node->right);
                if (node->left) dq.push_front(node->left);
            }
        }

        // Append the level's values to the result
        result.insert(result.end(), levelValues.begin(), levelValues.end());

        // Switch direction for the next level
        leftToRight = !leftToRight;
    }

    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! ✓

Đúng

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

Câu hỏi **5**

Đúng

Đạt điểm 1,00 trên 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);`

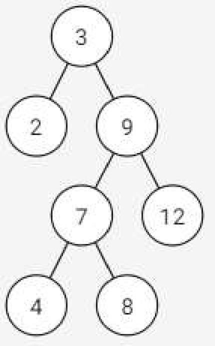
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements), `lo` and `hi` are 2 positives integer and `lo ≤ hi`. This function returns the number of all nodes whose values are between `[lo, hi]` in this binary search tree.

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

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

✓	`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! ✓

Đúng

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

Câu hỏi **6**

Đúng

Đạt điểm 1,00 trên 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);`

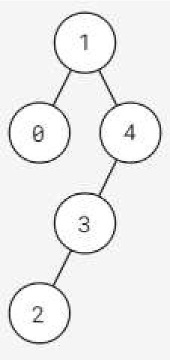
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the number of single children in the tree.

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 singleChildHelper(BSTNode* node, BSTNode* parent) {
    if (!node) return 0;
    int count = 0;
    if (parent && ((parent->left && !parent->right) || (!parent->left && parent->righ
        count++;
    }
    count += singleChildHelper(node->left, node);
    count += singleChildHelper(node->right, node);
    return count;
}

int singleChild(BSTNode* root) {
    return singleChildHelper(root, nullptr);
}
```

	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! ✓

Đúng

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

Câu hỏi **7**

Đúng

Đạt điểm 1,00 trên 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);
```

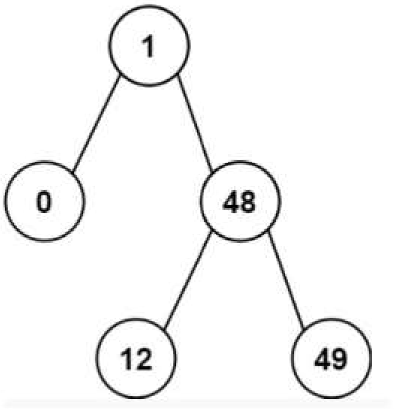
Request: Implement function:

`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

```
int kthSmallestHelper(BSTNode* root, int& k) {
    if (!root) return -1;
    // Traverse left subtree
    int val = kthSmallestHelper(root->left, k);
    if (k == 0) return val;
    // Visit current node
    k--;
    if (k == 0) return root->val;
    // Traverse right subtree
    return kthSmallestHelper(root->right, k);
}

int kthSmallest(BSTNode* root, int k) {
    return kthSmallestHelper(root, k);
}
```

	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! ✓

Đúng

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

Câu hỏi **8**

Đúng

Đạt điểm 1,00 trên 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);
```

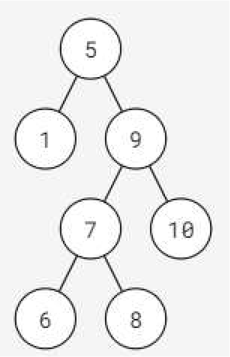
Request: Implement function:

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

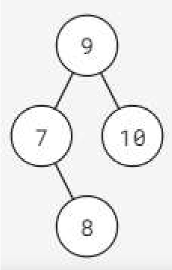
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the binary search tree after deleting all nodes whose values are outside the range `[lo, hi]` (inclusive).

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* subtreeWithRange(BSTNode* root, int lo, int hi) {
    if (!root) return nullptr;
    if (root->val < lo) {
        return subtreeWithRange(root->right, lo, hi);
    }
    if (root->val > hi) {
        return subtreeWithRange(root->left, lo, hi);
    }
    root->left = subtreeWithRange(root->left, lo, hi);
    root->right = subtreeWithRange(root->right, 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! ✓

Đúng

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