Câu hỏi 1

Chính xác

Chấm điểm 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: 0 %)

Reset answer

```
void BFS()
{
    if (root == NULL)
        return;

    // Tạo một hàng đợi cho BFS
    queue<Node*> q;

    // Thêm root vào hàng đợi và khởi tạo chiều cao
    q.push(root);

    while (q.empty() == false)
    {
        // In phần tử đầu hàng đợi và loại bỏ nó khỏi hàng đợi
        Node *node = q.front();
        cout << node->value << " ";
        q.pop();

        /* Thêm con bên trái vào hàng đợi */
        if (node->pLeft != NULL)
            q.push(node->pLeft);

        /* Thêm con bên phải vào hàng đợi */
        if (node->pRight != NULL)
            q.push(node->pRight);
    }
}

```

Precheck Kiểm tra

	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! ✓

Câu hỏi 2

Chính xác

Chấm điểm 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.

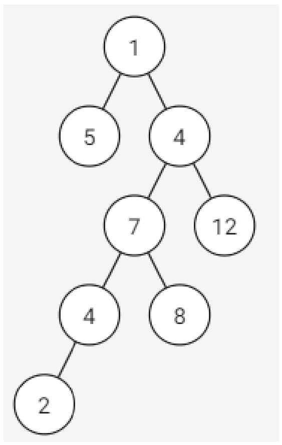
Request: Implement function:

`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`.

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

`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

```
int longestPathSum(BTNode *root)
{
    // If the tree is empty, return 0
    if (root == NULL)
    {
        return 0;
    }

    // Initialize the maximum sum and maximum depth to 0
    int maxSum = 0;
    int maxDepth = 0;

    // Create a stack to store the nodes and their current sum and depth
    std::stack<std::pair<BTNode *, std::pair<int, int>>> s;

    // Start with the root node, its value as the initial sum, and depth as 1
    s.push({root, {root->val, 1}});

    // Continue until all nodes have been processed
    while (!s.empty())
    {
        // Get the current node, sum and depth from the top of the stack
        auto node = s.top().first;
        auto sum = s.top().second.first;
        auto depth = s.top().second.second;

        // Remove the top element from the stack
        s.pop();

        // If it's a leaf node (both left and right children are NULL)
        if (node->left == NULL && node->right == NULL)
        {
            // If the current path is longer or equal but has a larger sum than the maximum found s
            if (depth > maxDepth || (depth == maxDepth && sum > maxSum))
            {
                // Update the maximum sum and depth
                maxSum = sum;
                maxDepth = depth;
            }
        }

        // If there's a left child, add it to the stack with its sum and depth
        if (node->left != NULL)
        {
            s.push({node->left, {sum + node->left->val, depth + 1}});
        }

        // If there's a right child, add it to the stack with its sum and depth
```

Precheck Kiểm tra

	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! ✓

Câu hỏi 3

Chính xác

Chấm điểm 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:

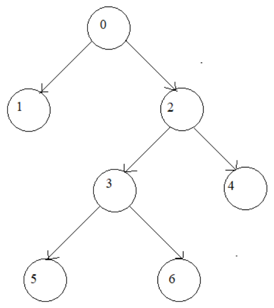
- A node is called as the **lowest ancestor** node of node `a` and node `b` if node `a` and node `b` are its descendants.

- 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`.

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

`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

```
// Function to find the lowest common ancestor of two nodes in a binary tree
int lowestAncestor(BTNode *root, int a, int b)
{
    if (root == NULL)
    { // If the root is NULL, return -1
        return -1;
    }
    if (root->val == a || root->val == b)
    { // If the root is one of the nodes we are looking for, return its value
        return root->val;
    }
    int left = lowestAncestor(root->left, a, b);   // Recursively call the function on the left child
    int right = lowestAncestor(root->right, a, b); // Recursively call the function on the right chilc

    if (left != -1 && right != -1)
    { // If both calls return a value other than -1, return the value of the root
        return root->val;
    }
    if (left != -1)
    { // If only the call on the left child node returns a value other than -1, return that value
        return left;
    }
    return right; // Otherwise, return the value returned by the call on the right child node
}
```

Precheck Kiểm tra

	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! ✓

Câu hỏi 4

Chính xác

Chấm điểm 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).

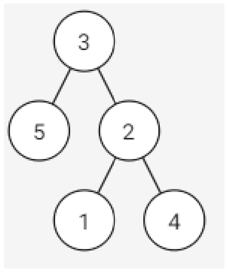
More information:

- 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`.

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
`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

```
int sumDigitPath(BTNode *root, int val = 0, int mod = 27022001)
{
    // If the node is NULL, return 0
    if (root == NULL)
        return 0;

    // Calculate the current value by shifting the previous value left by one digit (multiplying by 10
    // and adding the value of the current node. This is done modulo 'mod' to prevent overflow.
    val = ((10LL * val) + root->val) % mod;

    // If it's a leaf node (both left and right child nodes are NULL), return the current value
    if (root->left == NULL && root->right == NULL)
        return val;

    // If it's not a leaf node, recursively calculate the sum for the left and right subtrees
    // and return their sum modulo 'mod'
    return (sumDigitPath(root->left, val) + sumDigitPath(root->right, val)) % mod;
}
```

Precheck Kiểm tra

	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! ✓

Câu hỏi 5

Chính xác

Chấm điểm 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

```
// Hàm trợ giúp để đếm số nút có hai con
int countTwoChildrenNodeHelper(Node* node)
{
    // Nếu nút hiện tại là NULL, trả về 0
    if (node == NULL)
        return 0;
    int count = 0;
    // Nếu nút hiện tại có cả con trái và con phải, tăng biến đếm
    if (node->pLeft != NULL && node->pRight != NULL)
        count++;
    // Gọi đệ quy hàm cho con trái và con phải của nút hiện tại
    return count + countTwoChildrenNodeHelper(node->pLeft) + countTwoChildrenNodeHelper(node->pRight);
}

// Hàm để khởi tạo hàm trợ giúp
int countTwoChildrenNode()
{
    // Bắt đầu từ gốc của cây
    return countTwoChildrenNodeHelper(root);
}

```

Precheck Kiểm tra

	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! ✓

Câu hỏi 6

Chính xác

Chấm điểm 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

```
    stringstream ss;
    inOrder(root, ss);
    return ss.str();
}

// Hàm postOrder() duyệt cây theo thứ tự sau (trái -> phải -> gốc)
void postOrder(Node* node, stringstream& ss) {
    if (node == NULL)
        return;
    postOrder(node->pLeft, ss);
    postOrder(node->pRight, ss);
    ss << node->value << " ";
}

string postOrder() {
    stringstream ss;
    postOrder(root, ss);
    return ss.str();
}

// STUDENT ANSWER END

```

Precheck Kiểm tra

	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
			```
				✓

	Test	Expected	Got	

✓	`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`
				✓

✓	`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`
				✓

	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);`
	
	`    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
				✓

✓	`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
				✓

	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("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! ✓

Câu hỏi 7

Chính xác

Chấm điểm 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)

```
#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

```
int sumOfLeafNodes(Node* node) {
    // Base case: if node is null, return 0
    if (node == NULL)
        return 0;

    // If this node is a leaf node, return its value
    if (node->pLeft == NULL && node->pRight == NULL)
        return node->value;

    // Otherwise, return sum of values of leaf nodes in left and right subtrees
    return sumOfLeafNodes(node->pLeft) + sumOfLeafNodes(node->pRight);
}

int sumOfLeafs() {
    // Call the helper function starting from the root
    return sumOfLeafNodes(root);
}

```

Precheck Kiểm tra

	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! ✓