TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

BÁO CÁO BÀI TẬP:

CÀI ĐẶT CÂY AVL CHỨA CÁC SỐ NGHUYÊN

SINH VIÊN THỰC HIỆN: Nguyễn Đỗ Huy – 3121411085

LỚP: DCT124C7

GVHD: ĐỖ NHƯ TÀI

Thành phố Hồ Chí Minh , Tháng 3 Năm 2025

MỤC LỤC

I. Ý tưởng cây AVL

- Cây AVL là cây nhị phân tìm kiếm (BST) nhưng được cân bằng chiều cao.
- Tại mỗi nút, độ cao của cây con trái và cây con phải không được lệch nhau quá 1.
- Khi thêm hoặc xóa làm mất cân bằng, ta cần quay cây:
  - Quay đơn: LL, RR
  - Quay kép: LR, RL

II. Input / Output

1. Input

- Một danh sách các số nguyên cần thêm vào cây.
- Sau đó có thể yêu cầu xóa một số nút.
- Có thể yêu cầu duyệt cây (NLR, LNR, LRN) hoặc tìm kiếm.

2. Output

- Cây AVL sau khi thêm / xóa.
- Kết quả tìm kiếm (có hoặc không).
- Duyệt cây theo thứ tự được yêu cầu.

III. Tập testcase

Input	Output dự kiến
values[] = {5, 2, 4, 8, 1, 9, 6};	InOrder (LNR): 1 2 4 5 6 8 9 PreOrder (NLR): 4 2 1 8 5 6 9 PostOrder (LRN): 1 2 6 5 9 8 4 Tim 4? Co Tim 10? Khong Sau khi xoa 2, InOrder: 1 4 5 6 8 9

IV. Mã nguồn

1. Cấu trúc Node

```
// Cấu trúc Node
struct Node {
    int info, height, count;
    Node* left;
    Node* right;

    Node(int val) {
        info = val;
        height = 1;
        count = 1;
        left = right = nullptr;
    }
};
```

- info: giá trị của nút.

- height: chiều cao của cây tính từ nút này (dùng để tính cân bằng AVL).
- count: số lần giá trị này được chèn (nếu cho phép trùng lặp).
- left, right: con trái và con phải.

2. Hàm phụ trợ

- Height(Node*): trả về chiều cao của nút.

```
int Height(Node* n) {
    return n ? n->height : 0;
}
```

- BalanceFactor(Node*): trả về độ lệch cân bằng trái - phải

```
int BalanceFactor(Node* n) {
    return n ? Height(n->left) - Height(n->right) : 0;
}
```

- UpdateHeight(Node*): cập nhật chiều cao lại cho node.

```
void UpdateHeight(Node* n) {
    n->height = 1 + max(Height(n->left), Height(n->right));
}
```

3. Quay cây để cân bằng

- Quay phải (Right Rotation – LL Case)

```
Node* RotateRight(Node* y) {
    Node* x = y->left;
    Node* T2 = x->right;

    x->right = y;
    y->left = T2;

    UpdateHeight(y);
    UpdateHeight(x);

    return x;
}
```

- Quay trái (Left Rotation – RR Case)

```
Node* RotateLeft(Node* x) {
    Node* y = x->right;
    Node* T2 = y->left;

    y->left = x;
    x->right = T2;

    UpdateHeight(x);
```

```
    UpdateHeight(y);

    return y;
}
```

4. Chèn phần tử (Insert)

```
// Thêm nút
Node* Insert(Node* root, int key) {
    if (!root) return new Node(key);

    if (key < root->info)
        root->left = Insert(root->left, key);
    else if (key > root->info)
        root->right = Insert(root->right, key);
    else {
        root->count++;
        return root;
    }

    UpdateHeight(root);
    int balance = BalanceFactor(root);

    // Quay nếu mất cân bằng
    if (balance > 1 && key < root->left->info)
        return RotateRight(root);
    if (balance < -1 && key > root->right->info)
        return RotateLeft(root);
    if (balance > 1 && key > root->left->info) {
        root->left = RotateLeft(root->left);
        return RotateRight(root);
    }
    if (balance < -1 && key < root->right->info) {
        root->right = RotateRight(root->right);
        return RotateLeft(root);
    }

    return root;
}
```

- Chèn như cây BST bình thường.
- Sau khi chèn → cập nhật chiều cao → tính balance.
- Nếu balance > 1 hoặc < -1, thực hiện quay:
    - LL → quay phải
    - RR → quay trái
    - LR → quay trái con trái rồi quay phải

- RL → quay phải con phải rồi quay trái

5. Tìm nút nhỏ nhất (phục vụ xóa)

```
// Tìm node nhỏ nhất
Node* MinValueNode(Node* n) {
    Node* current = n;
    while (current->left) current = current->left;
    return current;
}
```

- Dò xuống nhánh trái đến tận cùng để tìm node nhỏ nhất.

