

Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Sáu, 22 tháng 3 2024, 9:55 PM
Kết thúc lúc	Thứ Tư, 27 tháng 3 2024, 4:07 PM
Thời gian thực hiện	4 Các ngày 18 giờ



Câu hỏi 1

Đúng

Đạt điểm 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: 0 %)

Reset answer

```
1 // STUDENT ANSWER BEGIN
2 // You can define other functions here to help you.
3
4 void BFS()
5 {
6     if (root == NULL) return;
7     queue<Node*> q;
8     q.push(root);
9     while(q.size()){
10         Node* tmp = q.front();
11         q.pop();
12         cout << tmp->value << " ";
13         if (tmp->pLeft) q.push(tmp->pLeft);
14         if (tmp->pRight) q.push(tmp->pRight);
15     }
16 }
17 // STUDENT ANSWER END
```

	Test	Expected	Got	
✓	<pre>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();</pre>	4 6 9	4 6 9	✓



Passed all tests! 



Câu hỏi **2**

Đúng

Đạt điểm 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:

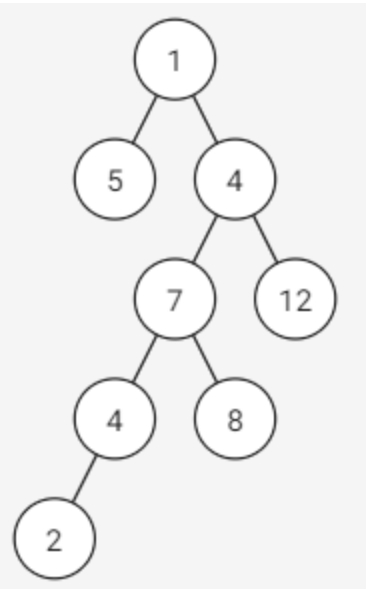
```
int longestPathSum(BTNode* root);
```

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.

Explanation of function `createTree`: The function has three parameters. The first two parameters take in an array containing the parent of each Node of the binary tree, and the third parameter takes in an array representing the respective values of the Nodes. After processing, the function will construct the binary tree and return the address of the root Node. Note that the root Node is designated with a parent value of -1.

Example:

```
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);
```

`arr[0]=-1` means the Node containing the value `value[0]=3` will be the root Node. Also, since `arr[1]=arr[2]=0`, it implies that the Nodes containing the values `value[1]=5` and `value[2]=2` will have the Node containing the value `value[0]=3` as their parent. Lastly, since `arr[3]=arr[4]=2`, it means the Nodes containing the values `value[3]=1` and `value[4]=4` will have the Node with the value `value[2]=2` as their parent. Final tree of this example are shown in the figure above.
Note that whichever Node appears first in the `arr` sequence will be the left Node, and the TestCase always ensures that the resulting tree is a binary tree.

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
<pre>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);</pre>	18

Test	Result
<pre>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);</pre>	61

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

Reset answer

```
1 void longestPathSumRec(BTNode* root, int sum, int height, int &heightMax, int& result) {
2     if (!root) {
3         if (heightMax < height) {
4             heightMax = height;
5             result = sum;
6         } else if (heightMax == height && result < sum)
7             result = sum;
8         return;
9     }
10    longestPathSumRec(root->left, sum + root->val, height + 1, heightMax,result);
11    longestPathSumRec(root->right, sum + root->val, height + 1, heightMax,result);
12 }
13 int longestPathSum(BTNode* root) {
14     int heightMax = 0;
15     int result = -9999;
16     longestPathSumRec(root, 0, 0, heightMax,result);
17     return result;
18 }
```





	Test	Expected	Got	
✓	<pre>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);</pre>	18	18	✓
✓	<pre>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);</pre>	61	61	✓

Passed all tests! ✓

Câu hỏi 3

Đúng

Đạt điểm 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).

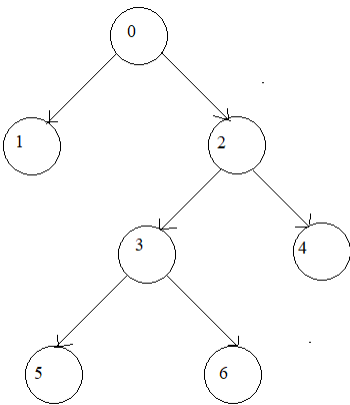
More information:

- 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.

