

Thời gian còn lại 0:07:20

Câu hỏi 7

Chính xác

Điểm 1,00 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
<pre>BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); // Add to root binaryTree.addNode("L", 3, 6); // Add to root's left node binaryTree.addNode("R", 5, 9); // Add to root's right node binaryTree.BFS();</pre>	4 6 9

Answer: (penalty regime: 0 %)

Reset answer

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

Precheck

Kiểm tra

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

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 8

Chính xác

Điểm 1,00 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.

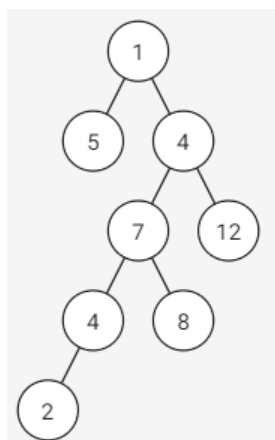
Request: Implement function:

```
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.

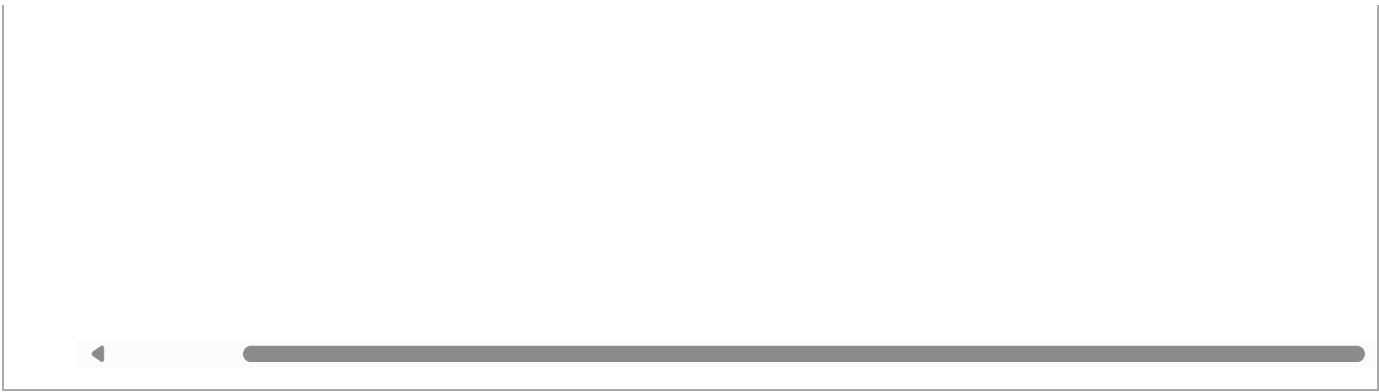
For example:

Test	Result
<pre>int arr[] = {-1,0,0,2,2,3,3,5}; int value[] = {1,5,4,7,12,4,8,2}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << longestPathSum(root);</pre>	18
<pre>int arr[] = {-1,0,1,0,1,4,5,3,7,3}; int value[] = {6,12,23,20,20,20,3,9,13,15}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << longestPathSum(root);</pre>	61

Answer: (penalty regime: 0 %)

Reset answer

```
1 PathSumUtil(BTNode* root, int& maxPathSum, int& maxPathLength, int currentPathSum, int currentPathLength)
2 {
3     return 0;
4
5     PathSum += root->val;
6     PathLength++;
7     if (root->left && !(root->right)) {
8         if (currentPathLength > maxPathLength || (currentPathLength == maxPathLength && currentPathSum > maxPathSum))
9             maxPathLength = currentPathLength;
10            maxPathSum = currentPathSum;
11
12
13     PathSumUtil(root->left, maxPathSum, maxPathLength, currentPathSum, currentPathLength);
14     PathSumUtil(root->right, maxPathSum, maxPathLength, currentPathSum, currentPathLength);
15
16     maxPathSum;
17
18
19 PathSum(BTNode* root) {
20     PathSum = 0;
21     PathLength = 0;
22     PathSumUtil(root, maxPathSum, maxPathLength, 0, 0);
23     maxPathSum;
24 }
```



Precheck

Kiểm tra

	Test	Expected	Got	
✓	<pre>int arr[] = {-1,0,0,2,2,3,3,5}; int value[] = {1,5,4,7,12,4,8,2}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << longestPathSum(root);</pre>	18	18	✓
✓	<pre>int arr[] = {-1,0,1,0,1,4,5,3,7,3}; int value[] = {6,12,23,20,20,20,3,9,13,15}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << longestPathSum(root);</pre>	61	61	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 9

Chính xác

Điểm 1,00 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);
```

