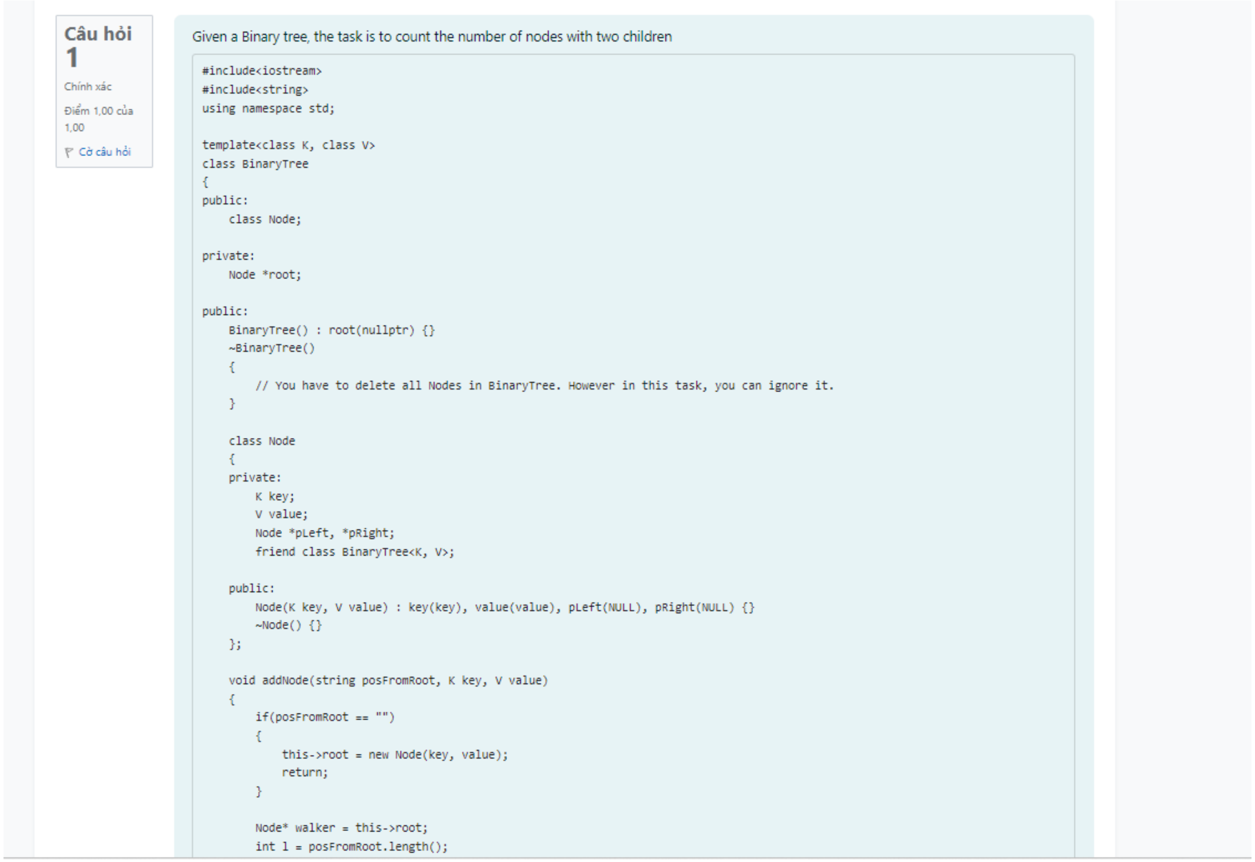


Câu hỏi **1**

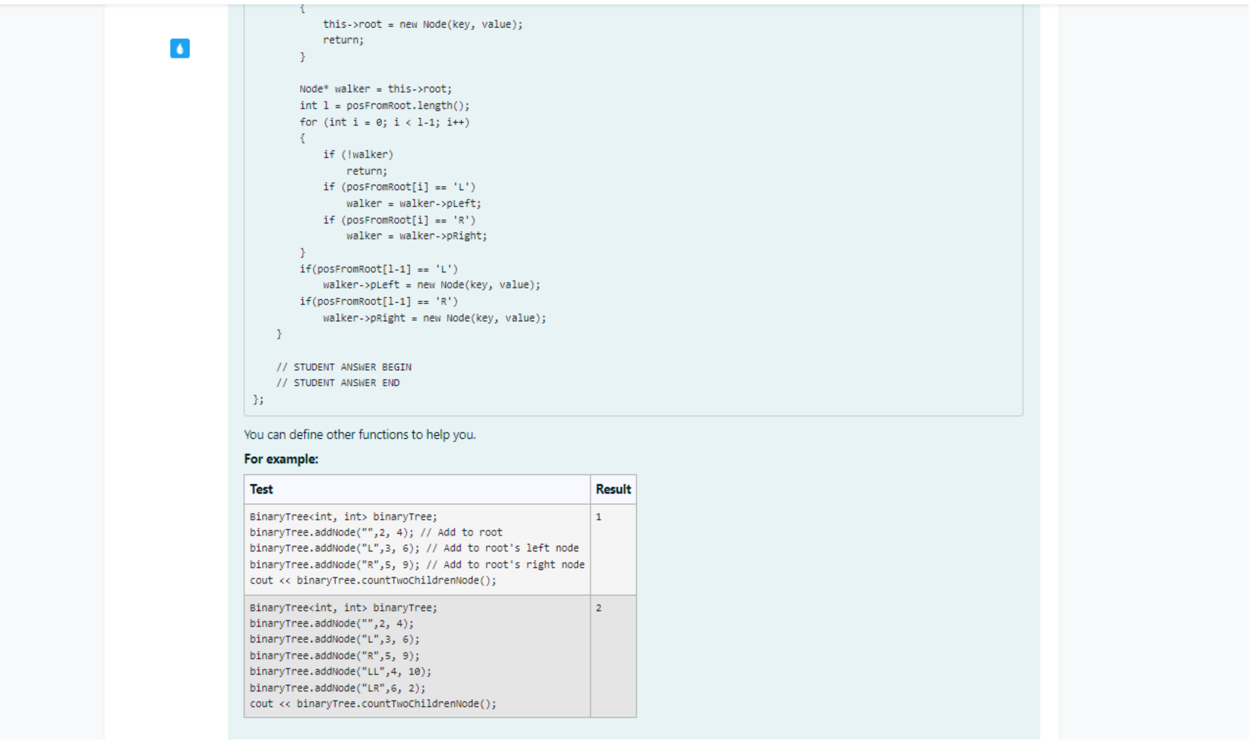
Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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();
```



