Bắt đầu vào lúc	Thứ Tư, 20 tháng 3 2024, 3:23 PM
Trạng thái	Đã xong
Kết thúc lúc	Thứ Ba, 2 tháng 4 2024, 7:45 PM
Thời gian thực hiện	13 Các ngày 4 giờ
Điểm	6,50/7,00
Điểm	**9,29** trên 10,00 (**92,86**%)

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

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

  int count(Node* node)
    {
        if (node == NULL) return 0;
        else if (node->pLeft != NULL && node->pRight != NULL) return count(node->pLeft) + count(node->pR
        else return count(node->pLeft) + count(node->pRight);
    }

    int countTwoChildrenNode()
    {
        return count(root);
    }
// STUDENT ANSWER END
```

	Test	Expected	Got	

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

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

Passed all tests! ✓

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

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

int Height(Node* root) {
       if (root == NULL)
           return 0;
       else
       {
           int l = Height(root->pLeft);
           int r = Height(root->pRight);
           return max(l,r) + 1;
       }
   }

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


   void printInorder(Node* node)
   {
       if (node == NULL)
           return;
       printInorder(node->pLeft);
       cout << node->value << " ";
       printInorder(node->pRight);
   }

   void printPreorder(Node* node)
   {
       if (node == NULL)
           return;
       cout << node->value << " ";
       printPreorder(node->pLeft);
       printPreorder(node->pRight);
   }

   void printPostorder(Node* node)
   {
       if (node == NULL)
           return;
       printPostorder(node->pLeft);
       printPostorder(node->pRight);
       cout << node->value << " ";
```

```
45      }
46
47    string preOrder() {
48          printPreorder(root);
49          return "";
50      }
51
52      string inOrder() {
```

	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

```
//Helping functions
int count(Node* node)
   {
       if (node == NULL) return 0;
       else if (node->pLeft == NULL && node->pRight == NULL) return node->value + count(node->pLeft) +
       else return count(node->pLeft) + count(node->pRight);
   }

   int sumOfLeafs() {
       return count(root);
   }
```

	Test	Expected	Got	

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

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

Passed all tests! ✓

Đúng

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

Câu hỏi **4**

Đúng một phần

Đạt điểm 0,50 trên 1,00

Class **BTNode** is used to store a node in binary tree, described on the following:

Explain

```
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.

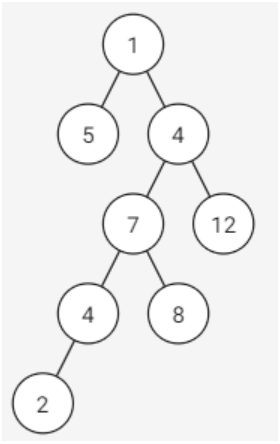
Request: Implement function:

```
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 (root == NULL) {
            return 0;
        }

        int maxSum = 0;
        stack<pair<BTNode*, int>> nodes;
        nodes.push(make_pair(root, root->val));

        while (!nodes.empty()) {
            pair<BTNode*, int> current = nodes.top();
            nodes.pop();
            BTNode* currentNode = current.first;
            int currentSum = current.second;

            if (currentNode->left == NULL && currentNode->right == NULL) {
                maxSum = max(maxSum, currentSum);
            }

            if (currentNode->right != NULL) {
                nodes.push(make_pair(currentNode->right, currentSum + currentNode->right->val));
            }

            if (currentNode->left != NULL) {
                nodes.push(make_pair(currentNode->left, currentSum + currentNode->left->val));
            }
        }

        return maxSum;
}
```

	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	20	✗

Some hidden test cases failed, too.

Show differences

Đúng một phần

Marks for this submission: 0,50/1,00.

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 breadthFirstSearch(Node *root) {
        if (root == NULL) return;
        queue<Node*> bf_queue;
        bf_queue.push(root);
        while(!bf_queue.empty()) {
            Node* current = bf_queue.front();
            bf_queue.pop();
            cout<<current->value<<" ";
            if(current->pLeft != NULL) {
                bf_queue.push(current->pLeft);
            }
            if(current->pRight != NULL) {
                bf_queue.push(current->pRight);
            }
        }
    }

    void BFS()
    {
        breadthFirstSearch(root);
    }
// STUDENT ANSWER END
```

	Test	Expected	Got	

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

Explain

```
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).

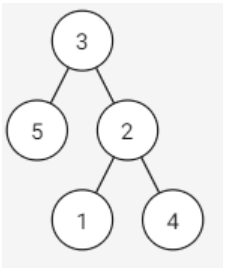
More information:

- 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 sumDigitPathUtil(BTNode* root, int pathSum) {
    if (!root) return 0;

    pathSum = (pathSum * 10 + root->val) % MOD;

    // If it's a leaf node, return the path sum
    if (!root->left && !root->right) {
        return pathSum;
    }

    // Recursively calculate the sum of digit paths in the left and right subtrees
    int leftSum = sumDigitPathUtil(root->left, pathSum);
    int rightSum = sumDigitPathUtil(root->right, pathSum);

    // Return the sum of digit paths from both subtrees
    return (leftSum + rightSum) % MOD;
}

int sumDigitPath(BTNode* root) {
    // Start DFS traversal from the root node with initial path sum 0
    int totalSum = sumDigitPathUtil(root, 0);
    return totalSum;
}
```

	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:

Explain

```
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:

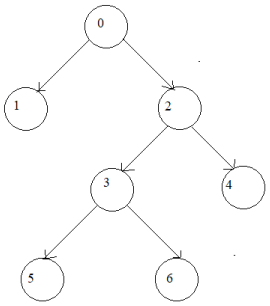
- A node is called as the **lowest ancestor** node of node `a` and node `b` if node `a` and node `b` are its descendants.

- 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

```
bool findPath(BTNode* root, vector<int>& path, int k)
{
    // base case
    if (root == NULL)
        return false;

    // Store this node in path vector. The node will be
    // removed if not in path from root to k
    path.push_back(root->val);

    // See if the k is same as root's key
    if (root->val == k)
        return true;

    // Check if k is found in left or right sub-tree
    if ((root->left && findPath(root->left, path, k))
        || (root->right && findPath(root->right, path, k)))
        return true;

    // If not present in subtree rooted with root, remove
    // root from path[] and return false
    path.pop_back();
    return false;
}
int lowestAncestor(BTNode* root, int n1, int n2) {
     vector<int> path1, path2;
```

```
    // Find paths from root to n1 and root to n2. If either
    // n1 or n2 is not present, return -1
    if (!findPath(root, path1, n1)
        || !findPath(root, path2, n2))
        return -1;

    /* Compare the paths to get the first different value */
    long unsigned int i;
    for (i = 0; i < path1.size() && i < path2.size(); i++)
        if (path1[i] != path2[i])
            break;
    return path1[i - 1];
}
```

	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.