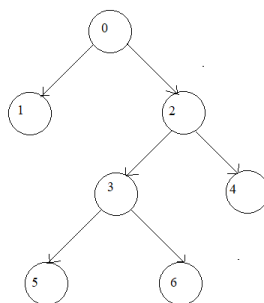
Where **root** is the root node of given binary tree (this tree has between 2 and 100000 elements). This function returns the **lowest ancestor** node's **val** of node **a** and node **b** in this binary tree (assume **a** and **b** always exist in the given binary tree).

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
<pre>int arr[] = {-1,0,0,2,2,3,3}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL); cout << lowestAncestor(root, 4, 5);</pre>	2
<pre>int arr[] = {-1,0,1,1,0,4,4,2,5,6}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL); cout << lowestAncestor(root, 4, 9);</pre>	4

Answer: (penalty regime: 0 %)

Reset answer

```
1 int lowestAncestor(BTNode* root, int a, int b) {
2     if (!root) return -1;
3     if (root->val == a || root->val == b)
4         return root->val;
5     int leftAncestor = lowestAncestor(root->left, a, b);
6     int rightAncestor = lowestAncestor(root->right, a, b);
7     if (leftAncestor != -1 && rightAncestor != -1)
8         return root->val;
9     return (leftAncestor != -1) ? leftAncestor : rightAncestor;
10 }
```

Precheck

Kiểm tra

	Test	Expected	Got	
✓	<pre>int arr[] = {-1,0,0,2,2,3,3}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL); cout << lowestAncestor(root, 4, 5);</pre>	2	2	✓
✓	<pre>int arr[] = {-1,0,1,1,0,4,4,2,5,6}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr) / sizeof(int), NULL); cout << lowestAncestor(root, 4, 9);</pre>	4	4	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 10

Chính xác

Điểm 1,00 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);
```

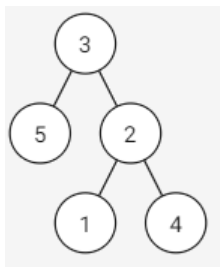
Where **root** is the root node of given binary tree (this tree has between 2 and 100000 elements). This function returns the sum of all **digit path** numbers of this binary tree (the result may be large, so you must use **mod 27022001** before returning).

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 represented by them are 35, 321, 324, respectively. The sum of them (after **mod 27022001**) is 680.

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

Example:

```
int arr[] = {-1,0,0,2,2};
int value[] = {3,5,2,1,4};
BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value);
```

`arr[0]=-1` means the Node containing the value `value[0]=3` will be the root Node. Also, since `arr[1]=arr[2]=0`, it implies that the Nodes containing the values `value[1]=5` and `value[2]=2` will have the Node containing the value `value[0]=3` as their parent. Lastly, since `arr[3]=arr[4]=2`, it means the Nodes containing the values `value[3]=1` and `value[4]=4` will have the Node with the value `value[2]=2` as their parent. Final tree of this example are shown in the figure above.

Note that whichever Node appears first in the `arr` sequence will be the left Node, and the TestCase always ensures that the resulting tree is a binary tree.

Note: In this exercise, the libraries `iostream`, `queue`, `stack`, `utility` and `using namespace std` are used. You can write helper functions; however, you are not allowed to use other libraries.

For example:

Test	Result
<pre>int arr[] = {-1,0,0,2,2}; int value[] = {3,5,2,1,4}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << sumDigitPath(root);</pre>	680
<pre>int arr[] = {-1,0,0}; int value[] = {1,2,3}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << sumDigitPath(root);</pre>	25

Answer: (penalty regime: 0 %)

Reset answer