```
        {
            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
};
```

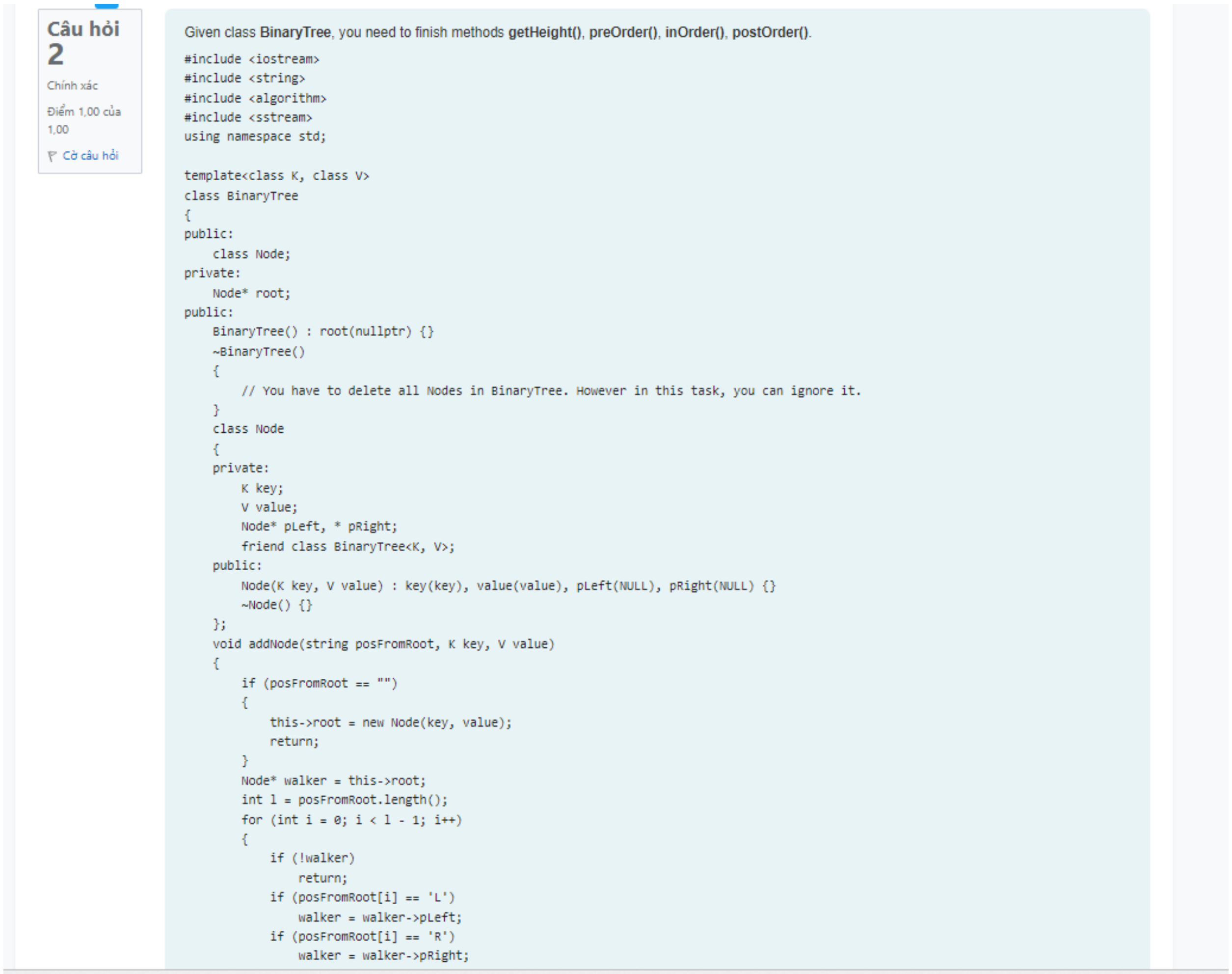
You can define other functions to help you.

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



Câu hỏi **2**

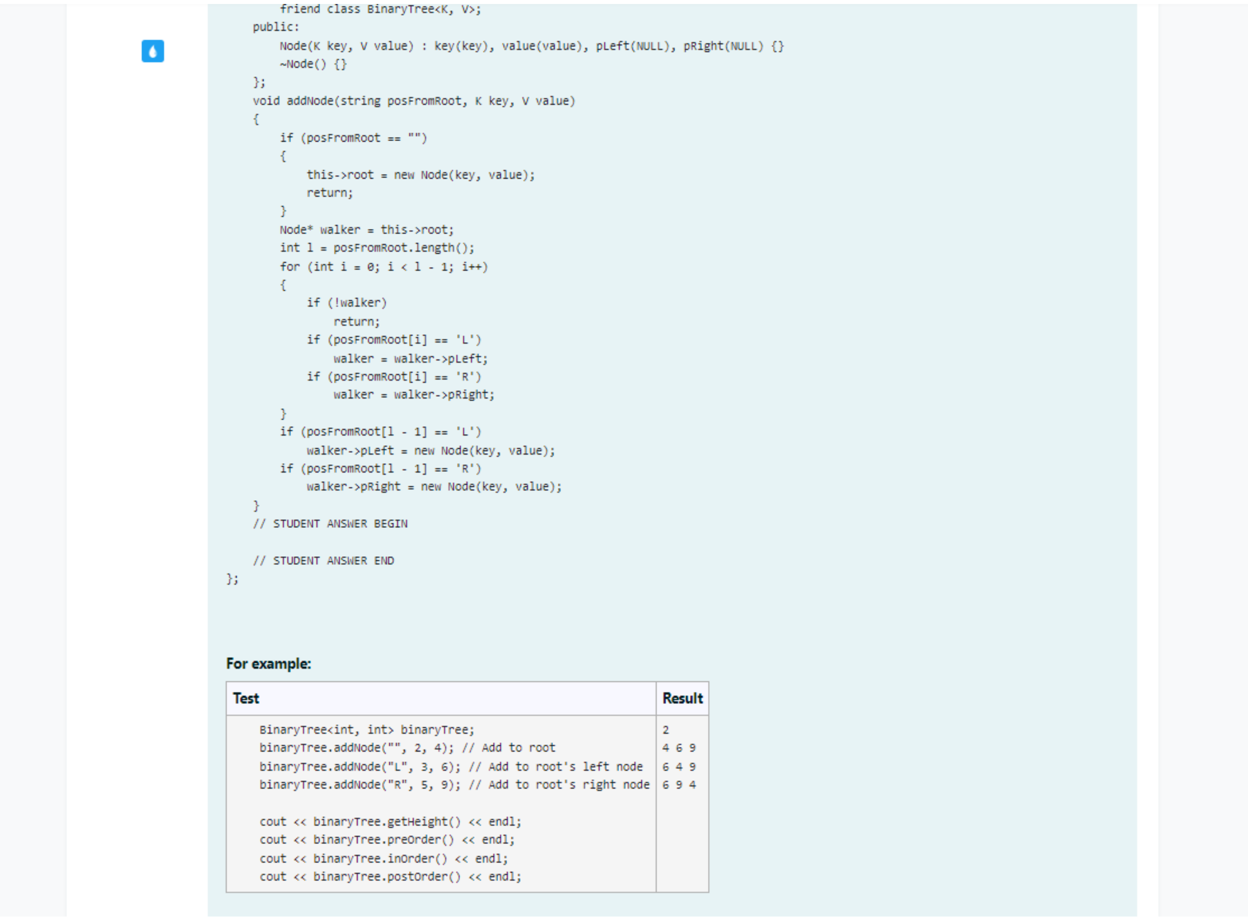
Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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;
```



