Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Tư, 7 tháng 5 2025, 12:41 AM
Kết thúc lúc	Thứ Tư, 7 tháng 5 2025, 12:46 AM
Thời gian thực hiện	5 phút 17 giây
Điểm	6,90/7,00
Điểm	**9,86** trên 10,00 (**98,57**%)

Câu hỏi **1**

Đúng

Đạt điểm 1,00 trên 1,00

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	Result

`// 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();`
	`After inserting 0, 1. Tree:`
	` 0`
	`  1`
	
	`After inserting 2, perform 'rotateLeft'. Tree:`
	` 1`
	`0 2`

Test	Result

`// 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();`
	`After inserting 10, 9. Tree:`
	` 10`
	`9`
	
	`After inserting 8, perform 'rotateRight'. Tree:`
	` 9`
	`8 10`

Answer: (penalty regime: 0 %)

Reset answer

```
// Hàm rotateLeft: Thực hiện quay trái một node trong cây AVL
Node* rotateLeft(Node* subroot)
{
    // Kiểm tra xem subroot có tồn tại hay không
    if (subroot == nullptr) return nullptr;

    // Lưu lại node con bên phải của subroot, node này sẽ trở thành root mới sau khi quay
    Node* newRoot = subroot->pRight;

    // Kiểm tra xem có thể quay không (newRoot phải tồn tại)
    if (newRoot == nullptr) return subroot;

    // Thực hiện quay trái:
    // 1. Gán con phải của subroot là con trái của newRoot
    subroot->pRight = newRoot->pLeft;

    // 2. Gán con trái của newRoot là subroot
    newRoot->pLeft = subroot;

    // 3. Cập nhật giá trị balance nếu cần (trong trường hợp này không cập nhật)
    // Có thể bổ sung dựa trên chiều cao của các nhánh

    // 4. Trả về root mới sau khi quay
    return newRoot;
}

// Hàm rotateRight: Thực hiện quay phải một node trong cây AVL
Node* rotateRight(Node* subroot)
{
    // Kiểm tra xem subroot có tồn tại hay không
    if (subroot == nullptr) return nullptr;

    // Lưu lại node con bên trái của subroot, node này sẽ trở thành root mới sau khi quay
    Node* newRoot = subroot->pLeft;

    // Kiểm tra xem có thể quay không (newRoot phải tồn tại)
    if (newRoot == nullptr) return subroot;

    // Thực hiện quay phải:
    // 1. Gán con trái của subroot là con phải của newRoot
    subroot->pLeft = newRoot->pRight;

    // 2. Gán con phải của newRoot là subroot
    newRoot->pRight = subroot;

    // 3. Cập nhật giá trị balance nếu cần (trong trường hợp này không cập nhật)
    // Có thể bổ sung dựa trên chiều cao của các nhánh

    // 4. Trả về root mới sau khi quay
    return newRoot;
}

```

	Test	Expected	Got	

✓	`// 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();`
		`After inserting 0, 1. Tree:`
		` 0`
		`  1`
		
		`After inserting 2, perform 'rotateLeft'. Tree:`
		` 1`
		`0 2`
			`After inserting 0, 1. Tree:`
			` 0`
			`  1`
			
			`After inserting 2, perform 'rotateLeft'. Tree:`
			` 1`
			`0 2`
				✓

✓	`// 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();`
		`After inserting 10, 9. Tree:`
		` 10`
		`9`
		
		`After inserting 8, perform 'rotateRight'. Tree:`
		` 9`
		`8 10`
			`After inserting 10, 9. Tree:`
			` 10`
			`9`
			
			`After inserting 8, perform 'rotateRight'. Tree:`
			` 9`
			`8 10`
				✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **2**

Đúng

Đạt điểm 1,00 trên 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**. The function should cover at least these cases:

+ Balanced tree

+ Left of left unbalanced tree

+ Right of left 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;`
`for (int i = 0; i >= -10; i--){`
`  avl.insert(i);`
`}`
`avl.printTreeStructure();`
	`  -3`
	`  -7  -1`
	`-9 -5 -2 0`
	`-10 -8 -6 -4`

`AVLTree<int> avlTree;`
`avlTree.insert(5);`
`avlTree.insert(7);`
`avlTree.insert(6);`
`avlTree.printTreeStructure();`
	`6`
	`5 7`

Answer: (penalty regime: 0 %)