```
1 |const int MOD = 27022001;
2 |int sumDigitPathUlti(BTNode* root, int currNum) {
3 |    if (!root) return 0;
4 |    currNum = (currNum * 10 + root->val) % MOD;
5 |    if (!root->left && !root->right)
6 |        return currNum;
7 |    int leftSum = sumDigitPathUlti(root->left, currNum) % MOD;
8 |    int rightSum = sumDigitPathUlti(root->right, currNum) % MOD;
9 |    return (leftSum+rightSum) % MOD;
10|}
11|int sumDigitPath(BTNode* root) {
12|    if (!root) return 0;
13|    int result = sumDigitPathUlti(root, 0);
14|    return result;
15|}
```

Precheck

Kiểm tra

	Test	Expected	Got	
✓	<pre>int arr[] = {-1,0,0,2,2}; int value[] = {3,5,2,1,4}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << sumDigitPath(root);</pre>	680	680	✓
✓	<pre>int arr[] = {-1,0,0}; int value[] = {1,2,3}; BTNode* root = BTNode::createTree(arr, arr + sizeof(arr)/sizeof(int), value); cout << sumDigitPath(root);</pre>	25	25	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 11

Chính xác

Điểm 1,00 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;	2
binaryTree.addNode("", 2, 4); // Add to root	4 6 9
binaryTree.addNode("L", 3, 6); // Add to root's left node	6 4 9
binaryTree.addNode("R", 5, 9); // Add to root's right node	6 9 4
cout << binaryTree.getHeight() << endl;	
cout << binaryTree.preOrder() << endl;	
cout << binaryTree.inOrder() << endl;	
cout << binaryTree.postOrder() << endl;	

Answer: (penalty regime: 0 %)

Reset answer