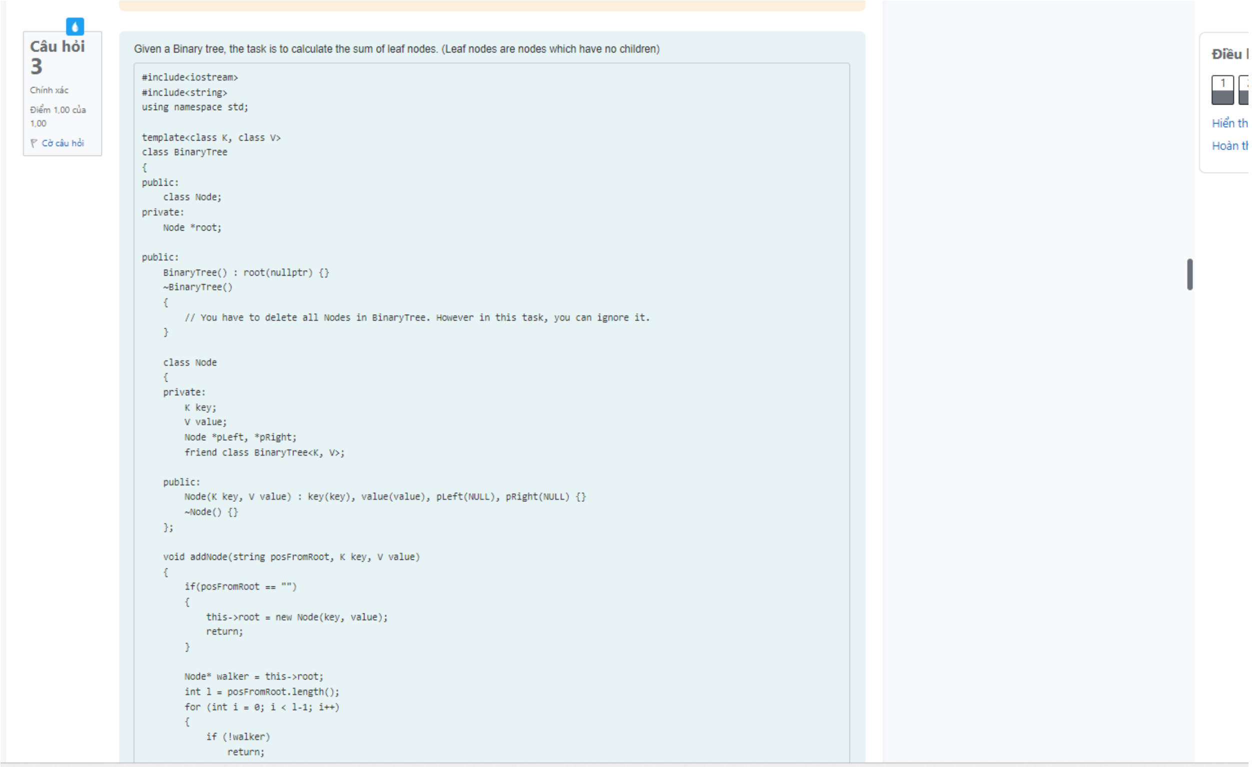
```
        friend class BinaryTree<K, V>;
    public:
        Node(K key, V value) : key(key), value(value), pLeft(NULL), pRight(NULL) {}
        ~Node() {}
    };
    void addNode(string posFromRoot, K key, V value)
    {
        if (posFromRoot == "")
        {
            this->root = new Node(key, 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:

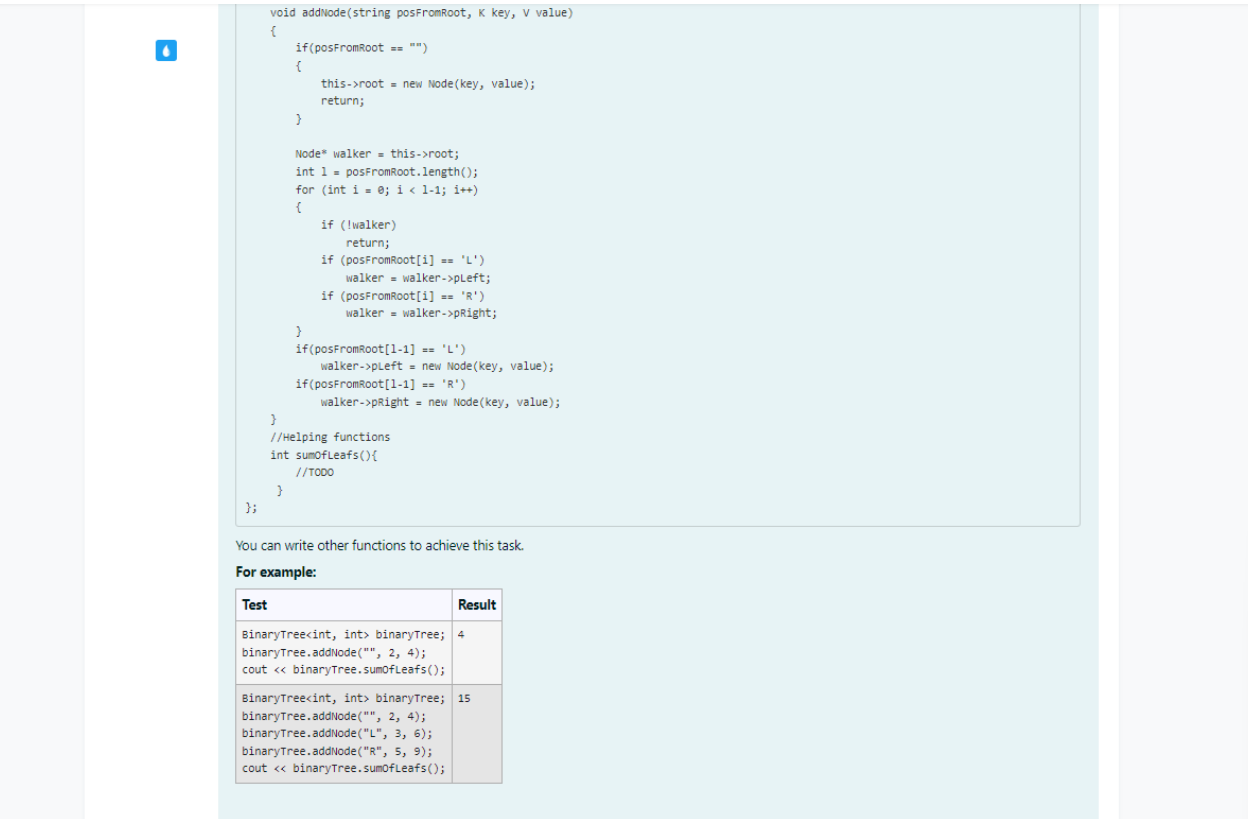
Test	Result

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



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;
```



```
    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
     }
};
```

