Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Sáu, 22 tháng 11 2024, 9:29 PM
Kết thúc lúc	Thứ Sáu, 22 tháng 11 2024, 9:31 PM
Thời gian thực hiện	2 phút 38 giây
Điểm	7,00/7,00
Điểm	**10,00** trên 10,00 (**100**%)

Câu hỏi **1**

Đúng

Đạt điểm 1,00 trên 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 countTwoChildrenNodeHelper(root);
4   }
5
6
7   int countTwoChildrenNodeHelper(Node* node)
8   {
9       if (node == nullptr)
10          return 0;
11
12      int count = 0;
13      if (node->pLeft && node->pRight)
14          count = 1;
15
16      return count + countTwoChildrenNodeHelper(node->pLeft) + countTwoChildrenNode
17  }
18
19
```

	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! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **2**

Đúng

Đạt điểm 1,00 trên 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

```
int getHeightHelper(Node* node) {
    if (node == nullptr)
        return 0;
    int leftHeight = getHeightHelper(node->pLeft);
    int rightHeight = getHeightHelper(node->pRight);
    return 1 + max(leftHeight, rightHeight);
}

int getHeight() {
    return getHeightHelper(root);
}

// Helper function for pre-order traversal
void preOrderHelper(Node* node, stringstream& ss) {
    if (node == nullptr)
        return;
    ss << node->value << " ";
    preOrderHelper(node->pLeft, ss);
    preOrderHelper(node->pRight, ss);
}

string preOrder() {
    stringstream ss;
    preOrderHelper(root, ss);
    string result = ss.str();
    if (!result.empty())
        result.pop_back(); // remove the trailing space
    return result;
}

// Helper function for in-order traversal
void inOrderHelper(Node* node, stringstream& ss) {
    if (node == nullptr)
        return;
    inOrderHelper(node->pLeft, ss);
    ss << node->value << " ";
    inOrderHelper(node->pRight, ss);
}

string inOrder() {
    stringstream ss;
    inOrderHelper(root, ss);
    string result = ss.str();
    if (!result.empty())
```

```
            result.pop_back(); // remove the trailing space
        return result;
    }

    // Helper function for post-order traversal
    void postOrderHelper(Node* node, stringstream& ss) {
        if (node == nullptr)
            return;
```

	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
				✓

✓	`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
				✓

	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("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
				✓

✓	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
				✓

	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);
	
	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! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **3**

Đúng

Đạt điểm 1,00 trên 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 sumOfLeafs() {
        return sumOfLeafsHelper(root);
    }


    int sumOfLeafsHelper(Node* node) {
        if (node == nullptr)
            return 0;

        // Check if the current node is a leaf
        if (node->pLeft == nullptr && node->pRight == nullptr)
            return node->value;

        // Recursively sum up leaf nodes in the left and right subtrees
        return sumOfLeafsHelper(node->pLeft) + sumOfLeafsHelper(node->pRight);
    }
```

	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! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **4**

Đúng

Đạt điểm 1,00 trên 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.

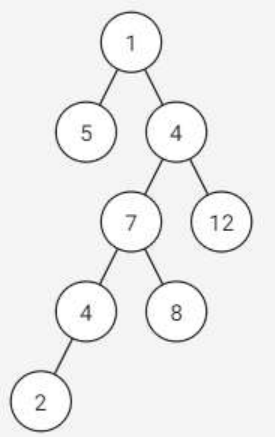
Request: Implement function:

```
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 longestPathSumHelper(BTNode* node, int& maxLength, int& maxSum, int currentLength
    if (!node) return 0;

    currentLength++;
    currentSum += node->val;

    if (!node->left && !node->right) { // Leaf node
        if (currentLength > maxLength || (currentLength == maxLength && currentSum >
            maxLength = currentLength;
            maxSum = currentSum;
        }
    } else {
        if (node->left) {
            longestPathSumHelper(node->left, maxLength, maxSum, currentLength, curren
        }
        if (node->right) {
            longestPathSumHelper(node->right, maxLength, maxSum, currentLength, curre
        }
    }
    return maxSum;
}

int longestPathSum(BTNode* root) {
    if (!root) return 0;
    int maxLength = 0;
    int maxSum = 0;
    return longestPathSumHelper(root, maxLength, maxSum, 0, 0);
}
```

	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! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **5**

Đúng

Đạt điểm 1,00 trên 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

```
// STUDENT ANSWER BEGIN
// You can define other functions here to help you.

void BFS() {
    if (this->root == nullptr) return; // If the tree is empty, there's nothing to tr

    queue<Node*> q;
    q.push(this->root); // Start from the root

    bool isFirst = true; // Used to avoid trailing spaces
    while (!q.empty()) {
        Node* current = q.front();
        q.pop();

        if (!isFirst) {
            cout << " ";
        }
        cout << current->value;
        isFirst = false;

        // Enqueue left and right children if they exist
        if (current->pLeft) q.push(current->pLeft);
        if (current->pRight) q.push(current->pRight);
    }
}
// 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! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **6**

Đúng

Đạt điểm 1,00 trên 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).

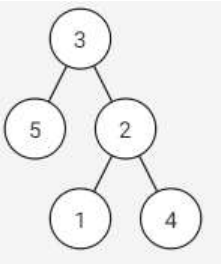
More information:

- 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

```
const int MOD = 27022001;
int sumDigitPathHelper(BTNode* node, int currentSum) {
    if (!node) return 0; // Base case: null node

    // Update the current sum to include the current node's value
    currentSum = (currentSum * 10 + node->val) % MOD;

    // If the current node is a leaf, return its path sum
    if (!node->left && !node->right) {
        return currentSum;
    }

    // Recur for left and right children
    int leftSum = sumDigitPathHelper(node->left, currentSum);
    int rightSum = sumDigitPathHelper(node->right, currentSum);

    // Combine the results from both subtrees
    return (leftSum + rightSum) % MOD;
}

int sumDigitPath(BTNode* root) {
    if (!root) return 0; // If the tree is empty, return 0
    return sumDigitPathHelper(root, 0); // Start recursion with initial sum 0
}
```

	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! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **7**

Đúng

Đạt điểm 1,00 trên 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:

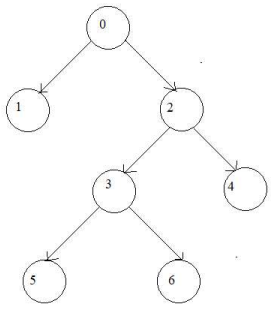
- A node is called as the **lowest ancestor** node of node `a` and node `b` if node `a` and node `b` are its descendants.

- 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

```
BTNode* findLowestAncestor(BTNode* root, int a, int b) {
    if (!root) return nullptr; // Base case: null node

    // If the current node matches either a or b, it's a potential ancestor
    if (root->val == a || root->val == b) {
        return root;
    }

    // Recur for left and right subtrees
    BTNode* leftLCA = findLowestAncestor(root->left, a, b);
    BTNode* rightLCA = findLowestAncestor(root->right, a, b);

    // If both left and right return non-null, the current node is the LCA
    if (leftLCA && rightLCA) {
        return root;
    }

    // Otherwise, return the non-null result from the subtrees
    return leftLCA ? leftLCA : rightLCA;
}

int lowestAncestor(BTNode* root, int a, int b) {
    BTNode* lcaNode = findLowestAncestor(root, a, b);
    return lcaNode ? lcaNode->val : -1; // Return the value of the LCA node
}
```

	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! ✓

Đúng

Marks for this submission: 1,00/1,00.