Câu hỏi 3

Chính xác

Chấm điểm của 2,00

In this question, you have to perform add 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: `insert`. The function should cover at least these cases:

- Balanced tree
- Right of right unbalanced tree
- Left of right unbalanced tree

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)
    {
        //TODO
    }

    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	Result

`AVLTree<int> avl;`
`int nums[] = {3, 1, 6, 2, 4, 8, 5, 7, 9};`
`for (int i = 0; i < 9; i++){`
`  avl.insert(nums[i]);`
`}`
`avl.printTreeStructure();`
	`  3`
	`1  6`
	`2 4 8`
	`5 7 9`

`AVLTree<int> avl;`
`int nums[] = {6, 8, 3, 5, 7, 9, 1, 2, 4};`
`for (int i = 0; i < 9; i++){`
`  avl.insert(nums[i]);`
`}`
`avl.printTreeStructure();`
	`  6`
	`3  8`
	`1 5 7 9`
	`2 4`

Answer: (penalty regime: 0 %)

Reset answer

```
int BalanceFactor(Node* pNode){
        if(pNode==NULL) return 0;
        return getHeightRec(pNode->pRight) - getHeightRec(pNode->pLeft);
    }
    Node* LLRotation(Node* pNode){
        Node* plNode = pNode->pLeft;
        Node* plrNode = plNode->pRight;

        plNode->pRight = pNode;
        pNode->pLeft = plrNode;

        if(root == pNode){
            root = plNode;
        }

        return plNode;
    }

    Node* RRRotation(Node* pNode){
        Node* prNode = pNode->pRight;
        Node* prlNode = prNode->pLeft;

        prNode->pLeft = pNode;
        pNode->pRight = prlNode;

        if(root == pNode){
            root = prNode;
        }

        return prNode;
    }
    Node* LRRotation(Node* pNode){
        pNode->pLeft = RRRotation(pNode->pLeft);
        return LLRotation(pNode);
    }
    Node* RLRotation(Node* pNode){
        pNode->pRight = LLRotation(pNode->pRight);
        return RRRotation(pNode);
    }

    Node* insertNode(Node* pNode, T key){
        if(!pNode){
            Node* newNode = new Node(key);
            return newNode;
        }

        if(key < pNode->data){
            pNode->pLeft = insertNode(pNode->pLeft, key);
        }
        else if(key >= pNode->data){
            pNode->pRight = insertNode(pNode->pRight, key);
        }

        int bf = BalanceFactor(pNode);

        //LL Rotation
        if(bf < LH && key < pNode->pLeft->data){
            return LLRotation(pNode);
        }
        //RR Rotation
        if(bf > RH && key >= pNode->pRight->data){
            return RRRotation(pNode);
        }
        //LR Rotation
        if(bf < LH && key >= pNode->pLeft->data){
            return LRRotation(pNode);
        }
        //RL Rotation
        if(bf > RH && key < pNode->pRight->data){
            return RLRotation(pNode);
        }
```

```
        return pNode;
    }

    void insert(const T &value){
        this->root = insertNode(this->root, value);
    }
    ////////////////////////////////////
    /////HELPING FUNCTIONS DELETION
    Node* minValueNode(Node* pNode){
        Node* curr = pNode;
        while (curr && curr->pLeft != NULL){
            curr = curr->pLeft;
        }
        return curr;
    }
```

Precheck | Kiểm tra

	Test	Expected	Got	

✔	AVLTree<int> avl;
	int nums[] = {3, 1, 6, 2, 4, 8, 5, 7, 9};
	for (int i = 0; i < 9; i++){
	avl.insert(nums[i]);
	}
	avl.printTreeStructure();
		3
		1 6
		2 4 8
		5 7 9
			3
			1 6
			2 4 8
			5 7 9
				✔

✔	AVLTree<int> avl;
	int nums[] = {6, 8, 3, 5, 7, 9, 1, 2, 4};
	for (int i = 0; i < 9; i++){
	avl.insert(nums[i]);
	}
	avl.printTreeStructure();
		6
		3 8
		1 5 7 9
		2 4
			6
			3 8
			1 5 7 9
			2 4
				✔

Passed all tests! ✔

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