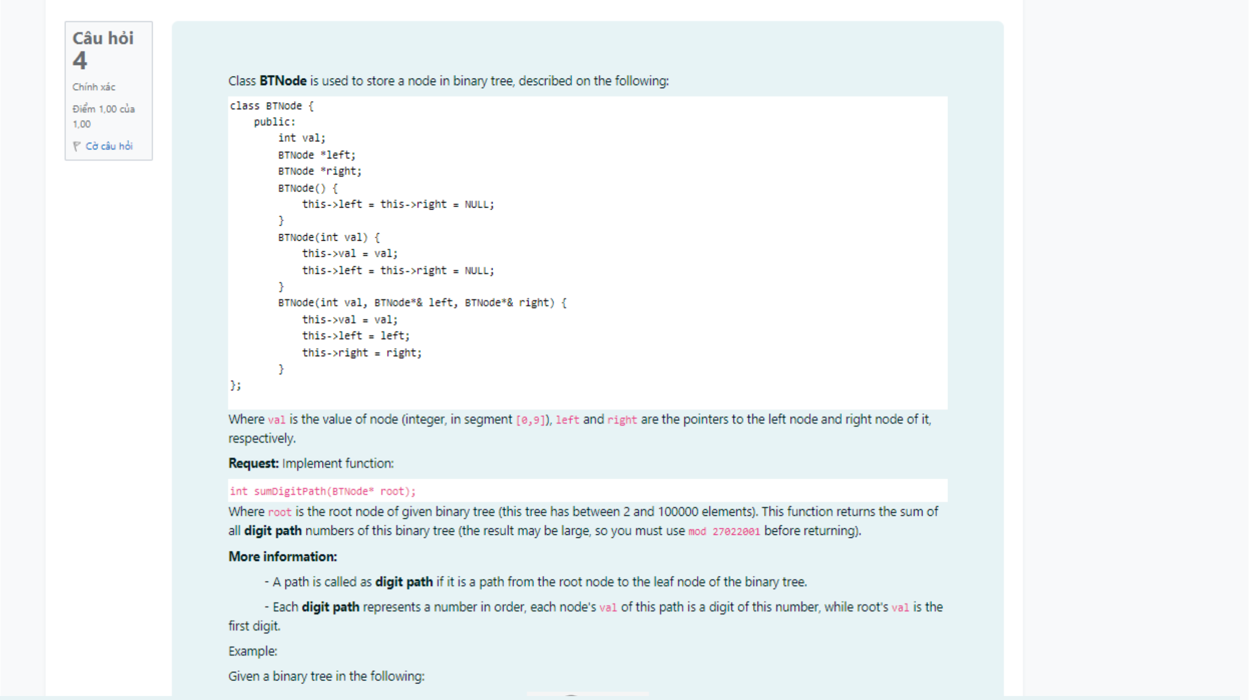
You can write other functions to achieve this task.

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



Câu hỏi **4**

Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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).

