Câu hỏi 4

Không chính xác

Chấm điểm của 2,00

In this question, you have to perform **delete in AVL tree - balanced, L-L, R-L, E-L**. Note that:

- Provided **insert** function already.

Your task is to implement function: **remove** to perform re-balancing (balanced, left of left, right of left, equal of left). 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 remove(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 arr[] = {10, 5, 15, 7};`
`for (int i = 0; i < 4; i++)`
`{`
`  avl.insert(arr[i]);`
`}`
`avl.remove(15);`
`avl.printTreeStructure();`
	7
	5 10

Test	Result

AVLTree<int> avl;
int arr[] = {10, 5, 15, 3};
for (int i = 0; i < 4; i++)
{
  avl.insert(arr[i]);
}
avl.remove(15);
avl.printTreeStructure();
	5
	3 10

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;
    }

    ////////////////////////////////
    /////HELPING FUNCTIONS DELETION
    Node* minValueNode(Node* pNode){
        Node* curr = pNode;
        while (curr && curr->pLeft != NULL){
            curr = curr->pLeft;
        }
        return curr;
    }

Node* maxValueNode(Node* pNode){
    Node* curr = pNode;
    while (curr && curr->pRight != NULL){
        curr = curr->pRight;
    }
    return curr;
}

Node* deleteNodeRec(Node* pNode, T key){
    if(!pNode){
        return pNode;
    }

    if(pNode->data < key){
        pNode->pRight = deleteNodeRec(pNode->pRight, key);
    }
    else if(pNode->data > key){
        pNode->pLeft = deleteNodeRec(pNode->pLeft, key);
    }
    else if(pNode->data == key && pNode->pLeft == NULL){
        Node* temp = pNode->pRight;
        free(pNode);
        return temp;
    }
    else if(pNode->data == key && pNode->pRight == NULL){
        Node* temp = pNode->pLeft;
        free(pNode);
        return temp;
    }
    else if(pNode->data == key && pNode->pRight != NULL && pNode->pLeft != NULL){
        Node* temp = maxValueNode(pNode->pLeft);
        pNode->data = temp->data;
        pNode->pLeft = deleteNodeRec(pNode->pLeft, temp->data);
    }

    if (pNode == NULL){
        return pNode;
    }

    int bf = BalanceFactor(pNode);
    //LL Rotation
    if(bf < LH && BalanceFactor(root->pLeft) <= EH){
        return LLRotation(pNode);
    }
    //RR Rotation
```

```
    if(bf > RH && BalanceFactor(root->pLeft) >= EH){
        return RRRotation(pNode);
    }
    //LR Rotation
    if(bf < LH && BalanceFactor(root->pLeft) < EH){
        return LRRotation(pNode);
    }
    //RL Rotation
    if(bf > RH && BalanceFactor(root->pLeft) > EH){
        return RLRotation(pNode);
    }
    return pNode;
}


void remove(const T& value){
    this->root = deleteNodeRec(this->root, value);
}
```

Precheck Kiểm tra

	Test	Expected	Got	

✗	AVLTree<int> avl;
	int arr[] = {10, 5, 15, 7};
	for (int i = 0; i < 4; i++)
	{
	  avl.insert(arr[i]);
	}
	avl.remove(15);
	avl.printTreeStructure();
		7
		5 10
			10
			5
			7
				✗

✓	AVLTree<int> avl;
	int arr[] = {10, 5, 15, 3};
	for (int i = 0; i < 4; i++)
	{
	  avl.insert(arr[i]);
	}
	avl.remove(15);
	avl.printTreeStructure();
		5
		3 10
			5
			3 10
				✓

✓	AVLTree<int> avl;
	int arr[] = {10,52,98,32,68,92,40,13,42,63,99,100};
	for (int i = 0; i < 12; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(52);
	avl.printTreeStructure();
		42
		32 92
		10 40 68 99
		13 63 98 100
			42
			32 92
			10 40 68 99
			13 63 98 100
				✓

✓	AVLTree<int> avl;
	int arr[] = {20,10,40,5,7,42,2};
	for (int i = 0; i < 7; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(20);
	avl.printTreeStructure();
		10
		5 40
		2 7 42
			10
			5 40
			2 7 42
				✓

	Test	Expected	Got	

✔	AVLTree<int> avl;
	int arr[] = {20,10,40,5,7,42,2};
	for (int i = 0; i < 7; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(10);
	avl.printTreeStructure();
		20
		5 40
		2 7 42
			20
			5 40
			2 7 42
				✔

✔	AVLTree<int> avl;
	int arr[] = {20,10,40,5,7,42,2,6};
	for (int i = 0; i < 8; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(10);
	avl.printTreeStructure();
		20
		5 40
		2 7 42
		6
			20
			5 40
			2 7 42
			6
				✔

✖	AVLTree<int> avl;
	int arr[] = {20,10,40,5,7,42,2,6};
	for (int i = 0; i < 8; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(2);
	avl.remove(10);
	avl.printTreeStructure();
		20
		6 40
		5 7 42
			20
			5 40
			7 42
			6
				✖

✔	AVLTree<int> avl;
	int arr[] = {20,10,40,5,7,42,2,6,15};
	for (int i = 0; i < 9; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(6);
	avl.remove(42);
	avl.printTreeStructure();
		7
		5 20
		2 10 40
		15
			7
			5 20
			2 10 40
			15
				✔

✔	AVLTree<int> avl;
	int arr[] = {20,10,40,5,7,42,2,6,15};
	for (int i = 0; i < 9; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(40);
	avl.printTreeStructure();
		7
		5 20
		2 6 10 42
		15
			7
			5 20
			2 6 10 42
			15
				✔

✔	AVLTree<int> avl;
	int arr[] = {20,10,40,5,7,42,2,6,15};
	for (int i = 0; i < 9; i++){
	\tavl.insert(arr[i]);
	}
	avl.remove(40);
	avl.remove(20);
	avl.remove(6);
	avl.remove(10);
	avl.remove(42);
	avl.remove(15);
	avl.printTreeStructure();
		5
		2 7
			5
			2 7
				✔

Your code must pass all tests to earn any marks. Try again.

Show differences

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