Đã bắt đầu vào lúc	Thứ năm, 17 Tháng mười một 2022, 11:23 PM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ năm, 17 Tháng mười một 2022, 11:27 PM
Thời gian thực hiện	4 phút 19 giây
Điểm	**12,00** của 12,00 (**100**%)

Câu hỏi 1

Chính xác

Điểm 1,00 của 1,00

Given a Binary tree, the task is to traverse all the nodes of the tree using Breadth First Search algorithm and print the order of visited nodes (has no blank space at the end)

```
#include<iostream>
#include<string>
#include<queue>
using namespace std;

template<class K, class V>
class BinaryTree
{
public:
    class Node;

private:
    Node *root;

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

    class Node
    {
    private:
        K key;
        V value;
        Node *pLeft, *pRight;
        friend class BinaryTree<K, V>;

    public:
        Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };

    void addNode(string posFromRoot, K key, V value)
    {
        if(posFromRoot == "")
        {
            this->root = new Node(key, value);
            return;
        }

        Node* walker = this->root;
        int l = posFromRoot.length();
        for (int i = 0; i < l-1; i++)
        {
            if (!walker)
                return;
            if (posFromRoot[i] == 'L')
                walker = walker->pLeft;
            if (posFromRoot[i] == 'R')
                walker = walker->pRight;
        }
        if(posFromRoot[l-1] == 'L')
            walker->pLeft = new Node(key, value);
        if(posFromRoot[l-1] == 'R')
            walker->pRight = new Node(key, value);
    }

    // STUDENT ANSWER BEGIN
    // STUDENT ANSWER END
};
```

You can define other functions to help you.

For example:

Test	Result

`BinaryTree<int, int> binaryTree;`
`binaryTree.addNode("",2, 4); // Add to root`
`binaryTree.addNode("L",3, 6); // Add to root's left node`
`binaryTree.addNode("R",5, 9); // Add to root's right node`
`binaryTree.BFS();`
	`4 6 9`

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

Reset answer

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.
void BFS()
{
    queue<Node*> q;
    q.push(root);
    while(q.size()!=0){
        Node*temp=q.front();
        if(temp->pLeft!=nullptr) q.push(temp->pLeft);
        if(temp->pRight!=nullptr) q.push(temp->pRight);
        cout<<temp->value<<" ";
        q.pop();
    }
}
// STUDENT ANSWER END
```

	Test	Expected	Got	

✓	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("",2, 4); // Add to root`
	`binaryTree.addNode("L",3, 6); // Add to root's left node`
	`binaryTree.addNode("R",5, 9); // Add to root's right node`
	`binaryTree.BFS();`
		`4 6 9`	`4 6 9`	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 2

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary 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.

Request: Implement function:

```
int distinctParities(BTNode* root);
```

Where `root` is the root node of given binary tree (this tree has between 0 and 100000 elements). This function returns the number of **P-nodes** the binary tree has.

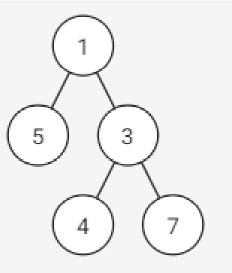
More information: A node is called as **P-node** if it satisfies these following rules:

- It has exactly 2 children.

- The sum of a subtree of this node is even, while the sum of the other subtree of this node is odd.

Example:

Given a binary tree in the following:



The number of **P-nodes** of this binary tree is 2, they are nodes 1 and 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 arr[] = {-1,0,0,2,2};
int value[] = {1,5,3,4,7};
BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);
cout << distinctParities(root);
	2

int arr[] = {-1,0,0,1,2,2,3,3,7,1};
int value[] = {13,11,22,19,2,23,26,16,22,23};
BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);
cout << distinctParities(root);
	1

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

Reset answer

```
int calsum(BTNode*root){
    if(root==nullptr) return 0;
    int s=root->val;
    s+=calsum(root->left);
    s+=calsum(root->right);
    return s;
}
int distinctParities(BTNode* root) {
    if(root==nullptr) return 0;
    int s=0;
    if(root->left!=nullptr&&root->right!=nullptr&&
    ((calsum(root->left)%2!=0&&calsum(root->right)%2==0)||(calsum(root->left)%2==0&&calsum(root->right)%2!=0
        s=1;
        //cout<<calsum(root->left)<<endl;
        //cout<<calsum(root->right)<<endl;
    }
    //cout<<s<<endl;
    //cout<<calsum(root->right)<<endl;
    //cout<<calsum(root->left)<<endl;
    s+=distinctParities(root->left);
    s+=distinctParities(root->right);
    return s;
}
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	`int arr[] = {-1,0,0,2,2};`
	`int value[] = {1,5,3,4,7};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << distinctParities(root);`
		2	2	✓

✓	`int arr[] = {-1,0,0,1,2,2,3,3,7,1};`
	`int value[] = {13,11,22,19,2,23,26,16,22,23};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << distinctParities(root);`
		1	1	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 3

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary 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.