Reset answer

```
// Hàm lấy hệ số cân bằng của một node
    int getBalance(Node* node)
    {
        if (node == NULL)
            return 0;
        return getHeightRec(node->pLeft) - getHeightRec(node->pRight);
    }
// Hàm quay phải - xử lý trường hợp Left-Left
    Node* rotateRight(Node* y)
    {
        // Lưu node con trái của y
        Node* x = y->pLeft;
        // Lưu con phải của x
        Node* T2 = x->pRight;

        // Thực hiện phép quay
        x->pRight = y;      // y trở thành con phải của x
        y->pLeft = T2;      // T2 trở thành con trái của y
```

```
        // Cập nhật giá trị balance nếu cần
        // (ở đây dựa vào hàm getBalance khi cần)
        // Trả về node gốc mới sau khi quay
        return x;
    }
// Hàm quay trái - xử lý trường hợp Right-Right
    Node* rotateLeft(Node* x)
    {
        // Lưu node con phải của x
        Node* y = x->pRight;
        // Lưu con trái của y
        Node* T2 = y->pLeft;
        // Thực hiện phép quay
        y->pLeft = x;       // x trở thành con trái của y
        x->pRight = T2;     // T2 trở thành con phải của x
        // Cập nhật giá trị balance nếu cần
        // Trả về node gốc mới sau khi quay
        return y;
    }
// Hàm đệ quy thực hiện chèn node vào cây AVL
    Node* insertNode(Node* node, const T& value)
    {
        /* 1. Thực hiện chèn BST bình thường */
        if (node == NULL)
            return new Node(value);

        // Nếu giá trị bằng node hiện tại, chèn vào cây con bên phải
        if (value == node->data)
            node->pRight = insertNode(node->pRight, value);
        // Nếu giá trị nhỏ hơn node hiện tại, chèn vào cây con bên trái
        else if (value < node->data)
            node->pLeft = insertNode(node->pLeft, value);
        // Nếu giá trị lớn hơn node hiện tại, chèn vào cây con bên phải
        else
            node->pRight = insertNode(node->pRight, value);

        /* 2. Kiểm tra và cân bằng lại cây nếu cần */
        // Tính hệ số cân bằng của node hiện tại
        int balance = getBalance(node);
        // Trường hợp Left-Left: Cây lệch trái và node chèn vào bên trái của con trái
        if (balance > 1 && value < node->pLeft->data)
            return rotateRight(node);
        // Trường hợp Right-Right: Cây lệch phải và node chèn vào bên phải của con phải
        if (balance < -1 && (value > node->pRight->data || value == node->pRight->data))
```

	Test	Expected	Got	

✓	`AVLTree<int> avl;`
	`for (int i = 0; i >= -10; i--){`
	`  avl.insert(i);`
	`}`
	`avl.printTreeStructure();`
		`-3`
		`-7 -1`
		`-9 -5 -2 0`
		`-10 -8 -6 -4`
			`-3`
			`-7 -1`
			`-9 -5 -2 0`
			`-10 -8 -6 -4`
				✓

✓	`AVLTree<int> avlTree;`
	`avlTree.insert(5);`
	`avlTree.insert(7);`
	`avlTree.insert(6);`
	`avlTree.printTreeStructure();`
		`6`
		`5 7`
			`6`
			`5 7`
				✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **3**

