Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Hai, 14 tháng 4 2025, 11:38 PM
Kết thúc lúc	Thứ Hai, 14 tháng 4 2025, 11:40 PM
Thời gian thực hiện	2 phút 6 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

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

```
// Hàm đệ quy để thêm một giá trị vào cây
    Node* addRec(Node* node, T value) {
        // Nếu cây rỗng hoặc đã đến vị trí cần thêm
        if (node == nullptr) {
            return new Node(value); // Tạo một nút mới và trả về
        }

        // Nếu giá trị nhỏ hơn hoặc bằng giá trị của nút hiện tại
        if (value <= node->value) {
            // Thêm vào cây con bên trái
            // Lưu ý: Khi thêm nút có giá trị bằng với nút hiện tại, thêm vào bên trái theo yêu cầu
            node->pLeft = addRec(node->pLeft, value);
        } else {
            // Thêm vào cây con bên phải
            node->pRight = addRec(node->pRight, value);
        }

        // Trả về nút hiện tại sau khi đã thêm giá trị mới
        return node;
    }
    // Hàm thêm một giá trị vào cây
    void add(T value) {
        root = addRec(root, value);
    }

    // Tìm nút có giá trị nhỏ nhất trong cây con
    Node* findMin(Node* node) {
        Node* current = node;
        // Đi theo con trỏ bên trái cho đến khi không thể đi tiếp
        while (current && current->pLeft != nullptr) {
            current = current->pLeft;
        }
        return current; // Trả về nút có giá trị nhỏ nhất
    }

    Node* deleteNodeRec(Node* node, T value) {
        // Nếu cây rỗng
        if (node == nullptr) {
            return nullptr;
        }

        // Tìm nút cần xóa
        if (value < node->value) {
            // Nếu giá trị cần xóa nhỏ hơn giá trị nút hiện tại, tìm trong cây con bên trái
            node->pLeft = deleteNodeRec(node->pLeft, value);
        } else if (value > node->value) {
            // Nếu giá trị cần xóa lớn hơn giá trị nút hiện tại, tìm trong cây con bên phải
            node->pRight = deleteNodeRec(node->pRight, value);
        } else {
            // Đã tìm thấy nút cần xóa

            // Trường hợp 1: Nút không có con (nút lá)
            if (node->pLeft == nullptr && node->pRight == nullptr) {
                delete node; // Xóa nút
```

```
            return nullptr; // Trả về nullptr để cập nhật con trỏ từ cha
        }
        // Trường hợp 2: Nút chỉ có một con bên phải
        else if (node->pLeft == nullptr) {
            Node* temp = node->pRight; // Lưu lại con trỏ đến con bên phải
            delete node; // Xóa nút
            return temp; // Trả về con trỏ đến con bên phải để cập nhật từ cha
        }
```

	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

```
/**
     * Phương thức tìm giá trị nhỏ nhất trong cây nhị phân tìm kiếm
     * Trong một cây nhị phân tìm kiếm, giá trị nhỏ nhất sẽ nằm ở nút ngoài cùng bên trái
     * @return giá trị nhỏ nhất trong cây
```

```
     * @throws runtime_error nếu cây rỗng
*/
    T getMin() {
        // Kiểm tra nếu cây rỗng
        if (root == nullptr) {
            throw runtime_error("Tree is empty"); // Ném ngoại lệ nếu cây rỗng
        }
        // Bắt đầu từ nút gốc
        Node* current = root;
        // Đi xuống bên trái cho đến khi không thể đi tiếp
        // Nút nhỏ nhất sẽ là nút ngoài cùng bên trái (Cây BST)
        while (current->pLeft != nullptr) {
            current = current->pLeft;
        }
        // Trả về giá trị của nút nhỏ nhất
        return current->value;
    }

/**
     * Phương thức tìm giá trị lớn nhất trong cây nhị phân tìm kiếm
     * Trong một cây nhị phân tìm kiếm, giá trị lớn nhất sẽ nằm ở nút ngoài cùng bên phải
     * @return giá trị lớn nhất trong cây
     * @throws runtime_error nếu cây rỗng
 */
    T getMax() {
        // Kiểm tra nếu cây rỗng
        if (root == nullptr) {
            throw runtime_error("Tree is empty"); // Ném ngoại lệ nếu cây rỗng
        }
        // Bắt đầu từ nút gốc
        Node* current = root;
        // Đi xuống bên phải cho đến khi không thể đi tiếp
        // Nút lớn nhất sẽ là nút ngoài cùng bên phải (Cây BST)
        while (current->pRight != nullptr) {
            current = current->pRight;
        }
        // Trả về giá trị của nút lớn nhất
        return current->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
				✓

	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.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
				✓

✓	`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! ✓

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

```
/**
     * Hàm đệ quy hỗ trợ cho phương thức find()
     * @param node: Node hiện tại đang xét
     * @param value: Giá trị cần tìm kiếm
     * @return true nếu tìm thấy giá trị, false nếu không tìm thấy
*/
    bool findRec(Node* node, T value) {
        // Trường hợp cơ sở: đã đến node NULL, không tìm thấy giá trị
        if (node == nullptr) {
```