Request: Implement function:

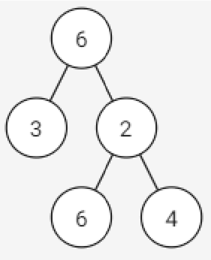
```
int greatAncestor(BTNode* root);
```

Where `root` is the root node of given binary tree (this tree has between 1 and 100000 elements). This function returns the number of **great ancestor** nodes this binary tree has.

More information: A node of the binary tree is called as **great ancestor** node if its value is greater than or equal to the values of all of its descendants. Each leaf node is also a **great ancestor** node.

Example:

Given a binary tree in the following:



All of **great ancestors** nodes this binary tree has are `6, 3, 6, 4`, therefore, return 4.

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

Test	Result

`int arr[] = {-1,0,0,2,2};`
`int value[] = {6,3,2,6,4};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(arr[0]), value);`
`cout << greatAncestor(root);`
	4

`int arr[] = {-1,0,0,2,3,3,4,5,6,4};`
`int value[] = {596,796,2168,148,1444,651,2279,1749,233,2008};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(arr[0]), value);`
`cout << greatAncestor(root);`
	5

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

Reset answer

```
bool greatancsearch(int& val,BTNode*root){
    if(root==nullptr) return 1;
    if(val<root->val) return 0;
    bool check=greatancsearch(val,root->left);
    if(check==0) return 0;
    check= greatancsearch(val,root->right);
    return check;

}
int greatAncestor(BTNode* root) {
    if(root==nullptr) return 0;
    int s=0;
    s+=greatancsearch(root->val,root);
    s+=greatAncestor(root->left);
    s+=greatAncestor(root->right);
    return s;
}
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	`int arr[] = {-1,0,0,2,2};`
	`int value[] = {6,3,2,6,4};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(arr[0]), value);`
	`cout << greatAncestor(root);`
		4	4	✓

✓	`int arr[] = {-1,0,0,2,3,3,4,5,6,4};`
	`int value[] = {596,796,2168,148,1444,651,2279,1749,233,2008};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(arr[0]), value);`
	`cout << greatAncestor(root);`
		5	5	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 4

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary 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.

Request: Implement function:

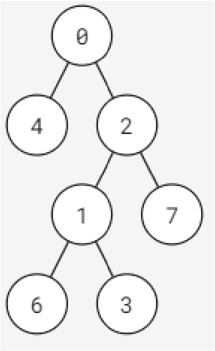
`int largestDiff(BTNode* root);`

Where `root` is the root node of given binary tree (this tree has between 2 and 100000 elements). This function returns the largest absolute difference between any node and its descendants.

More information: A node is also the descendant of itself.

Example:

Given a binary tree in the following:



The largest absolute difference is between node `0` and node `7`, therefore, return `7`.

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

Test	Result

`int arr[] = {-1,0,0,2,2,3,3};`
`int value[] = {0,4,2,1,7,6,3};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
`cout << largestDiff(root) << "\n";`
	7

`int arr[] = {-1,0};`
`int value[] = {1,0};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
`cout << largestDiff(root) << "\n";`
	1

Answer: (penalty regime: 0 %)

Reset answer

```
void smallestandbiggest(BTNode *&root,int small,int big,int&max){
    if(root==nullptr) return ;
    if(root->val<=small){
        small=root->val;
    }
    if(root->val>=big){
        big=root->val;
    }
    if(max<big-small) max=big-small;
    smallestandbiggest(root->left,small,big,max);
    smallestandbiggest(root->right,small,big,max);
}
int largestDiff(BTNode* root) {
    int s=root->val;
    int b=root->val;
    int m=0;
    smallestandbiggest(root,s,b,m);
    return m;

}
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	int arr[] = {-1,0,0,2,2,3,3};
	int value[] = {0,4,2,1,7,6,3};
	BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);
	cout << largestDiff(root) << "\n";
		7	7	✓

✓	int arr[] = {-1,0};
	int value[] = {1,0};
	BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);
	cout << largestDiff(root) << "\n";
		1	1	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 5

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary 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.