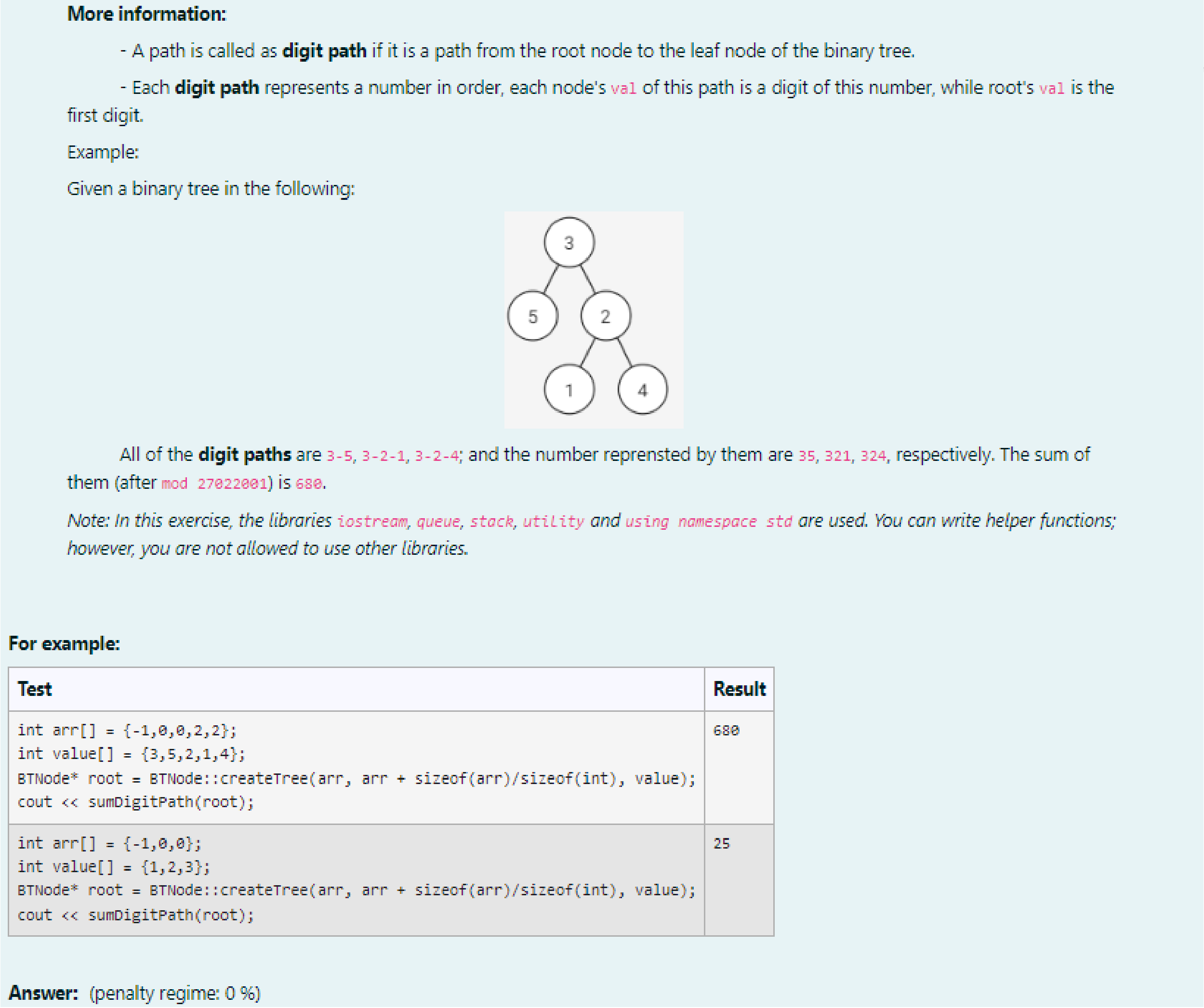
More information:

- 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:



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.

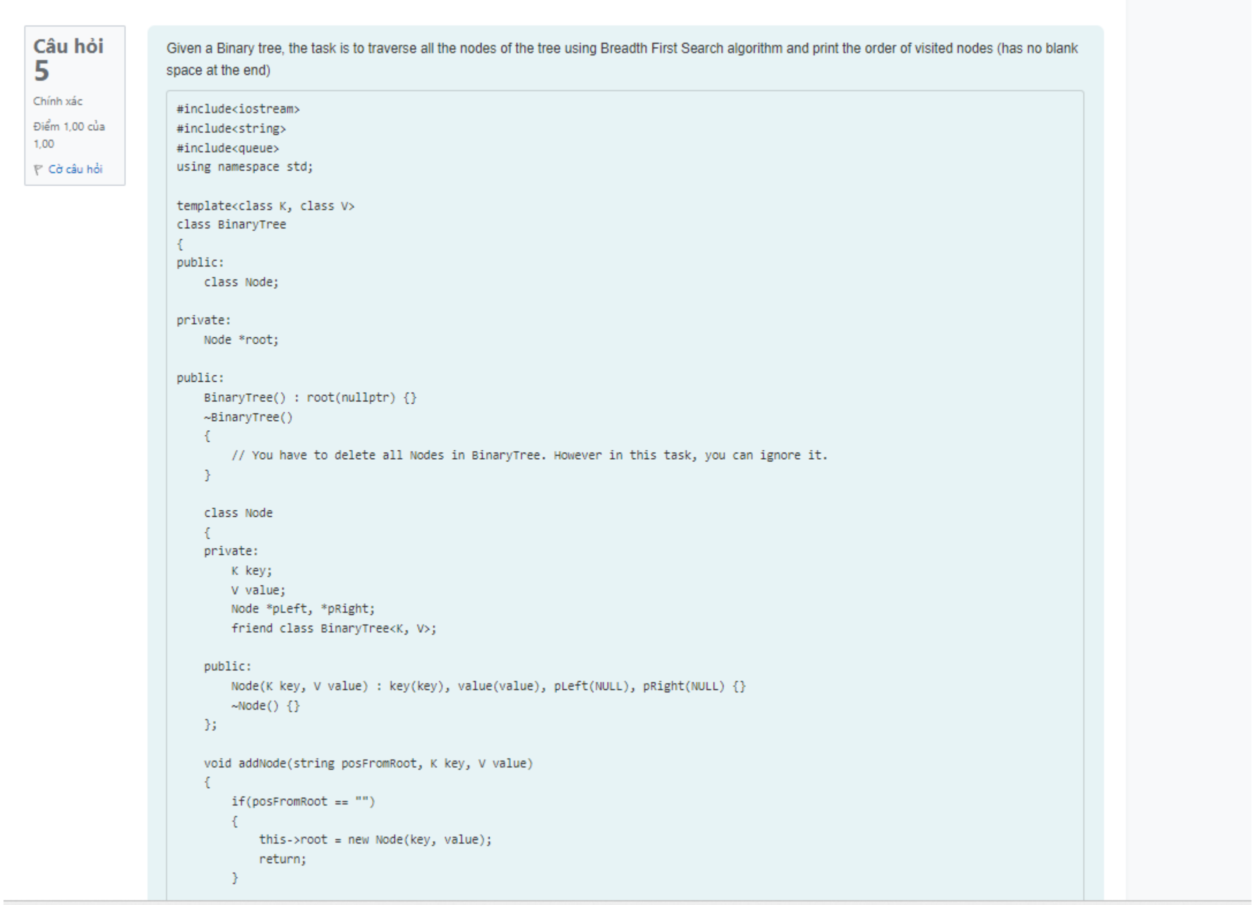
For example:

