CHƯƠNG 4: CÂY (TREE)

---

1. LÝ THUYẾT CHUNG

- **Cây (Tree)** là một cấu trúc dữ liệu phi tuyến, bao gồm tập hợp các nút (nodes), trong đó có một nút gốc (root), và các nút con phân cấp theo nhánh.

Khái niệm:

- **Nút gốc (Root)**: Nút không có cha.
- **Nút lá (Leaf)**: Nút không có con.
- **Cấp (Level)**: Số lượng cạnh từ root đến node đó.
- **Chiều cao (Height)**: Số mức sâu nhất trong cây.
- **Cây nhị phân**: Mỗi node có tối đa 2 node con.

---

2. CÂY NHỊ PHÂN TÌM KIẾM (BST – Binary Search Tree)

- Mỗi node có tối đa 2 con: bên trái nhỏ hơn, bên phải lớn hơn.

Cài đặt BST + duyệt cây:

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;
};

Node* createNode(int val) {
    Node* newNode = new Node{val, nullptr, nullptr};
    return newNode;
}

Node* insert(Node* root, int val) {
    if (root == nullptr)
        return createNode(val);
    if (val < root->data)
        root->left = insert(root->left, val);
    else if (val > root->data)
        root->right = insert(root->right, val);
    return root;
}

// NLR: Pre-order
void preorder(Node* root) {
    if (root) {
        cout << root->data << " ";
        preorder(root->left);
        preorder(root->right);
    }
}
```

```
// LNR: In-order (duyệt tăng dần)
void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

// LRN: Post-order
void postorder(Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        cout << root->data << " ";
    }
}

int main() {
    Node* root = nullptr;
    root = insert(root, 10);
    insert(root, 5);
    insert(root, 15);
    insert(root, 2);
    insert(root, 7);

    cout << "Duyệt NLR: "; preorder(root); cout << endl;
    cout << "Duyệt LNR: "; inorder(root); cout << endl;
    cout << "Duyệt LRN: "; postorder(root); cout << endl;

    return 0;
}
```

---

3. CÂY CÂN BẰNG AVL

- **AVL Tree** là cây nhị phân tìm kiếm, nhưng luôn giữ cho chiều cao cân bằng.
- Mỗi node có `balance factor = height(left) - height(right)` ∈ {-1, 0, 1}

Các loại xoay:

- Xoay đơn trái / phải.
- Xoay kép trái phải / phải trái.

---

4. CÂY HEAP

- **Heap** là cây nhị phân hoàn chỉnh, thường dùng để xây dựng **Priority Queue**.
- **Min-Heap**: node cha ≤ node con.
- **Max-Heap**: node cha ≥ node con.

Min-Heap dùng mảng:

```
#include <iostream>
#include <vector>
using namespace std;
```

```
class MinHeap {
private:
    vector<int> heap;

    void heapifyUp(int i) {
        while (i > 0 && heap[i] < heap[(i - 1) / 2]) {
            swap(heap[i], heap[(i - 1) / 2]);
            i = (i - 1) / 2;
        }
    }

    void heapifyDown(int i) {
        int n = heap.size();
        while (2 * i + 1 < n) {
            int smallest = i;
            int left = 2 * i + 1, right = 2 * i + 2;
            if (left < n && heap[left] < heap[smallest]) smallest = left;
            if (right < n && heap[right] < heap[smallest]) smallest =
right;
            if (smallest == i) break;
            swap(heap[i], heap[smallest]);
            i = smallest;
        }
    }

public:
    void insert(int val) {
        heap.push_back(val);
        heapifyUp(heap.size() - 1);
    }

    void removeMin() {
        if (heap.empty()) return;
        heap[0] = heap.back();
        heap.pop_back();
        heapifyDown(0);
    }

    int getMin() {
        return heap.empty() ? -1 : heap[0];
    }

    void print() {
        for (int x : heap)
            cout << x << " ";
        cout << endl;
    }
};

int main() {
    MinHeap h;
    h.insert(5);
    h.insert(3);
    h.insert(8);
    h.insert(1);

    h.print(); // 1 3 8 5 (có thể khác tùy cách xây)
    h.removeMin();
    h.print(); // 3 5 8

    return 0;
}
```

---

5. CÂY B-TREE (tổng quan)

- **B-Tree** là cây tìm kiếm tự cân bằng đa phân (đa nhánh), dùng trong hệ thống cơ sở dữ liệu.
- Mỗi node chứa nhiều khóa.
- Đặc điểm:
    - Cân bằng, tự động chia node khi đầy.
    - Tốc độ tìm kiếm/log nhanh, lưu trữ tốt trên đĩa cứng.