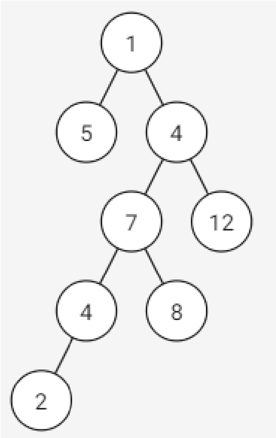
Request: Implement function:

`int longestPathSum(BTNode* root);`

Where `root` is the root node of given binary tree (this tree has between 1 and 100000 elements). This function returns the sum of the largest path from the root node to a leaf node. If there are more than one equally long paths, return the larger sum.

Example:

Given a binary tree in the following:



The longest path from the root node to the leaf node is `1-4-7-4-2`, so return the sum of this path, is `18`.

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

For example:

Test	Result

Test	Result

`int arr[] = {-1,0,0,2,2,3,3,5};`
`int value[] = {1,5,4,7,12,4,8,2};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
`cout << longestPathSum(root);`
	18

`int arr[] = {-1,0,1,0,1,4,5,3,7,3};`
`int value[] = {6,12,23,20,20,20,3,9,13,15};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
`cout << longestPathSum(root);`
	61

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

Reset answer

```

//Hung
//Hung Hoang
void sumOfLongRootToLeafPath(BTNode* root, int sum,
                      int len, int& maxLen, int& maxSum)
{
    // if true, then we have traversed a
    // root to leaf path
    if (!root) {
        // update maximum length and maximum sum
        // according to the given conditions
        if (maxLen < len) {
            maxLen = len;
            maxSum = sum;
        } else if (maxLen == len && maxSum < sum)
            maxSum = sum;
        return;
    }

    // recur for left subtree
    sumOfLongRootToLeafPath(root->left, sum + root->val,
                            len + 1, maxLen, maxSum);

    // recur for right subtree
    sumOfLongRootToLeafPath(root->right, sum + root->val,
                            len + 1, maxLen, maxSum);
}

int longestPathSum(BTNode* root) {
    if (!root)
        return 0;

    int maxSum = -999999, maxLen = 0;

    // finding the maximum sum 'maxSum' for the
    // maximum length root to leaf path
    sumOfLongRootToLeafPath(root, 0, 0, maxLen, maxSum);

    // required maximum sum
    return maxSum;

}
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	int arr[] = {-1,0,0,2,2,3,3,5};
	int value[] = {1,5,4,7,12,4,8,2};
	BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);
	cout << longestPathSum(root);
		18	18	✓

✓	int arr[] = {-1,0,1,0,1,4,5,3,7,3};
	int value[] = {6,12,23,20,20,20,3,9,13,15};
	BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);
	cout << longestPathSum(root);
		61	61	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 6

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary 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.

Request: Implement function:

`int lowestAncestor(BTNode* root, int a, int b);`

Where `root` is the root node of given binary tree (this tree has between 2 and 100000 elements). This function returns the **lowest ancestor** node's `val` of node `a` and node `b` in this binary tree (assume a and b always exist in the given binary tree).

More information:

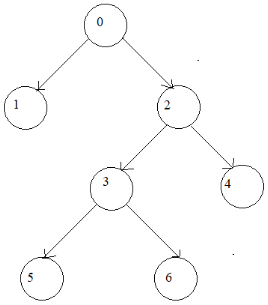
- A node is called as the **lowest ancestor** node of node `a` and node `b` if node `a` and node `b` are its descendants.

- A node is also the descendant of itself.

- On the given binary tree, each node's `val` is distinguish from the others' `val`

Example:

Given a binary tree in the following:



- The **lowest ancestor** of node `4` and node `5` is node `2`.

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 arr[] = {-1,0,0,2,2,3,3};
BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL);
cout << lowestAncestor(root, 4, 5);
	2

int arr[] = {-1,0,1,1,0,4,4,2,5,6};
BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL);
cout << lowestAncestor(root, 4, 9);
	4

Answer: (penalty regime: 0 %)

Reset answer

```
bool search(BTNode*root,int& a){
    if(root==nullptr) return 0;
    if(root->val==a) return 1;
    bool check= search(root->left,a);
    if(check==1) return 1;
    check= search(root->right,a);
    return check;
}
int lowestAncestor(BTNode* root, int a, int b) {
    if((search(root->left,a)&&search(root->left,b))){
        return lowestAncestor( root->left, a, b );
    }
    else if(search(root->right,a)&&search(root->right,b)){
        return lowestAncestor( root->right, a, b );
    }
    else return root->val;

}
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	`int arr[] = {-1,0,0,2,2,3,3};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL);`
	`cout << lowestAncestor(root, 4, 5);`
		2	2	✓

✓	`int arr[] = {-1,0,1,1,0,4,4,2,5,6};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL);`
	`cout << lowestAncestor(root, 4, 9);`
		4	4	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 7

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary 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.