```
            return false;
        }

        // Đã tìm thấy giá trị
        if (node->value == value) {
            return true;
        }

        // Tìm kiếm đệ quy trong cây con trái hoặc phải
        if (value < node->value) {
            // Nếu giá trị cần tìm nhỏ hơn giá trị node hiện tại, tìm trong cây con trái
            return findRec(node->pLeft, value);
        } else {
            // Nếu giá trị cần tìm lớn hơn giá trị node hiện tại, tìm trong cây con phải
            return findRec(node->pRight, value);
        }
    }
/**
     * Kiểm tra xem một giá trị có tồn tại trong cây hay không
     * @param i: Giá trị cần kiểm tra
     * @return true nếu giá trị tồn tại trong cây, false nếu không tồn tại
*/
    bool find(T i) {
        return findRec(root, i);
    }

/**
     * Hàm đệ quy hỗ trợ cho phương thức sum()
     * @param node: Node hiện tại đang xét
     * @param l: Giới hạn dưới của khoảng tính tổng
     * @param r: Giới hạn trên của khoảng tính tổng
     * @return Tổng các giá trị trong khoảng [l, r]
*/
    T sumRec(Node* node, T l, T r) {
        // Trường hợp cơ sở: đã đến node NULL
        if (node == nullptr) {
            return 0;
        }

        T sum = 0;

        // Nếu giá trị của node hiện tại nằm trong khoảng [l, r], cộng vào tổng
        if (node->value >= l && node->value <= 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
				✓

✓	`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);
```

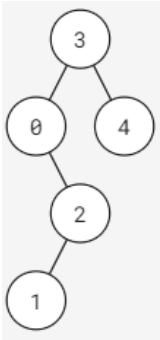
Request: Implement function:

```
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

```
/** * Hàm duyệt cây nhị phân tìm kiếm theo từng tầng, với hướng duyệt xen kẽ
    * - Các tầng chẵn (0, 2, 4, ...) duyệt từ trái sang phải
     * - Các tầng lẻ (1, 3, 5, ...) duyệt từ phải sang trái
     * @param root Nút gốc của cây nhị phân tìm kiếm
     * @return Vector chứa giá trị các phần tử theo thứ tự duyệt
*/
vector<int> levelAlterTraverse(BSTNode* root) {
    // Vector để lưu kết quả duyệt
    vector<int> result;

    // Kiểm tra trường hợp cây rỗng
    if (root == nullptr) {
        return result; // Trả về vector rỗng
    }

    // Sử dụng hàng đợi để duyệt cây theo tầng
    queue<BSTNode*> q;
    q.push(root); // Bắt đầu với nút gốc

    // Biến theo dõi tầng hiện tại (bắt đầu từ 0)
    int level = 0;

    // Duyệt từng tầng của cây
    while (!q.empty()) {
        // Lấy số lượng nút ở tầng hiện tại
        int levelSize = q.size();

        // Vector tạm thời để lưu các nút ở tầng hiện tại
        vector<int> levelNodes;
        // Xử lý tất cả các nút ở tầng hiện tại
        for (int i = 0; i < levelSize; i++) {
            // Lấy nút đầu tiên từ hàng đợi
            BSTNode* current = q.front();
            q.pop();
            // Thêm giá trị của nút hiện tại vào vector tạm thời
            levelNodes.push_back(current->val);
            // Thêm các con của nút hiện tại vào hàng đợi để xử lý ở tầng tiếp theo
            if (current->left != nullptr) {
                q.push(current->left);
            }
            if (current->right != nullptr) {
                q.push(current->right);
            }
        }
        // Thêm các nút vào kết quả theo hướng duyệt tương ứng với tầng
        if (level % 2 == 0) {
            // Tầng chẵn (0, 2, 4, ...) - duyệt từ trái sang phải
            result.insert(result.end(), levelNodes.begin(), levelNodes.end());
        } else {
            // Tầng lẻ (1, 3, 5, ...) - duyệt từ phải sang trái
            result.insert(result.end(), levelNodes.rbegin(), levelNodes.rend());
        }
```

	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);
```

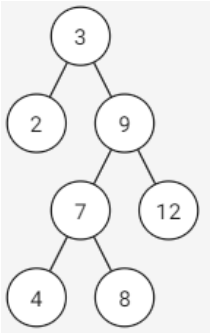
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements), `lo` and `hi` are 2 positives integer and $lo \leq hi$. This function returns the number of all nodes whose values are between `[lo, hi]` in this binary search tree.

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

```
// Đếm số nút có giá trị nằm trong khoảng [lo, hi] trong BST
int rangeCount(BTNode* root, int lo, int hi) {
    // Trường hợp cơ sở: cây rỗng
    if (root == NULL) {
        return 0;
    }

    int count = 0;

    // Kiểm tra nếu nút hiện tại nằm trong khoảng
    if (root->val >= lo && root->val <= hi) {
        count = 1;
    }

    // Nếu giá trị hiện tại lớn hơn hoặc bằng lo, duyệt cây con trái
    if (root->val >= lo) {
        count += rangeCount(root->left, lo, hi);
    }
    // Nếu giá trị hiện tại nhỏ hơn hoặc bằng hi, duyệt cây con phải
    if (root->val <= hi) {
        count += rangeCount(root->right, lo, hi);
    }

    return count;
}
```

	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);
```

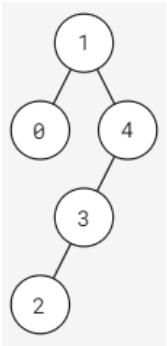
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the number of single children in the tree.

