Đã bắt đầu vào	Thứ hai, 12 Tháng mười hai 2022, 5:59 PM
lúc	
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ hai, 12 Tháng mười hai 2022, 6:00 PM
Thời gian thực hiện	37 giây
Điểm	3,00 của 3,00 (100 %)

Câu hỏi 1

Chính xác

Điểm 1,00 của 1,00

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. 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";</pre>
            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;</pre>
                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) {}
   };
};
```

For example:

Test	Result
AVLTree <int> avl;</int>	3
for (int $i = 0$; $i < 9$; $i++$){	1 5
<pre>avl.insert(i);</pre>	0 2 4 7
}	6 8
<pre>avl.printTreeStructure();</pre>	
AVLTree <int> avl;</int>	7
for (int $i = 10$; $i >= 0$; i){	3 9
<pre>avl.insert(i);</pre>	1 5 8 10
}	0 2 4 6
<pre>avl.printTreeStructure();</pre>	

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
//Helping functions
    Node* rotL(Node*root){
3
        Node*temp=root->pRight;
4
        root->pRight=temp->pLeft;
5
        temp->pLeft=root;
 6
        return temp;
7
    Node* rotR(Node*root){
8 .
9
        Node*temp=root->pLeft;
10
        root->pLeft=temp->pRight;
11
        temp->pRight=root;
        return temp;
12
13
14 -
    int balance(Node*root){
        int leftheight=getHeightRec(root->pLeft);
15
16
        int rightheight=getHeightRec(root->pRight);
17
        return rightheight-leftheight;
18
   Node* insert(T value,Node*root){
19 •
20
        if(!root) root=new Node(value);
21
        else if(root->data>value) root->pLeft=insert(value,root->pLeft);
```

```
else root->pRight=insert(value,root->pRight);
23
        int b_fact=balance(root);
24 -
        if(b_fact>1){
            if(root->pRight->data<=value) {//có dấu bằng chỗ so sánh bên trái
25
26
                root=rotL(root);
27
               // return root;
28
            }
29
            else{
30
                root->pRight=rotR(root->pRight);
                root=rotL(root);
31
32
                //return root;
            }
33
34
35
        else if(b_fact<-1){</pre>
36
            if(root->pLeft->data>value) {
37
                root=rotR(root);
38
39
                //return root;
40
            }
            else{
41
42
                root->pLeft=rotL(root->pLeft);
43
                root=rotR(root);
44
                //return root;
45
            }
46
47
        return root;
48
49
    void insert(const T &value){
50
51
        root=insert(value,root);
52
53
    }
54
55
```

	Test	Expected	Got	
~	<pre>AVLTree<int> avl; for (int i = 0; i < 9; i++){ avl.insert(i); } avl.printTreeStructure();</int></pre>	3 1 5 0 2 4 7 6 8	3 1 5 0 2 4 7 6 8	~
~	AVLTree <int> avl; for (int i = 10; i >= 0; i){ \tavl.insert(i); } avl.printTreeStructure();</int>	7 3 9 1 5 8 10 0 2 4 6	7 3 9 1 5 8 10 0 2 4 6	~

Passed all tests! 🗸

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

11



Chính xác

Điểm 1,00 của 1,00

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

- Provided insert function already.

Your task is to implement two functions: **remove**. 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";</pre>
            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;</pre>
                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) {}
   };
};
```

For example:

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

Answer: (penalty regime: 5, 10, 15, ... %)

Reset answer

```
Node* rotL(Node* root) {
1 .
2
            Node* temp = root->pRight;
            root->pRight = temp->pLeft;
3
4
            temp->pLeft = root;
5
            return temp;
6
7
        Node* rotR(Node* root) {
8
            Node* temp = root->pLeft;
9
            root->pLeft = temp->pRight;
10
            temp->pRight = root;
11
            return temp;
12
        int balance(Node* root) {
13
14
            int leftheight = getHeightRec(root->pLeft);
15
            int rightheight = getHeightRec(root->pRight);
16
            return rightheight - leftheight;
17
```