Request: Implement function:

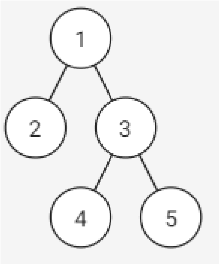
`int maximizeProduct(BTNode* root);`

Where `root` is the root node of given binary tree (this tree has between 2 and 100000 elements). This function returns the largest `P` which can be gotten after deleting an edge of this tree.

More information: Split the binary tree into two subtrees by deleting an edge of it. Take the sum of each subtree, `P` is the product of these sums.

Example:

Given a binary tree in the following:



Cut the edge between nodes 3 and 5, the P we have is `(1+2+3+4)*5=50` - it is the largest P.

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

Test	Result

int arr[] = {-1,0,0,2,2};
int value[] = {1,2,3,4,5};
BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(arr[0]), value);
cout << maximizeProduct(root);
	50

int arr[] = {-1,0,0,1,2,1,4,4,3,3};
int value[] = {4,4,5,5,5,4,0,1,3,3};
BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(arr[0]), value);
cout << maximizeProduct(root);
	285

Answer: (penalty regime: 0 %)

Reset answer

```
int max(int a, int b){
    if(a<b){
        return b;
    }
    else return a;
}
int sum(BTNode* root){
    if(!root) return 0;
    else return root->val+sum(root->left)+sum(root->right);
}
void largestP(int& maxP, BTNode* root, int sumall){
    if(!root) return;
    else{
        maxP=max(max(sum(root->left)*(-sum(root->left)+sumall),sum(root->right)*(sumall-sum(root->right))),ma
        largestP(maxP,root->right,sumall);
        largestP(maxP,root->left,sumall);
    }
}
int maximizeProduct(BTNode* root) {
    int maxP=0;
    int sumall=sum(root);
    largestP(maxP,root,sumall);
    return maxP;
}
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	`int arr[] = {-1,0,0,2,2};`
	`int value[] = {1,2,3,4,5};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(arr[0]), value);`
	`cout << maximizeProduct(root);`
		50	50	✓

✓	`int arr[] = {-1,0,0,1,2,1,4,4,3,3};`
	`int value[] = {4,4,5,5,5,4,0,1,3,3};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(arr[0]), value);`
	`cout << maximizeProduct(root);`
		285	285	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 8

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary 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.

Request: Implement function:

`int secondDeepest(BTNode* root);`

Where `root` is the root node of given binary tree (this tree has between 2 and 100000 elements). This function returns the depth of the second deepest leaf/leaves of the tree (if there is no leaf satisfying, return `-1`).

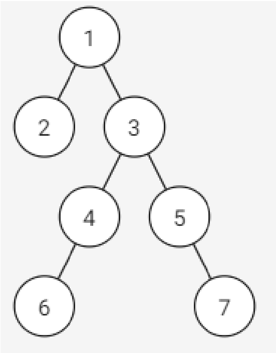
More information:

- The root has a depth of `0`.

- In a binary tree, the second deepest leaf's/leaves' depth is smaller than the deepest leaf/leaves's depth and higher than the others' depth.

Example:

Given a binary tree in the following:



The second deepest leaf is node `2`, the depth of node `2` is `1`; therefore, the function returns `1`.

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 arr[] = {-1,0,0,2,2,3,4};`
`int value[] = {1,2,3,4,5,6,7};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
`cout << secondDeepest(root);`
	1

`int arr[] = {-1,0,1,2,3,4,5,6,7,8};`
`int value[] = {1,2,3,4,5,6,7,8,9,10};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
`cout << secondDeepest(root);`
	-1

Answer: (penalty regime: 0 %)

Reset answer