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 %)



Câu hỏi **5**

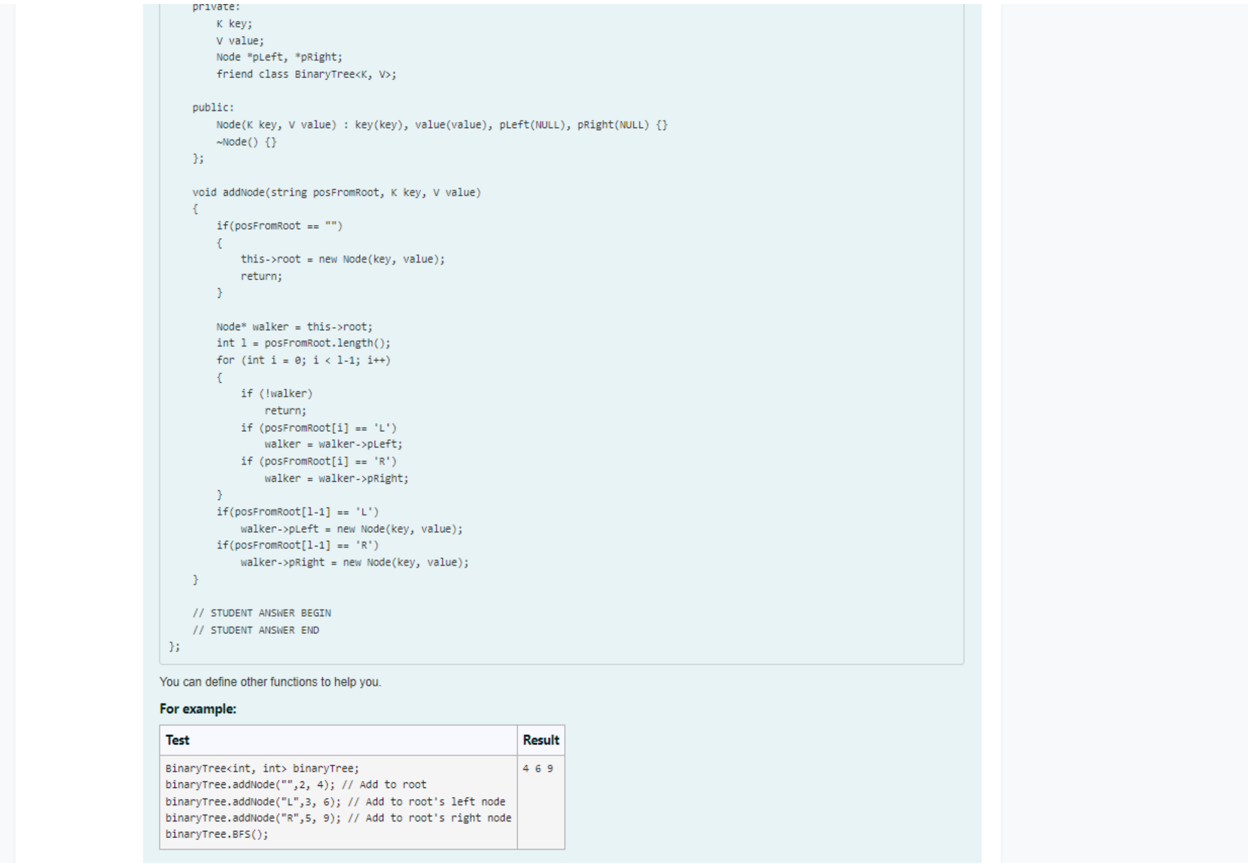
Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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;
        }
```



```
    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;
        }
```

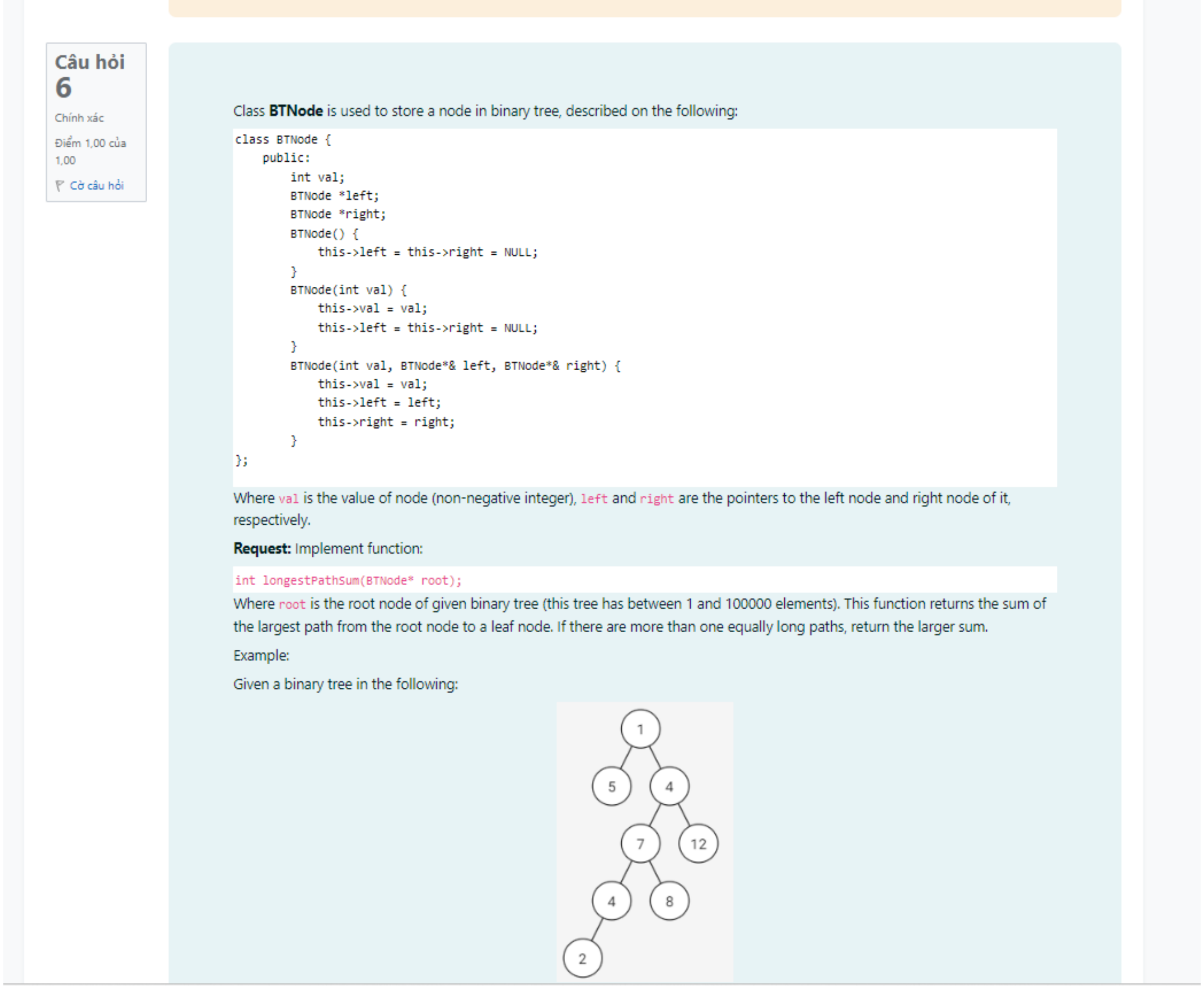
```