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

```
/**
    * Hàm đếm số nút"có một nút con" trong BST
    * Nút đó có nút con bên trái nhưng không có nút con bên phải, HOẶC
    * Nút đó có nút con bên phải nhưng không có nút con bên trái
*/
int singleChild(BSTNode* root) {
    // Trường hợp cơ sở: nếu cây rỗng (root là nullptr), không có nút nào
    if (root == nullptr) {
        return 0;
    }
    int count = 0;

    // Kiểm tra nếu nút hiện tại có đúng một nút con
    // Trường hợp 1: Có con trái nhưng không có con phải
    if (root->left != nullptr && root->right == nullptr) {
        count++;  // Nút con trái là "single child"
    }
    // Trường hợp 2: Có con phải nhưng không có con trái
    if (root->left == nullptr && root->right != nullptr) {
        count++;  // Nút con phải là "single child"
    }

    // Đệ quy đếm số "single child" trong các cây con
    if (root->left != nullptr) {
        count += singleChild(root->left);  // Đệ quy cho cây con trái
    }
    if (root->right != nullptr) {
        count += singleChild(root->right);  // Đệ quy cho cây con phải
    }
    return count;
}
```

	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);
```

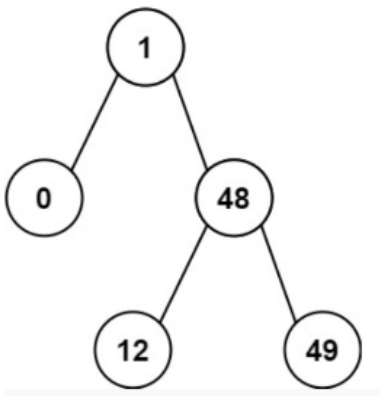
Request: Implement function:

```
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

```
// Tìm phần tử nhỏ thứ k trong cây nhị phân tìm kiếm (BST)
int kthSmallest(BSTNode* root, int k) {
    // Sử dụng stack để duyệt inorder
    stack<BSTNode*> nodeStack;
    BSTNode* current = root;
    int count = 0;

    // Duyệt inorder không đệ quy
    while (current != nullptr || !nodeStack.empty()) {
        // Đi xuống hết bên trái
        while (current != nullptr) {
            nodeStack.push(current);
            current = current->left;
        }
        // Lấy phần tử nhỏ nhất tiếp theo
        current = nodeStack.top();
        nodeStack.pop();

        // Tăng biến đếm
        count++;
        if (count == k) {
            return current->val;
        }

        // Chuyển sang duyệt cây con phải
        current = current->right;
    }

    return -1; // Trường hợp k > số lượng node trong cây
}
```

	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);
```

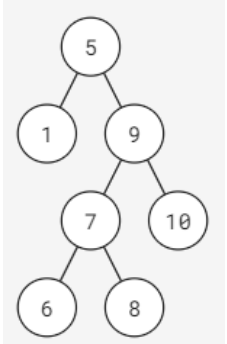
Request: Implement function:

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

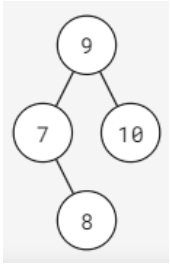
Where `root` is the root node of given binary search tree (this tree has between 0 and 100000 elements). This function returns the binary search tree after deleting all nodes whose values are outside the range `[lo, hi]` (inclusive).

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

```
// Hàm loại bỏ các nút trong (BST) có giá trị nằm ngoài khoảng [lo, hi]
// Trả về con trỏ đến gốc của cây mới
BSTNode* subtreeWithRange(BSTNode* root, int lo, int hi) {
    // Trường hợp cơ sở: nếu cây rỗng
    if (root == nullptr) {
        return nullptr;
    }
    // Nếu giá trị của nút hiện tại nhỏ hơn lo,
    // bỏ qua nút hiện tại và cây con trái, chỉ xét cây con phải
    if (root->val < lo) {
        return subtreeWithRange(root->right, lo, hi);
    }
    // Nếu giá trị của nút hiện tại lớn hơn hi,
    // bỏ qua nút hiện tại và cây con phải, chỉ xét cây con trái
    if (root->val > hi) {
        return subtreeWithRange(root->left, lo, hi);
    }

    // Nếu giá trị của nút hiện tại nằm trong khoảng [lo, hi]
    // Giữ lại nút này và tiếp tục xử lý cả cây con trái và phải
    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.