Đúng

Đạt điểm 1,00 trên 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`. 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

```
24  // Hàm quay trái - xử lý trường hợp Right-Right
25      Node* rotateLeft(Node* x)
26      {
27          // Lưu node con phải của x
28          Node* y = x->pRight;
29          // Lưu con trái của y
30          Node* T2 = y->pLeft;
31          // Thực hiện phép quay
32          y->pLeft = x;       // x trở thành con trái của y
33          x->pRight = T2;     // T2 trở thành con phải của x
34          // Cập nhật giá trị balance nếu cần
```

```
        // Trả về node gốc mới sau khi quay
        return y;
    }
// Hàm đệ quy thực hiện chèn node vào cây AVL
    Node* insertNode(Node* node, const T& value)
    {
        /* 1. Thực hiện chèn BST bình thường */
        if (node == NULL)
            return new Node(value);

        // Nếu giá trị bằng node hiện tại, chèn vào cây con bên phải
        if (value == node->data)
            node->pRight = insertNode(node->pRight, value);
        // Nếu giá trị nhỏ hơn node hiện tại, chèn vào cây con bên trái
        else if (value < node->data)
            node->pLeft = insertNode(node->pLeft, value);
        // Nếu giá trị lớn hơn node hiện tại, chèn vào cây con bên phải
        else
            node->pRight = insertNode(node->pRight, value);

        /* 2. Kiểm tra và cân bằng lại cây nếu cần */
        // Tính hệ số cân bằng của node hiện tại
        int balance = getBalance(node);
        // Trường hợp Left-Left: Cây lệch trái và node chèn vào bên trái của con trái
        if (balance > 1 && value < node->pLeft->data)
            return rotateRight(node);
        // Trường hợp Right-Right: Cây lệch phải và node chèn vào bên phải của con phải
        if (balance < -1 && (value > node->pRight->data || value == node->pRight->data))
            return rotateLeft(node);
        // Trường hợp Left-Right: Cây lệch trái nhưng node chèn vào bên phải của con trái
        if (balance > 1 && (value > node->pLeft->data || value == node->pLeft->data))
        {
            node->pLeft = rotateLeft(node->pLeft);
            return rotateRight(node);
        }
        // Trường hợp Right-Left: Cây lệch phải nhưng node chèn vào bên trái của con phải
        if (balance < -1 && value < node->pRight->data)
        {
            node->pRight = rotateRight(node->pRight);
            return rotateLeft(node);
        }
        // Trả về node hiện tại (không cần cân bằng)
        return node;
    }
// Hàm insert công khai gọi từ bên ngoài
    void insert(const T &value)
    {
        this->root = insertNode(this->root, value);
    }


```

	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! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **4**

Đúng

Đạt điểm 1,00 trên 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";
            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;`
`for (int i = 0; i < 9; i++){`
`  avl.insert(i);`
`}`
`avl.printTreeStructure();`
	`  3`
	`1  5`
	`0 2 4 7`
	`  6 8`

`AVLTree<int> avl;`
`for (int i = 10; i >= 0; i--){`
`  avl.insert(i);`
`}`
`avl.printTreeStructure();`
	`  7`
	`3  9`
	`1 5 8 10`
	`0 2 4 6`

Answer: (penalty regime: 0 %)

Reset answer

```
// Hàm lấy hệ số cân bằng của một node
    int getBalance(Node* node)
    {
        if (node == NULL)
            return 0;
        return getHeightRec(node->pLeft) - getHeightRec(node->pRight);
    }
// Hàm quay phải - xử lý trường hợp Left-Left
    Node* rotateRight(Node* y)
    {
        // Lưu node con trái của y
        Node* x = y->pLeft;
        // Lưu con phải của x
        Node* T2 = x->pRight;

        // Thực hiện phép quay
        x->pRight = y;      // y trở thành con phải của x
        y->pLeft = T2;      // T2 trở thành con trái của y
```

```
        // Cập nhật giá trị balance nếu cần
        // (ở đây dựa vào hàm getBalance khi cần)
        // Trả về node gốc mới sau khi quay
        return x;
    }
// Hàm quay trái - xử lý trường hợp Right-Right
    Node* rotateLeft(Node* x)
    {
        // Lưu node con phải của x
        Node* y = x->pRight;
        // Lưu con trái của y
        Node* T2 = y->pLeft;
        // Thực hiện phép quay
        y->pLeft = x;       // x trở thành con trái của y
        x->pRight = T2;     // T2 trở thành con phải của x
        // Cập nhật giá trị balance nếu cần
        // Trả về node gốc mới sau khi quay
        return y;
    }
// Hàm đệ quy thực hiện chèn node vào cây AVL
    Node* insertNode(Node* node, const T& value)
    {
        /* 1. Thực hiện chèn BST bình thường */
        if (node == NULL)
            return new Node(value);

        // Nếu giá trị bằng node hiện tại, chèn vào cây con bên phải
        if (value == node->data)
            node->pRight = insertNode(node->pRight, value);
        // Nếu giá trị nhỏ hơn node hiện tại, chèn vào cây con bên trái
        else if (value < node->data)
            node->pLeft = insertNode(node->pLeft, value);
        // Nếu giá trị lớn hơn node hiện tại, chèn vào cây con bên phải
        else
            node->pRight = insertNode(node->pRight, value);

        /* 2. Kiểm tra và cân bằng lại cây nếu cần */
        // Tính hệ số cân bằng của node hiện tại
        int balance = getBalance(node);
        // Trường hợp Left-Left: Cây lệch trái và node chèn vào bên trái của con trái
        if (balance > 1 && value < node->pLeft->data)
            return rotateRight(node);
        // Trường hợp Right-Right: Cây lệch phải và node chèn vào bên phải của con phải
        if (balance < -1 && (value > node->pRight->data || value == node->pRight->data))
```

	Test	Expected	Got	

✓	AVLTree<int> avl;
	for (int i = 0; i < 9; i++){
	avl.insert(i);
	}
	avl.printTreeStructure();
		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();
		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! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **5**

Đúng một phần

Đạt điểm 0,90 trên 1,00

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

- Provided **insert** function already.