Explanation of function `createTree`: The function has three parameters. The first two parameters take in an array containing the parent of each Node of the binary tree, and the third parameter takes in an array representing the respective values of the Nodes. After processing, the function will construct the binary tree and return the address of the root Node. Note that the root Node is designated with a parent value of -1.

Example:

```
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);
```

`arr[0]=-1` means the Node containing the value `value[0]=3` will be the root Node. Also, since `arr[1]=arr[2]=0`, it implies that the Nodes containing the values `value[1]=5` and `value[2]=2` will have the Node containing the value `value[0]=3` as their parent. Lastly, since `arr[3]=arr[4]=2`, it means the Nodes containing the values `value[3]=1` and `value[4]=4` will have the Node with the value `value[2]=2` as their parent. Final tree of this example are shown in the figure above.
Note that whichever Node appears first in the `arr` sequence will be the left Node, and the TestCase always ensures that the resulting tree is a binary tree.

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
<pre>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);</pre>	2
<pre>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);</pre>	4

Answer: (penalty regime: 0 %)

Reset answer

```
1 int lowestAncestor(BTNode* root, int a, int b) {  
2     if(root == NULL) return -9999;  
3     if(root->val == a){  
4         return root->val;  
5     }  
6     else if(root->val == b){  
7         return root->val;  
8     }  
9     int left = lowestAncestor(root->left,a,b);  
10    int right = lowestAncestor(root->right,a,b);  
11    if(left != -9999 && right != -9999) return root->val;  
12    else if(left != -9999) return left;  
13    return right;  
14 }
```

	Test	Expected	Got	
✓	<pre>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);</pre>	2	2	✓
✓	<pre>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);</pre>	4	4	✓

Passed all tests! ✓

Câu hỏi 4

Đúng

Đạt điểm 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).

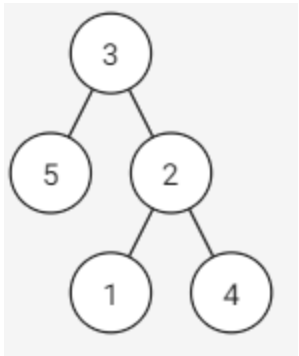
More information:

- 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.

Explanation of function createTree: The function has three parameters. The first two parameters take in an array containing the parent of each Node of the binary tree, and the third parameter takes in an array representing the respective values of the Nodes. After processing, the function will construct the binary tree and return the address of the root Node. Note that the root Node is designated with a parent value of -1.

Example:

```
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);
```

arr[0]=-1 means the Node containing the value value[0]=3 will be the root Node. Also, since arr[1]=arr[2]=0, it implies that the Nodes containing the values value[1]=5 and value[2]=2 will have the Node containing the value value[0]=3 as their parent. Lastly, since arr[3]=arr[4]=2, it means the Nodes containing the values value[3]=1 and value[4]=4 will have the Node with the value value[2]=2 as their parent. Final tree of this example are shown in the figure above.
Note that whichever Node appears first in the arr sequence will be the left Node, and the TestCase always ensures that the resulting tree is a binary tree.

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
<pre>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);</pre>	680

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

Answer: (penalty regime: 0 %)

Reset answer

```
1 int sumDigitPath(BTreeNode* root, int sum = 0) {
2     if (root == NULL) return 0;
3     if (root->left == NULL && root->right == NULL) return (sum*10+ root->val) % 27022001;
4     sum = (sum*10) % 27022001;
5     return (sumDigitPath(root->left,sum + root->val)% 27022001 + sumDigitPath(root->right,sum + root->val)% 27022001) % 27022001;
6 }
```



	Test	Expected	Got	
✓	<pre>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);</pre>	680	680	✓
✓	<pre>int arr[] = {-1,0,0}; int value[] = {1,2,3}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << sumDigitPath(root);</pre>	25	25	✓

Passed all tests! ✓

Câu hỏi **5**

Đúng

Đạt điểm 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

```
1 | int countTwoChildrenNode()
2 | {
3 |     return countTwoChildrenNodeREC(root);
4 | }
5 | int countTwoChildrenNodeREC(Node* root)
6 | {
7 |     if(root == nullptr) return 0;
8 |     else if(root->pLeft != nullptr && root->pRight != nullptr) return 1+ countTwoChildrenNodeREC(root->pLeft) + countTwoChildrenNodeREC(root->pRight);
9 |     return countTwoChildrenNodeREC(root->pLeft) + countTwoChildrenNodeREC(root->pRight);
10| }
```