```
int secondDeepest(BTNode* root) {
    queue<BTNode*> q;
    int height=0;
    int heightfirstdeep=-1;
    int heightsecondeep=-1;
    q.push(root);
    while(q.size()!=0){
        height++;
        bool deepalready=0;
        //cout<<heightfirstdeep<<endl;
        //cout<<heightsecondeep<<endl;
        for(int count=q.size();count>0;count--){
            BTNode*temp=q.front();
            if(temp->left!=nullptr) {
                q.push(temp->left);
                //cout<<"deepal"<<deepalready<<endl;
                if(!deepalready&&temp->left->left==nullptr&&temp->left->right==nullptr) {
                    //cout<<"templeft"<<temp->val;
                    //cout<<"   "<<temp->left->val<<endl;
                    //cout<<"   "<<temp->left->val<<endl;
                    heightsecondeep=heightfirstdeep;
                    heightfirstdeep=height;
                    deepalready=1;
                }
            }
            if(temp->right!=nullptr){
                q.push(temp->right);
                if(!deepalready&&temp->right->left==nullptr&&temp->right->right==nullptr) {
                    //cout<<"tempright"<<temp->right->val<<endl;
                    heightsecondeep=heightfirstdeep;
                    heightfirstdeep=height;
                    deepalready=1;
                }
            }
            q.pop();
        }
    }
    return heightsecondeep;
}
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	`int arr[] = {-1,0,0,2,2,3,4};`
	`int value[] = {1,2,3,4,5,6,7};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << secondDeepest(root);`
		`1`	`1`	✓

✓	`int arr[] = {-1,0,1,2,3,4,5,6,7,8};`
	`int value[] = {1,2,3,4,5,6,7,8,9,10};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << secondDeepest(root);`
		`-1`	`-1`	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 9

Chính xác

Điểm 1,00 của 1,00

Class **BTNode** is used to store a node in binary 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 (integer, in segment `[0,9]`), `left` and `right` are the pointers to the left node and right node of it, respectively.

Request: Implement function:

`int sumDigitPath(BTNode* root);`

Where `root` is the root node of given binary tree (this tree has between 2 and 100000 elements). This function returns the sum of all **digit path** numbers of this binary tree (the result may be large, so you must use `mod 27022001` before returning).

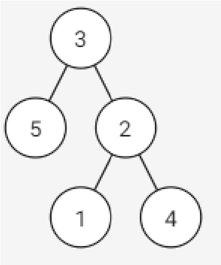
More information:

- A path is called as **digit path** if it is a path from the root node to the leaf node of the binary tree.

- Each **digit path** represents a number in order, each node's `val` of this path is a digit of this number, while root's `val` is the first digit.

Example:

Given a binary tree in the following:



All of the **digit paths** are `3-5`, `3-2-1`, `3-2-4`; and the number reprensted by them are `35`, `321`, `324`, respectively. The sum of them (after `mod 27022001`) is `680`.

Note: In this exercise, the libraries `iostream`, `queue`, `stack`, `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 arr[] = {-1,0,0,2,2};`
`int value[] = {3,5,2,1,4};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
`cout << sumDigitPath(root);`
	680

`int arr[] = {-1,0,0};`
`int value[] = {1,2,3};`
`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
`cout << sumDigitPath(root);`
	25

Answer: (penalty regime: 0 %)

Reset answer

```
long long sumrec(BTNode* root,long long b){
    if(root==nullptr) return 0;
    if(root->left==nullptr&&root->right==nullptr) {
        //cout<<b*10+root->val<<endl;
        return (b*10+root->val);
    }
    long long left;
    long long right;
    left=sumrec(root->left,(b*10+root->val)%27022001);
    right=sumrec(root->right,(b*10+root->val)%27022001);
    return (left+right);
}
int sumDigitPath(BTNode* root) {
    return sumrec(root,0)%27022001;
}
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	`int arr[] = {-1,0,0,2,2};`
	`int value[] = {3,5,2,1,4};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << sumDigitPath(root);`
		`680`	`680`	✓

✓	`int arr[] = {-1,0,0};`
	`int value[] = {1,2,3};`
	`BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);`
	`cout << sumDigitPath(root);`
		`25`	`25`	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 10

Chính xác

Điểm 1,00 của 1,00

Given a Binary tree, the task is to count the number of nodes with two children

```
#include<iostream>
#include<string>
using namespace std;

template<class K, class V>
class BinaryTree
{
public:
    class Node;

private:
    Node *root;

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

    class Node
    {
    private:
        K key;
        V value;
        Node *pLeft, *pRight;
        friend class BinaryTree<K, V>;

    public:
        Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };

    void addNode(string posFromRoot, K key, V value)
    {
        if(posFromRoot == "")
        {
            this->root = new Node(key, value);
            return;
        }