- You **must** adjust the **balance factor**, **not the height** of tree.

Your task is to implement function: **remove** to perform re-balancing (balanced, left of left, right of left, equal of left). When deleting a root with both left and right tree, please choose the maximum value of left tree to replace the root. 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

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

```
// Xoay kép trái-phải (LR)
    void doubleRotateLeftRight(Node *&node) {
        // Thực hiện xoay kép: xoay trái node->pLeft, sau đó xoay phải node
        Node *temp = node->pLeft;
        Node *temp2 = temp->pRight;

        temp->pRight = temp2->pLeft;
        temp2->pLeft = temp;
        node->pLeft = temp2->pRight;
        temp2->pRight = node;
        // Cập nhật hệ số cân bằng dựa trên hệ số cân bằng của node giữa
        switch(temp2->balance) {
            case LH: // Nếu node giữa lệch trái
                node->balance = RH;  // Node gốc cũ sẽ lệch phải
                temp->balance = EH;  // Node con trái (temp) sẽ cân bằng
                break;
            case EH: // Nếu node giữa cân bằng
                node->balance = EH;  // Cả hai node đều cân bằng
                temp->balance = EH;
                break;
            case RH: // Nếu node giữa lệch phải
                node->balance = EH;  // Node gốc cũ sẽ cân bằng
                temp->balance = LH;  // Node con trái (temp) sẽ lệch trái
                break;
        }
        temp2->balance = EH; // Node gốc mới sẽ cân bằng
        node = temp2; // Node gốc mới
    }

// Xoay kép phải-trái (RL)
    void doubleRotateRightLeft(Node *&node) {
        // Thực hiện xoay kép: xoay phải node->pRight, sau đó xoay trái node
        Node *temp = node->pRight;
        Node *temp2 = temp->pLeft;

        temp->pLeft = temp2->pRight;
        temp2->pRight = temp;
        node->pRight = temp2->pLeft;
        temp2->pLeft = node;

        // Cập nhật hệ số cân bằng
        switch(temp2->balance) {
            case LH: // Nếu node giữa lệch trái
                node->balance = EH;  // Node gốc cũ sẽ cân bằng
                temp->balance = RH;  // Node con phải (temp) sẽ lệch phải
                break;
            case EH: // Nếu node giữa cân bằng
                node->balance = EH;  // Cả hai node đều cân bằng
                temp->balance = EH;
                break;
            case RH: // Nếu node giữa lệch phải
```

```
52                  node->balance = LH;  // Node gốc cũ sẽ lệch trái
53                  temp->balance = EH;  // Node con phải (temp) sẽ cân bằng
54                  break;
55          }
56
57          temp2->balance = EH; // Node gốc mới sẽ cân bằng
58          node = temp2; // Node gốc mới
59      }
60  // Tìm node có giá trị lớn nhất trong cây con
61  Node* findMax(Node *node) {
62      while (node->pRight != NULL) {
```

	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
			7
			5 10
				✓

✓	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
				✓

Your code failed one or more hidden tests.

Đúng một phần

Marks for this submission: 0,90/1,00.

Câu hỏi **6**

Đúng

Đạt điểm 1,00 trên 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";
            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[] = {10,52,98,32,68,92,40,13,42,63};`
`for (int i = 0; i < 10; i++){`
`  avl.insert(arr[i]);`
`}`
`avl.remove(10);`
`avl.printTreeStructure();`
	52
	32 92
	13 40 68 98
	42 63

`AVLTree<int> avl;`
`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.remove(13);`
`avl.printTreeStructure();`
	52
	32 92
	10 40 68 99
	42 63 98 100

Answer: (penalty regime: 0 %)