```
1 // STUDENT ANSWER BEGIN
2 // You can define other functions here to help you.
3 int getHeightHelper(Node* node) {
4     if (node == nullptr) {
5         return 0;
6     }
7     int leftHeight = getHeightHelper(node->pLeft);
8     int rightHeight = getHeightHelper(node->pRight);
9     return 1 + max(leftHeight, rightHeight);
10 }
11 int getHeight() {
12     // TODO: return height of the binary tree.
13     return getHeightHelper(root);
14 }
15 void preOrderHelper(Node* node, stringstream& ss) {
16     if (node) {
17         ss << node->value << " ";
18         preOrderHelper(node->pLeft, ss);
19         preOrderHelper(node->pRight, ss);
20     }
21 }
22 string preOrder() {
```

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	
✓	<pre> BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl; </pre>	<pre> 1 4 4 4 </pre>	<pre> 1 4 4 4 </pre>	✓
✓	<pre> BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); binaryTree.addNode("L", 3, 6); binaryTree.addNode("R", 5, 9); binaryTree.addNode("LL", 4, 10); binaryTree.addNode("LR", 6, 2); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl; </pre>	<pre> 3 4 6 10 2 9 10 6 2 4 9 10 2 6 9 4 </pre>	<pre> 3 4 6 10 2 9 10 6 2 4 9 10 2 6 9 4 </pre>	✓
✓	<pre> BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); binaryTree.addNode("L", 3, 6); binaryTree.addNode("R", 5, 9); binaryTree.addNode("LL", 4, 10); binaryTree.addNode("RL", 6, 2); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl; </pre>	<pre> 3 4 6 10 9 2 10 6 4 2 9 10 6 2 9 4 </pre>	<pre> 3 4 6 10 9 2 10 6 4 2 9 10 6 2 9 4 </pre>	✓
✓	<pre> BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); binaryTree.addNode("L", 3, 6); binaryTree.addNode("R", 5, 9); binaryTree.addNode("LL", 4, 10); binaryTree.addNode("LLL", 6, 2); binaryTree.addNode("LLLR", 7, 7); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl; </pre>	<pre> 5 4 6 10 2 7 9 2 7 10 6 4 9 7 2 10 6 9 4 </pre>	<pre> 5 4 6 10 2 7 9 2 7 10 6 4 9 7 2 10 6 9 4 </pre>	✓
✓	<pre> BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); binaryTree.addNode("L", 3, 6); binaryTree.addNode("R", 5, 9); binaryTree.addNode("LL", 4, 10); binaryTree.addNode("LLL", 6, 2); binaryTree.addNode("LLLR", 7, 7); binaryTree.addNode("RR", 8, 30); binaryTree.addNode("RL", 9, 307); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl; </pre>	<pre> 5 4 6 10 2 7 9 307 30 2 7 10 6 4 307 9 30 7 2 10 6 307 30 9 4 </pre>	<pre> 5 4 6 10 2 7 9 307 30 2 7 10 6 4 307 9 30 7 2 10 6 307 30 9 4 </pre>	✓

	Test	Expected	Got	
✓	<pre> BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); binaryTree.addNode("L", 3, 6); binaryTree.addNode("R", 5, 9); binaryTree.addNode("LL", 4, 10); binaryTree.addNode("LR", 6, -3); binaryTree.addNode("LLL", 7, 2); binaryTree.addNode("LLLR", 8, 7); binaryTree.addNode("RR", 9, 30); binaryTree.addNode("RL", 10, 307); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl; </pre>	<pre> 5 4 6 10 2 7 -3 9 307 30 2 7 10 6 -3 4 307 9 30 7 2 10 -3 6 307 30 9 4 </pre>	<pre> 5 4 6 10 2 7 -3 9 307 30 2 7 10 6 -3 4 307 9 30 7 2 10 -3 6 307 30 9 4 </pre>	✓
✓	<pre> BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); binaryTree.addNode("L", 3, 6); binaryTree.addNode("R", 5, 9); binaryTree.addNode("LL", 4, 10); binaryTree.addNode("LR", 6, -3); binaryTree.addNode("LLL", 7, 2); binaryTree.addNode("LLLR", 8, 7); binaryTree.addNode("RR", 9, 30); binaryTree.addNode("RL", 10, 307); binaryTree.addNode("RLL", 11, 2000); binaryTree.addNode("RLR", 12, 2000); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl; </pre>	<pre> 5 4 6 10 2 7 -3 9 307 2000 2000 30 2 7 10 6 -3 4 2000 307 2000 9 30 7 2 10 -3 6 2000 2000 307 30 9 4 </pre>	<pre> 5 4 6 10 2 7 -3 9 307 2000 2000 30 2 7 10 6 -3 4 2000 307 2000 9 30 7 2 10 -3 6 2000 2000 307 30 9 4 </pre>	✓
✓	<pre> BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); binaryTree.addNode("L", 3, 6); binaryTree.addNode("R", 5, 9); binaryTree.addNode("LL", 4, 10); binaryTree.addNode("LR", 6, -3); binaryTree.addNode("LLL", 7, 2); binaryTree.addNode("LLLR", 8, 7); binaryTree.addNode("RR", 9, 30); binaryTree.addNode("RL", 10, 307); binaryTree.addNode("RLL", 11, 2000); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl; </pre>	<pre> 5 4 6 10 2 7 -3 9 307 2000 30 2 7 10 6 -3 4 2000 307 9 30 7 2 10 -3 6 2000 307 30 9 4 </pre>	<pre> 5 4 6 10 2 7 -3 9 307 2000 30 2 7 10 6 -3 4 2000 307 9 30 7 2 10 -3 6 2000 307 30 9 4 </pre>	✓

	Test	Expected	Got	
✓	<pre> BinaryTree<int, int> binaryTree; binaryTree.addNode("", 2, 4); binaryTree.addNode("L", 3, 6); binaryTree.addNode("R", 5, 9); binaryTree.addNode("LL", 4, 10); binaryTree.addNode("LR", 6, -3); binaryTree.addNode("LLL", 7, 2); binaryTree.addNode("LLLR", 8, 7); binaryTree.addNode("RR", 9, 30); binaryTree.addNode("RL", 10, 307); binaryTree.addNode("RLL", 11, 2000); binaryTree.addNode("RLLL", 11, 2000); cout << binaryTree.getHeight() << endl; cout << binaryTree.preOrder() << endl; cout << binaryTree.inOrder() << endl; cout << binaryTree.postOrder() << endl; </pre>	<pre> 5 4 6 10 2 7 -3 9 307 2000 2000 30 2 7 10 6 -3 4 2000 2000 307 9 30 7 2 10 -3 6 2000 2000 307 30 9 4 </pre>	<pre> 5 4 6 10 2 7 -3 9 307 2000 2000 30 2 7 10 6 -3 4 2000 2000 307 9 30 7 2 10 -3 6 2000 2000 307 30 9 4 </pre>	✓

Passed all tests! ✓

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

BÁCH KHOA E-LEARNING



WEBSITE

HCMUT

MyBK

BKSI

LIÊN HỆ

📍 268 Lý Thường Kiệt, P.14, Q.10, TP.HCM

☎ (028) 38 651 670 - (028) 38 647 256 (Ext: 5258, 5234)

✉ elearning@hcmut.edu.vn