6. Xóa phần tử

```
// Xóa nút
Node* Delete(Node* root, int key) {
    if (!root) return root;

    if (key < root->info)
        root->left = Delete(root->left, key);
    else if (key > root->info)
        root->right = Delete(root->right, key);
    else {
        if (root->count > 1) {
            root->count--;
            return root;
        }
        if (!root->left || !root->right) {
            Node* temp = root->left ? root->left : root->right;
            delete root;
            return temp;
        }

        Node* temp = MinValueNode(root->right);
        root->info = temp->info;
        root->count = temp->count;
        root->right = Delete(root->right, temp->info);
    }

    UpdateHeight(root);
    int balance = BalanceFactor(root);

    // Quay lại nếu mất cân bằng
    if (balance > 1 && BalanceFactor(root->left) >= 0)
```

```
        return RotateRight(root);
    if (balance > 1 && BalanceFactor(root->left) < 0) {
        root->left = RotateLeft(root->left);
        return RotateRight(root);
    }
    if (balance < -1 && BalanceFactor(root->right) <= 0)
        return RotateLeft(root);
    if (balance < -1 && BalanceFactor(root->right) > 0) {
        root->right = RotateRight(root->right);
        return RotateLeft(root);
    }

    return root;
}
```

- Xóa như cây BST:
    - Nếu node cần xóa có 2 con: thay bằng node nhỏ nhất bên phải.
- Sau khi xóa → cập nhật lại chiều cao → kiểm tra balance.
- Nếu mất cân bằng thì quay lại giống như Insert.

7. Tìm kiếm

```
// Tìm kiếm
bool Search(Node* root, int key) {
    if (!root) return false;
    if (key == root->info) return true;
    else if (key < root->info) return Search(root->left, key);
    else return Search(root->right, key);
}
```

- Tìm giá trị theo kiểu cây BST.
- Trả về true nếu có, false nếu không.

8. Duyệt cây

- InOrder (LNR): Trái → Gốc → Phải (tăng dần với BST)

```
void InOrder(Node* root) {
    if (!root) return;
    InOrder(root->left);
    cout << root->info << " ";
    InOrder(root->right);
}
```

- PreOrder (NLR): Gốc → Trái → Phải

```
void PreOrder(Node* root) {
    if (!root) return;
```

```
    cout << root->info << " ";
    PreOrder(root->left);
    PreOrder(root->right);
}
```

- PostOrder (LRN): Trái → Phải → Gốc

```
void PostOrder(Node* root) {
    if (!root) return;
    PostOrder(root->left);
    PostOrder(root->right);
    cout << root->info << " ";
}
```

9. Hàm main()

```
// Main
int main() {
    Node* root = nullptr;

    int values[] = {5, 2, 4, 8, 1, 9, 6};
    int n = sizeof(values) / sizeof(values[0]);

    for (int i = 0; i < n; ++i)
        root = Insert(root, values[i]);

    cout << "InOrder (LNR): ";
    InOrder(root); cout << endl;

    cout << "PreOrder (NLR): ";
    PreOrder(root); cout << endl;

    cout << "PostOrder (LRN): ";
    PostOrder(root); cout << endl;

    cout << "\nTim 4? " << (Search(root, 4) ? "Co" : "Khong") << endl;
    cout << "Tim 10? " << (Search(root, 10) ? "Co" : "Khong") << endl;

    root = Delete(root, 2);
    cout << "\nSau khi xoa 2, InOrder: ";
    InOrder(root); cout << endl;

    return 0;
}
```

- Chèn lần lượt 7 số vào cây AVL.
- In cây theo 3 cách duyệt.
- Tìm thử số 4 (tồn tại) và 10 (không tồn tại).

- Xóa node 2 rồi in lại cây.