```
18 •
        int getmax(Node* root) {
            while (root->pRight != nullptr) {
19
20
                root = root->pRight;
21
22
            return root->data;
23
        Node* remove(Node* root, T value) {
24
25
            if (root == nullptr) return nullptr;
            else if (value < root->data) root->pLeft = remove(root->pLeft, value);
26
            else if (root->data < value) root->pRight = remove(root->pRight, value);
27
28
            else {
29
30
                if (root->pleft == nullptr && root->pRight == nullptr) { delete root; return nullptr; }
                else if (root->pLeft != nullptr && root->pRight != nullptr) {
31
32
                    int temp = getmax(root->pLeft);
33
                    root->data = temp;
                    root->pLeft = remove(root->pLeft, temp);
34
35
                }
                else {
36
37
                    Node* ok = root->pLeft ? root->pLeft : root->pRight;
38
                    return ok;
39
40
            };
            int b_fact = getHeightRec(root->pRight) - getHeightRec(root->pLeft);
41
42
            if (b_fact > 1) {
43
                int b_fact2 = getHeightRec(root->pRight->pRight) - getHeightRec(root->pRight->pLeft);
44
                if (b_fact2 >= 0) {
45
                    root = rotL(root);
46
47
                    root->pRight = rotR(root->pRight);
48
49
                    root = rotL(root);
50
                }
51
52
            else if (b_fact < -1) {</pre>
                int b_fact2 = getHeightRec(root->pLeft->pRight) - getHeightRec(root->pLeft->pLeft);
53
54
                if (b_fact2 <= 0) {</pre>
55
                    root = rotR(root);
56
57
                else {
                    root->pLeft = rotL(root->pLeft);
58
59
                    root = rotR(root);
60
                }
61
62
            return root;
63
64
        void remove(const T& value) {
65
66
            root = remove(root, value);
67
68
        }
```

	Test	Expected	Got	
~	AVLTree <int> avl;</int>	52	52	~
	int arr[] = {10,52,98,32,68,92,40,13,42,63};	32 92	32 92	
	for (int i = 0; i < 10; i++){	13 40 68 98	13 40 68 98	
	<pre>\tavl.insert(arr[i]);</pre>	42 63	42 63	
	}			
	avl.remove(10);			
	<pre>avl.printTreeStructure();</pre>			

11

	Test	Exped	tec	ı		Got				
~	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(13); avl.printTreeStructure();</int>	32 10	57 40 42	2 9 68 63	_	32 10	52 ! 40 42 (9: 68 63	2 99 98 100	~

Passed all tests! 🗸

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

```
Câu hỏi 3
Chính xác
Điểm 1,00 của 1,00
```

In this question, you have to search and print inorder on AVL tree. You have o implement functions: search and printlnorder to complete the task. Note that:

- When the tree is null, don't print anything.
- There's a whitespace at the end when print the tree inorder in case the tree is not null.
- When tree contains value, search return true.

```
#include <iostream>
#include <queue>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"
enum BalanceValue
    LH = -1,
   EH = 0,
    RH = 1
template<class T>
class AVLTree
public:
   class Node;
private:
   Node *root;
public:
   AVLTree() : root(nullptr) {}
   ~AVLTree(){}
    void printInorder(){
        //TODO
   bool search(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> av1;
int arr[] = {10,52,98,32,68,92,40,13,42,63,99,100};
for (int i = 0; i < 12; i++){
    avl.insert(arr[i]);
}
avl.printInorder();
cout << endl;
cout << avl.search(10);

Result

10 13 32 40 42 52 63 68 92 98 99 100
1
```

Answer: (penalty regime: 5, 10, 15, ... %)

```
void inorderrec(Node*cur){
   if(cur==nullptr) return;
   inorderrec(cur->pLeft);
```

```
inorderrec(cur->pRight);
 6
    }
 7
    void printInorder(){
 8 .
 9
        inorderrec(root);
10
    bool searchrec(Node*cur,const T&value){
11 ,
12
        if(cur==nullptr) return 0;
        if(value>cur->data) return searchrec(cur->pRight,value);
13
        else if(value<cur->data) return searchrec(cur->pLeft,value);
14
15
        else return 1;
16
    bool search(const T &value){
17
18
            //TODOr
        return searchrec(root,value);
19
20
```

	Test	Expected	Got					
~	AVLTree <int> avl;</int>	10 13 32 40 42 52 63 68 92 98 99 100	10 13 32 40 42 52 63 68 92 98 99 100	~				
	int arr[] = {10,52,98,32,68,92,40,13,42,63,99,100};	1	1					
	for (int i = 0; i < 12; i++){							
	\tavl.insert(arr[i]);							
	}							
	avl.printInorder();							
	<pre>cout << endl;</pre>							
	<pre>cout << avl.search(10);</pre>							

Passed all tests! 🗸

Question author's solution (Cpp):

```
//Helping funtions

void printInorder(){}
```

12/12/22, 10:10 PM AVL: Attempt review

bool search(const T &value){}

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

Copyright 2007-2022 BKEL - Phát triển dựa trên Moodle