        Node* walker = this->root;
        int l = posFromRoot.length();
        for (int i = 0; i < l-1; i++)
        {
            if (!walker)
                return;
            if (posFromRoot[i] == 'L')
                walker = walker->pLeft;
            if (posFromRoot[i] == 'R')
                walker = walker->pRight;
        }
        if(posFromRoot[l-1] == 'L')
            walker->pLeft = new Node(key, value);
        if(posFromRoot[l-1] == 'R')
            walker->pRight = new Node(key, value);
    }

    // STUDENT ANSWER BEGIN
    // STUDENT ANSWER END
};
```

You can define other functions to help you.

For example:

Test	Result

BinaryTree<int, int> binaryTree;
binaryTree.addNode("",2, 4); // Add to root
binaryTree.addNode("L",3, 6); // Add to root's left node
binaryTree.addNode("R",5, 9); // Add to root's right node
cout << binaryTree.countTwoChildrenNode();
	1

BinaryTree<int, int> binaryTree;
binaryTree.addNode("",2, 4);
binaryTree.addNode("L",3, 6);
binaryTree.addNode("R",5, 9);
binaryTree.addNode("LL",4, 10);
binaryTree.addNode("LR",6, 2);
cout << binaryTree.countTwoChildrenNode();
	2

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

Reset answer

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.
int countrec(Node* root){
    if(root==nullptr) return 0;
    int sum=0;
    if(root->pLeft!=nullptr && root->pRight!=nullptr) sum++;
    sum+=countrec(root->pLeft);
    sum+=countrec(root->pRight);
    return sum;

}
int countTwoChildrenNode()
{
    return countrec(this->root);
}
// STUDENT ANSWER END
```

	Test	Expected	Got	

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4); // Add to root
	binaryTree.addNode("L",3, 6); // Add to root's left node
	binaryTree.addNode("R",5, 9); // Add to root's right node
	cout << binaryTree.countTwoChildrenNode();
		1	1	✓

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LR",6, 2);
	cout << binaryTree.countTwoChildrenNode();
		2	2	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 11

Chính xác

Điểm 1,00 của 1,00

Given class **BinaryTree**, you need to finish methods **getHeight()**, **preOrder()**, **inOrder()**, **postOrder()**.

```
#include <iostream>
#include <string>
#include <algorithm>
#include <sstream>
using namespace std;

template<class K, class V>
class BinaryTree
{
public:
    class Node;
private:
    Node* root;
public:
    BinaryTree() : root(nullptr) {}
    ~BinaryTree()
    {
        // You have to delete all Nodes in BinaryTree. However in this task, you can ignore it.
    }
    class Node
    {
    private:
        K key;
        V value;
        Node* pLeft, * pRight;
        friend class BinaryTree<K, V>;
    public:
        Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
    void addNode(string posFromRoot, K key, V value)
    {
        if (posFromRoot == "")
        {
            this->root = new Node(key, value);
            return;
        }
        Node* walker = this->root;
        int l = posFromRoot.length();
        for (int i = 0; i < l - 1; i++)
        {
            if (!walker)
                return;
            if (posFromRoot[i] == 'L')
                walker = walker->pLeft;
            if (posFromRoot[i] == 'R')
                walker = walker->pRight;
        }
        if (posFromRoot[l - 1] == 'L')
            walker->pLeft = new Node(key, value);
        if (posFromRoot[l - 1] == 'R')
            walker->pRight = new Node(key, value);
    }
    // STUDENT ANSWER BEGIN

    // STUDENT ANSWER END
};
```

For example:

Test	Result

`BinaryTree<int, int> binaryTree;`
`binaryTree.addNode("", 2, 4); // Add to root`
`binaryTree.addNode("L", 3, 6); // Add to root's left node`
`binaryTree.addNode("R", 5, 9); // Add to root's right node`
 	
`cout << binaryTree.getHeight() << endl;`
`cout << binaryTree.preOrder() << endl;`
`cout << binaryTree.inOrder() << endl;`
`cout << binaryTree.postOrder() << endl;`
	2
	4 6 9
	6 4 9
	6 9 4