	Test	Expected	Got	
✓	<pre>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();</pre>	1	1	✓
✓	<pre>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();</pre>	2	2	✓

Passed all tests! ✓



Câu hỏi 6

Đúng

Đạt điểm 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	2 4 6 9 6 4 9 6 9 4
cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl;	

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

Reset answer

```
1 | int getHeightRec(Node* root) {
2 |     if(root == NULL) return 0;
3 |     return 1 + max( getHeightRec(root->pLeft), getHeightRec(root->pRight));
4 | }
5 |
6 | int getHeight() {
7 |     // TODO: return height of the binary tree.
8 |     return getHeightRec(root);
9 | }
10 |
11 | void preOrderRec(Node* root, string & result) {
12 |     if(root == NULL) return;
13 |     result += to_string(root->value) + " ";
14 |     preOrderRec(root->pLeft,result);
15 |     preOrderRec(root->pRight,result);
16 | }
17 | string preOrder() {
18 |     // TODO: return the sequence of values of nodes in in-order.
19 |     string result;
20 |     preOrderRec(root,result);
21 |     return result;
```

```
22 }
23 void inOrderRec(Node* root, string & result) {
24     if(root == NULL) return;
25     inOrderRec(root->pLeft,result);
26     result += to_string(root->value) + " ";
27     inOrderRec(root->pRight,result);
28 }
29 string inOrder() {
30     // TODO: return the sequence of values of nodes in in-order.
31     string result;
32     inOrderRec(root,result);
33     return result;
34 }
35
36 void postOrderRec(Node* root, string & result) {
37     if(root == NULL) return;
38     postOrderRec(root->pLeft,result);
39     postOrderRec(root->pRight,result);
40     result += to_string(root->value) + " ";
41 }
42 string postOrder() {
43     // TODO: return the sequence of values of nodes in in-order.
44     string result;
45     postOrderRec(root,result);
46     return result;
47 }
```

	Test	Expected	Got	
✓	<pre>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;</pre>	<pre>2 4 6 9 6 4 9 6 9 4</pre>	<pre>2 4 6 9 6 4 9 6 9 4</pre>	✓



	Test	Expected	Got	
✓	<pre>BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl;</pre>	<pre>1 4 4 4</pre>	<pre>1 4 4 4</pre>	✓
✓	<pre>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;</pre>	<pre>3 4 6 10 2 9 10 6 2 4 9 10 2 6 9 4</pre>	<pre>3 4 6 10 2 9 10 6 2 4 9 10 2 6 9 4</pre>	✓
✓	<pre>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;</pre>	<pre>3 4 6 10 9 2 10 6 4 2 9 10 6 2 9 4</pre>	<pre>3 4 6 10 9 2 10 6 4 2 9 10 6 2 9 4</pre>	✓
✓	<pre>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;</pre>	<pre>5 4 6 10 2 7 9 2 7 10 6 4 9 7 2 10 6 9 4</pre>	<pre>5 4 6 10 2 7 9 2 7 10 6 4 9 7 2 10 6 9 4</pre>	✓



	Test	Expected	Got	
✓	<pre>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;</pre>	<pre>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</pre>	<pre>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</pre>	✓
✓	<pre>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;</pre>	<pre>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</pre>	<pre>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</pre>	✓
✓	<pre>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;</pre>	<pre>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</pre>	<pre>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</pre>	✓



	Test	Expected	Got	
✓	<pre>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;</pre>	<pre>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</pre>	<pre>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</pre>	✓
✓	<pre>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;</pre>	<pre>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</pre>	<pre>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</pre>	✓

Passed all tests! ✓



Câu hỏi 7

Đúng

Đạt điểm 1,00

Given a Binary tree, the task is to calculate the sum of leaf nodes. (Leaf nodes are nodes which have no 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);
}
//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: 0 %)

Reset answer

```
1 void inorderTraversal()
2 {
3     inorderTraversal(root);
4     cout << endl;
5 }
6 int sumOfLeafsRec(Node* root){
7     if(root == NULL) return 0;
8     else if(root->pLeft == NULL && root->pRight == NULL) return root->value;
9     return sumOfLeafsRec(root->pLeft) + sumOfLeafsRec(root->pRight);
10 }
11
12 int sumOfLeafs(){
13     return sumOfLeafsRec(root);
14 }
```



	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! ✓