        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



Câu hỏi **6**

Chính xác

Điểm 1,00 của 1,00

Cờ câu hỏi

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.

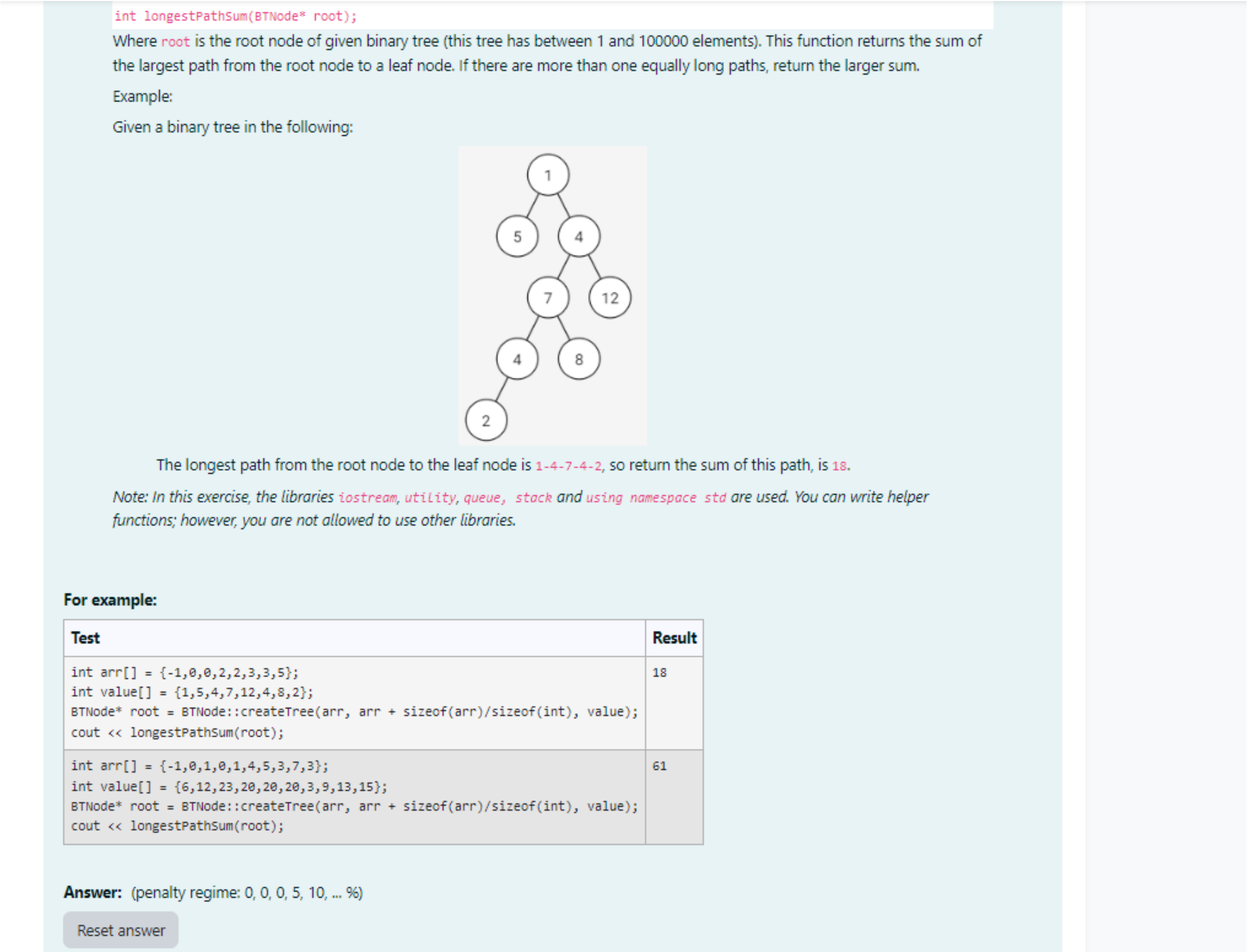
Request: Implement function:

```
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:



```
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:

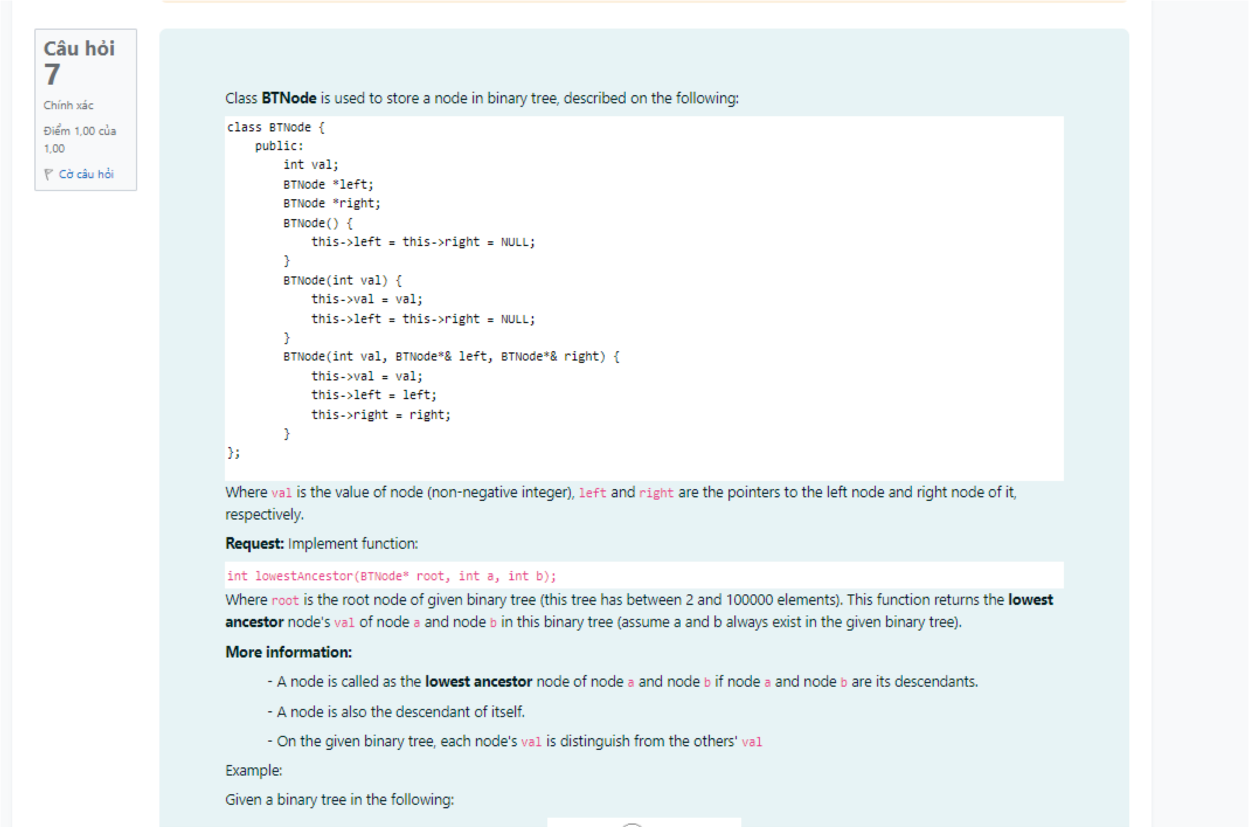
Test	Result

`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



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:

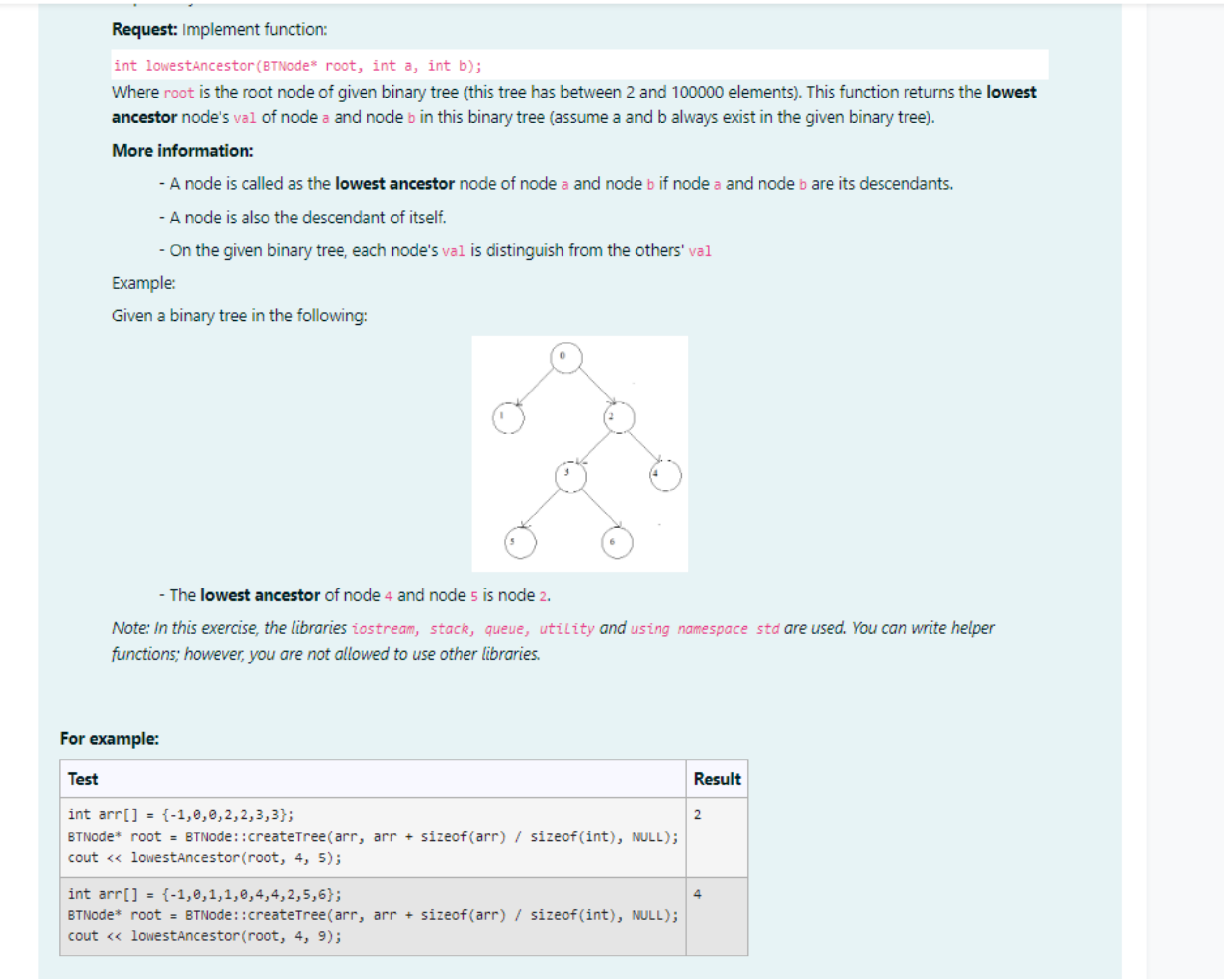
- A node is called as the **lowest ancestor** node of node `a` and node `b` if node `a` and node `b` are its descendants.

- 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:



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