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

Reset answer

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.
int heightrec(Node*root){
    if(root==nullptr) return 0;
    int left=heightrec(root->pLeft);
    int right=heightrec(root->pRight);
    return max(left,right)+1;
}
int getHeight() {
    // TODO: return height of the binary tree.
    return heightrec(this->root);
}
string prerec(Node*root){
    if(root==nullptr) return "";
    string s="";
    s+=to_string(root->value)+" ";
    s+=prerec(root->pLeft);
    s+=prerec(root->pRight);
    return s;
}
string preOrder() {
    // TODO: return the sequence of values of nodes in pre-order.
```

	Test	Expected	Got	

✓	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("", 2, 4); // Add to root`
	`binaryTree.addNode("L", 3, 6); // Add to root's left node`
	`binaryTree.addNode("R", 5, 9); // Add to root's right node`
	
	`cout << binaryTree.getHeight() << endl;`
	`cout << binaryTree.preOrder() << endl;`
	`cout << binaryTree.inOrder() << endl;`
	`cout << binaryTree.postOrder() << endl;`
		2
		4 6 9
		6 4 9
		6 9 4
			2
			4 6 9
			6 4 9
			6 9 4
				✓

✓	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("", 2, 4);`
	
	`cout << binaryTree.getHeight() << endl;`
	`cout << binaryTree.preOrder() << endl;`
	`cout << binaryTree.inOrder() << endl;`
	`cout << binaryTree.postOrder() << endl;`
		1
		4
		4
		4
			1
			4
			4
			4
				✓

	Test	Expected	Got	

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("", 2, 4);
	binaryTree.addNode("L", 3, 6);
	binaryTree.addNode("R", 5, 9);
	binaryTree.addNode("LL", 4, 10);
	binaryTree.addNode("LR", 6, 2);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		3
		4 6 10 2 9
		10 6 2 4 9
		10 2 6 9 4
			3
			4 6 10 2 9
			10 6 2 4 9
			10 2 6 9 4
				✓

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("", 2, 4);
	binaryTree.addNode("L", 3, 6);
	binaryTree.addNode("R", 5, 9);
	binaryTree.addNode("LL", 4, 10);
	binaryTree.addNode("RL", 6, 2);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		3
		4 6 10 9 2
		10 6 4 2 9
		10 6 2 9 4
			3
			4 6 10 9 2
			10 6 4 2 9
			10 6 2 9 4
				✓

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LLL",6, 2);
	binaryTree.addNode("LLLR",7, 7);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 9
		2 7 10 6 4 9
		7 2 10 6 9 4
			5
			4 6 10 2 7 9
			2 7 10 6 4 9
			7 2 10 6 9 4
				✓

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LLL",6, 2);
	binaryTree.addNode("LLLR",7, 7);
	binaryTree.addNode("RR",8, 30);
	binaryTree.addNode("RL",9, 307);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 9 307 30
		2 7 10 6 4 307 9 30
		7 2 10 6 307 30 9 4
			5
			4 6 10 2 7 9 307 30
			2 7 10 6 4 307 9 30
			7 2 10 6 307 30 9 4
				✓

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LR",6, -3);
	binaryTree.addNode("LLL",7, 2);
	binaryTree.addNode("LLLR",8, 7);
	binaryTree.addNode("RR",9, 30);
	binaryTree.addNode("RL",10, 307);
	
	cout << binaryTree.getHeight() << endl;
	cout << binaryTree.preOrder() << endl;
	cout << binaryTree.inOrder() << endl;
	cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 -3 9 307 30
		2 7 10 6 -3 4 307 9 30
		7 2 10 -3 6 307 30 9 4
			5
			4 6 10 2 7 -3 9 307 30
			2 7 10 6 -3 4 307 9 30
			7 2 10 -3 6 307 30 9 4
				✓

	Test	Expected	Got	

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LR",6, -3);
	binaryTree.addNode("LLL",7, 2);
	binaryTree.addNode("LLLR",8, 7);
	binaryTree.addNode("RR",9, 30);
	binaryTree.addNode("RL",10, 307);
	binaryTree.addNode("RLL",11, 2000);
	binaryTree.addNode("RLR",12, 2000);
	
	    cout << binaryTree.getHeight() << endl;
	    cout << binaryTree.preOrder() << endl;
	    cout << binaryTree.inOrder() << endl;
	    cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 -3 9 307 2000 2000 30
		2 7 10 6 -3 4 2000 307 2000 9 30
		7 2 10 -3 6 2000 2000 307 30 9 4
			5
			4 6 10 2 7 -3 9 307 2000 2000 30
			2 7 10 6 -3 4 2000 307 2000 9 30
			7 2 10 -3 6 2000 2000 307 30 9 4
				✓

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LR",6, -3);
	binaryTree.addNode("LLL",7, 2);
	binaryTree.addNode("LLLR",8, 7);
	binaryTree.addNode("RR",9, 30);
	binaryTree.addNode("RL",10, 307);
	binaryTree.addNode("RLL",11, 2000);
	
	    cout << binaryTree.getHeight() << endl;
	    cout << binaryTree.preOrder() << endl;
	    cout << binaryTree.inOrder() << endl;
	    cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 -3 9 307 2000 30
		2 7 10 6 -3 4 2000 307 9 30
		7 2 10 -3 6 2000 307 30 9 4
			5
			4 6 10 2 7 -3 9 307 2000 30
			2 7 10 6 -3 4 2000 307 9 30
			7 2 10 -3 6 2000 307 30 9 4
				✓

✓	BinaryTree<int, int> binaryTree;
	binaryTree.addNode("",2, 4);
	binaryTree.addNode("L",3, 6);
	binaryTree.addNode("R",5, 9);
	binaryTree.addNode("LL",4, 10);
	binaryTree.addNode("LR",6, -3);
	binaryTree.addNode("LLL",7, 2);
	binaryTree.addNode("LLLR",8, 7);
	binaryTree.addNode("RR",9, 30);
	binaryTree.addNode("RL",10, 307);
	binaryTree.addNode("RLL",11, 2000);
	binaryTree.addNode("RLLL",11, 2000);
	
	    cout << binaryTree.getHeight() << endl;
	    cout << binaryTree.preOrder() << endl;
	    cout << binaryTree.inOrder() << endl;
	    cout << binaryTree.postOrder() << endl;
		5
		4 6 10 2 7 -3 9 307 2000 2000 30
		2 7 10 6 -3 4 2000 2000 307 9 30
		7 2 10 -3 6 2000 2000 307 30 9 4
			5
			4 6 10 2 7 -3 9 307 2000 2000 30
			2 7 10 6 -3 4 2000 2000 307 9 30
			7 2 10 -3 6 2000 2000 307 30 9 4
				✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 12

Chính xác

Điểm 1,00 của 1,00

Given a Binary tree, the task is to calculate the sum of leaf nodes. (Leaf nodes are nodes which have no children)