Reset answer

```
60 // Tìm node có giá trị lớn nhất trong cây con
61 Node* findMax(Node *node) {
62     while (node->pRight != NULL) {
63         node = node->pRight;
64     }
65     return node;
66 }
67 // Cân bằng khi chiều cao cây con trái giảm sau khi xóa node
68 void balanceLeftAfterRemove(Node *&node, bool &shorter) {
69     switch (node->balance) {
70         case LH: // Nếu trước đó cây lệch trái, giờ sẽ cân bằng
71             node->balance = EH;
72             break;
```

```
        case EH: // Nếu trước đó cây cân bằng, giờ sẽ lệch phải
            node->balance = RH;
            shorter = false; // Chiều cao tổng thể không giảm
            break;
        case RH: // Nếu trước đó cây lệch phải, giờ sẽ mất cân bằng nghiêm trọng
            Node *rightNode = node->pRight;
            BalanceValue rb = rightNode->balance;

            if (rb != LH) {
                // Trường hợp R-R hoặc cây con phải cân bằng
                rotateLeft(node);
                if (rb == EH) {
                    // Nếu cây con phải cân bằng, chiều cao tổng thể không giảm
                    shorter = false;
                }
            } else {
                // Trường hợp R-L
                doubleRotateRightLeft(node);
            }
            break;
    }
}

// Cân bằng khi chiều cao cây con phải giảm sau khi xóa node
void balanceRightAfterRemove(Node *&node, bool &shorter) {
    switch (node->balance) {
        case RH: // Nếu trước đó cây lệch phải, giờ sẽ cân bằng
            node->balance = EH;
            break;
        case EH: // Nếu trước đó cây cân bằng, giờ sẽ lệch trái
            node->balance = LH;
            shorter = false; // Chiều cao tổng thể không giảm
            break;
        case LH: // Nếu trước đó cây lệch trái, giờ sẽ mất cân bằng nghiêm trọng
            Node *leftNode = node->pLeft;
            BalanceValue lb = leftNode->balance;

            if (lb != RH) {
                // Trường hợp L-L hoặc cây con trái cân bằng
                rotateRight(node);
                if (lb == EH) {
                    // Nếu cây con trái cân bằng, chiều cao tổng thể không giảm
                    shorter = false;
                }
            } else {
                // Trường hợp L-R
                doubleRotateLeftRight(node);
            }
            break;
```

	Test	Expected	Got	

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

✓	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();
		52
		32 92
		10 40 68 99
		42 63 98 100
			52
			32 92
			10 40 68 99
			42 63 98 100
				✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.

Câu hỏi **7**

Đúng

Đạt điểm 1,00 trên 1,00

In this question, you have to search and print inorder on **AVL tree**. You have o implement functions: **search** and **printInorder** 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> avl;
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);
	10 13 32 40 42 52 63 68 92 98 99 100
	1

Answer: (penalty regime: 0 %)

```
1  // Hàm in cây theo thứ tự inorder (trái-gốc-phải)
```

```
    void printInorder() {
        printInorderRec(root);
    }

    // Hàm đệ quy để duyệt cây theo thứ tự inorder
    void printInorderRec(Node* node) {
        if (node != nullptr) {
            // Duyệt cây con trái
            printInorderRec(node->pLeft);

            // In giá trị của node hiện tại
            cout << node->data << " ";

            // Duyệt cây con phải
            printInorderRec(node->pRight);
        }
    }

    // Hàm tìm kiếm một giá trị trong cây
    bool search(const T &value) {
        return searchRec(root, value);
    }

    // Hàm đệ quy để tìm kiếm một giá trị trong cây
    bool searchRec(Node* node, const T &value) {
        // Nếu node hiện tại là null, giá trị không tồn tại trong cây
        if (node == nullptr) {
            return false;
        }

        // Nếu giá trị bằng với giá trị của node hiện tại, trả về true
        if (value == node->data) {
            return true;
        }

        // Nếu giá trị nhỏ hơn giá trị của node hiện tại, tìm kiếm ở cây con trái
        if (value < node->data) {
            return searchRec(node->pLeft, value);
        }

        // Nếu giá trị lớn hơn giá trị của node hiện tại, tìm kiếm ở cây con phải
        return searchRec(node->pRight, value);
    }
```

	Test	Expected	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.printInorder();
	cout << endl;
	cout << avl.search(10);
		10 13 32 40 42 52 63 68 92 98 99 100
		1
			10 13 32 40 42 52 63 68 92 98 99 100
			1
				✓

Passed all tests! ✓

Đúng

Marks for this submission: 1,00/1,00.