Câu hỏi 1

Chính xác

Chấm điểm của 2,00

In this question, you have to perform **rotate nodes** on AVL tree. Note that:

- When adding a node which has the same value as parent node, add it in the **right sub tree**.

Your task is to implement function: **rotateRight, rotateLeft**. You could define one or more functions to achieve this task.

```
#include <iostream>
#include <math.h>
#include <queue>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"

enum BalanceValue
{
    LH = -1,
    EH = 0,
    RH = 1
};

void printNSpace(int n)
{
    for (int i = 0; i < n - 1; i++)
        cout << " ";
}

void printInteger(int &n)
{
    cout << n << " ";
}

template<class T>
class AVLTree
{
public:
    class Node;
private:
    Node *root;
protected:
    int getHeightRec(Node *node)
    {
        if (node == NULL)
            return 0;
        int lh = this->getHeightRec(node->pLeft);
        int rh = this->getHeightRec(node->pRight);
        return (lh > rh ? lh : rh) + 1;
    }
public:
    AVLTree() : root(nullptr) {}
    ~AVLTree(){}
    int getHeight()
    {
        return this->getHeightRec(this->root);
    }
    void printTreeStructure()
    {
        int height = this->getHeight();
        if (this->root == NULL)
        {
            cout << "NULL\n";
            return;
        }
        queue<Node *> q;
        q.push(root);
        Node *temp;
        int count = 0;
        int maxNode = 1;
        int level = 0;
        int space = pow(2, height);
        printNSpace(space / 2);
        while (!q.empty())
        {
            temp = q.front();
            q.pop();
            if (temp == NULL)
            {
                cout << " ";
                q.push(NULL);
                q.push(NULL);
            }
            else
            {
                cout << temp->data;
                q.push(temp->pLeft);
                q.push(temp->pRight);
            }
            printNSpace(space);
            count++;
            if (count == maxNode)
            {
                cout << endl;
                count = 0;
                maxNode *= 2;
                level++;
                space /= 2;
                printNSpace(space / 2);
            }
            if (level == height)
                return;
        }
    }

    void insert(const T &value);
```

```
int getBalance(Node*subroot){
    if(!subroot) return 0;
    return getHeightRec(subroot->pLeft)- getHeightRec(subroot->pRight);
}
```

```
Node* rotateLeft(Node* subroot)
```

```
{
```

```
//TODO: Rotate and return new root after rotate
```

```
};
```

```
Node* rotateRight(Node* subroot)
```

```
{
```

```
//TODO: Rotate and return new root after rotate
```

```
};
```

```

    class Node
    {
    private:
        T data;
        Node *pLeft, *pRight;
        BalanceValue balance;
        friend class AVLTree<T>;

    public:
        Node(T value) : data(value), pLeft(NULL), pRight(NULL), balance(EH) {}
        ~Node() {}
    };
};
```

For example:

Test

```
// Test rotateLeft
AVLTree<int> avl;
avl.insert(0);
avl.insert(1);
cout << "After inserting 0, 1. Tree:" << endl;
avl.printTreeStructure();
avl.insert(2);
cout << endl << "After inserting 2, perform 'rotateLeft'. Tree:" << endl;
avl.printTreeStructure();
```

Result

```
After inserting 0, 1. Tree:
 0
  1

After inserting 2, perform 'rotateLeft'. Tree:
 1
0 2
```

Test

```
// Test rotateRight
AVLTree<int> avl;
avl.insert(10);
avl.insert(9);
cout << "After inserting 10, 9. Tree:" << endl;
avl.printTreeStructure();
avl.insert(8);
cout << endl << "After inserting 8, perform 'rotateRight'. Tree:" << endl;
avl.printTreeStructure();
```

Result

```
After inserting 10, 9. Tree:
 10
9

After inserting 8, perform 'rotateRight'. Tree:
 9
8 10
```

Answer: (penalty regime: 0 %)

Reset answer

```
Node* rotateLeft(Node* root)
{
    Node* newRoot = root->pRight;
    Node* transferNode = newRoot->pLeft;

    newRoot->pLeft = root;
    root->pRight = transferNode;

    root->balance = (getHeightRec(root->pLeft) >= getHeightRec(root->pRight)) ? LH : EH;
    newRoot->balance = (getHeightRec(newRoot->pLeft) <= getHeightRec(newRoot->pRight)) ? RH : EH;

    return newRoot;
};

Node* rotateRight(Node* root)
{
    Node* newRoot = root->pLeft;
    Node* transferNode = newRoot->pRight;

    newRoot->pRight = root;
    root->pLeft = transferNode;

    // Update balance
    root->balance = (getHeightRec(root->pLeft) >= getHeightRec(root->pRight)) ? LH : EH;
    newRoot->balance = (getHeightRec(newRoot->pLeft) <= getHeightRec(newRoot->pRight)) ? RH : EH;

    return newRoot;
};
```

Precheck Kiểm tra

	Test	Expected	Got	
✓	`// Test rotateLeft`	`After inserting 0, 1. Tree:`	`After inserting 0, 1. Tree:`	✓
	`AVLTree<int> avl;`	` 0`	` 0`	
	`avl.insert(0);`	`  1`	`  1`	
	`avl.insert(1);`			
	`cout << "After inserting 0, 1. Tree:" << endl;`	`After inserting 2, perform`	`After inserting 2, perform`	
	`avl.printTreeStructure();`	`'rotateLeft'. Tree:`	`'rotateLeft'. Tree:`	
	`avl.insert(2);`	` 1`	` 1`	
	`cout << endl << "After inserting 2, perform`	`0 2`	`0 2`	
	`'rotateLeft'. Tree:" << endl;`			
	`avl.printTreeStructure();`			
✓	`// Test rotateRight`	`After inserting 10, 9. Tree:`	`After inserting 10, 9. Tree:`	✓
	`AVLTree<int> avl;`	` 10`	` 10`	
	`avl.insert(10);`	`9`	`9`	
	`avl.insert(9);`			
	`cout << "After inserting 10, 9. Tree:" << endl;`	`After inserting 8, perform`	`After inserting 8, perform`	
	`avl.printTreeStructure();`	`'rotateRight'. Tree:`	`'rotateRight'. Tree:`	
	`avl.insert(8);`	` 9`	` 9`	
	`cout << endl << "After inserting 8, perform`	`8 10`	`8 10`	
	`'rotateRight'. Tree:" << endl;`			
	`avl.printTreeStructure();`			

Passed all tests! ✓