```
template<class K, class V>
class BinaryTree
{
public:
    class Node;
private:
    Node *root;

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

    class Node
    {
    private:
        K key;
        V value;
        Node *pLeft, *pRight;
        friend class BinaryTree<K, V>;

    public:
        Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };

    void addNode(string posFromRoot, K key, V value)
    {
        if(posFromRoot == "")
        {
            this->root = new Node(key, value);
            return;
        }

        Node* walker = this->root;
        int l = posFromRoot.length();
        for (int i = 0; i < l-1; i++)
        {
            if (!walker)
                return;
            if (posFromRoot[i] == 'L')
                walker = walker->pLeft;
            if (posFromRoot[i] == 'R')
                walker = walker->pRight;
        }
        if(posFromRoot[l-1] == 'L')
            walker->pLeft = new Node(key, value);
        if(posFromRoot[l-1] == 'R')
            walker->pRight = new Node(key, value);
    }
    //Helping functions
    int sumOfLeafs(){
        //TODO
     }
};
```

You can write other functions to achieve this task.

For example:

Test	Result

`BinaryTree<int, int> binaryTree;`
`binaryTree.addNode("", 2, 4);`
`cout << binaryTree.sumOfLeafs();`
	4

`BinaryTree<int, int> binaryTree;`
`binaryTree.addNode("", 2, 4);`
`binaryTree.addNode("L", 3, 6);`
`binaryTree.addNode("R", 5, 9);`
`cout << binaryTree.sumOfLeafs();`
	15

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

Reset answer

```
//Helping functions
int sumrec(Node*root){
    if(root==nullptr) return 0;
    if(root->pLeft==nullptr&&root->pRight==nullptr) return root->value;
    int sum=0;
    sum+=sumrec(root->pLeft);
    sum+=sumrec(root->pRight);
    return sum;
}
int sumOfLeafs(){
    //TODO
    return sumrec(root);

}
```

	Test	Expected	Got	

✓	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("", 2, 4);`
	`cout << binaryTree.sumOfLeafs();`
		4	4	✓

✓	`BinaryTree<int, int> binaryTree;`
	`binaryTree.addNode("", 2, 4);`
	`binaryTree.addNode("L", 3, 6);`
	`binaryTree.addNode("R", 5, 9);`
	`cout << binaryTree.sumOfLeafs();`
		15	15	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.