⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

MẢNG (PHẦN 1)

1 Two Sum

Bài toán:

Cho mảng số nguyên nums và số nguyên target, hãy trả về chỉ số của hai số có tổng bằng target.

Ví dụ:

nums = [2, 7, 11, 15], target = 9

Output: [0, 1]

(2 + 7 = 9)

Cách tiếp cận:

Dùng HashMap để lưu giá trị và chỉ số của nó. Với mỗi số, kiểm tra xem (target – num) đã có trong map hay chưa.

Code C++:

```cpp
vector<int> twoSum(vector<int>& nums,
                   int target) {
    unordered_map<int, int> mp;
    for(int i = 0; i < nums.size(); i++) {
        int diff = target - nums[i];
        if(mp.count(diff)) {
            return {mp[diff], i};
        }
        mp[nums[i]] = i;
    }
    return {};
}
```

Thời gian: O(n)

Bộ nhớ: O(n)

Mẹo phỏng vấn:

Đây là bài kinh điển. Luôn thử xem HashMap có giúp giảm độ phức tạp xuống O(n) hay không.

2 Best Time to Buy & Sell Stock

Bài toán:

Cho mảng prices, trong đó prices[i] là giá cổ phiếu ngày thứ i. Trả về lợi nhuận lớn nhất đạt được khi mua vào một ngày và bán ở một ngày sau đó.

Ví dụ:

prices = [7, 1, 5, 3, 6, 4]

Output: 5

(Mua ở 1, bán ở 6)

Cách tiếp cận:

Luôn theo dõi giá nhỏ nhất từ đầu đến hiện tại và lợi nhuận lớn nhất thu được.

Code C++:

```cpp
int maxProfit(vector<int>& prices) {
    int minPrice = INT_MAX;
    int maxProfit = 0;
    for(int price : prices) {
        minPrice = min(minPrice, price);
        maxProfit = max(maxProfit,
                        price - minPrice);
    }
    return maxProfit;
}
```

Thời gian: O(n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Cách tiếp cận greedy hoạt động rất tốt ở đây. Luôn ghi nhớ lựa chọn tốt nhất đã gặp được.

3 Contains Duplicate

Bài toán:

Cho mảng số nguyên nums, trả về true nếu có giá trị nào xuất hiện ít nhất hai lần, ngược lại trả về false.

Ví dụ:

nums = [1, 2, 3, 1]

Output: true

(số 1 lặp lại)

Cách tiếp cận:

Dùng HashSet để lưu các phần tử. Nếu một phần tử đã tồn tại trong set thì trả về true.

Code C++:

```cpp
bool containsDuplicate(vector<int>& nums) {
    unordered_set<int> st;
    for(int num : nums) {
        if(st.count(num)) return true;
        st.insert(num);
    }
    return false;
}
```

Thời gian: O(n)

Bộ nhớ: O(n)

Mẹo phỏng vấn:

HashSet cho thao tác tra cứu O(1). Nếu bị giới hạn bộ nhớ, hãy sắp xếp rồi so sánh hai phần tử liền kề với O(n log n).

⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

MẢNG (PHẦN 2)

4 Product of Array Except Self

Bài toán:

Cho mảng số nguyên nums, trả về mảng ans sao cho ans[i] là tích của tất cả phần tử trong nums ngoại trừ nums[i]. Không được dùng phép chia.

Ví dụ:

nums = [1, 2, 3, 4]

Output: [24, 12, 8, 6]

Cách tiếp cận:

Dùng mảng tích tiền tố (prefix) và mảng tích hậu tố (suffix).

Code C++:

```
vector<int> productExceptSelf(
                    vector<int>& nums) {
    int n = nums.size();
    vector<int> ans(n, 1);
    int prefix = 1;
    // Tích tiền tố (prefix)
    for(int i = 0; i < n; i++) {
        ans[i] = prefix;
        prefix *= nums[i];
    }
    // Tích hậu tố (suffix)
    int suffix = 1;
    for(int i = n - 1; i >= 0; i--) {
        ans[i] *= suffix;
        suffix *= nums[i];
    }
    return ans;
}
```

Thời gian: O(n)

Bộ nhớ: O(n)

Mẹo phỏng vấn:

Hãy nghĩ tới prefix và suffix product. Cách này tránh được phép chia và vẫn đúng khi mảng chứa số 0.

5 Maximum Subarray (Kadane's Algorithm)

Bài toán:

Cho mảng số nguyên nums, tìm mảng con (subarray) có tổng lớn nhất và trả về tổng đó.

Ví dụ:

nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output: 6

(Mảng con: [4, -1, 2, 1])

Cách tiếp cận:

Dùng thuật toán Kadane. Luôn theo dõi tổng hiện tại và tổng lớn nhất tìm được.

Code C++:

```
int maxSubArray(vector<int>& nums) {
    int maxSum = nums[0];
    int currentSum = nums[0];
    for(int i = 1; i < nums.size(); i++) {
        currentSum = max(nums[i],
                         currentSum + nums[i]);
        maxSum = max(maxSum, currentSum);
    }
    return maxSum;
}
```

Thời gian: O(n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Đặt lại tổng hiện tại bằng chính phần tử đang xét nếu tổng đó trở thành số âm.

6 Merge Sorted Arrays

Bài toán:

Cho hai mảng số nguyên đã sắp xếp nums1 và nums2, cùng hai số m và n là số phần tử thực của mỗi mảng. Hãy trộn nums2 vào nums1 thành một mảng đã sắp xếp.

Ví dụ:

nums1 = [1, 2, 3, 0, 0, 0], m = 3

nums2 = [2, 5, 6], n = 3

Output: [1, 2, 2, 3, 5, 6]

Cách tiếp cận:

Dùng ba con trỏ. Bắt đầu từ cuối cả hai mảng và đặt phần tử lớn hơn vào cuối nums1.

Code C++:

```
void merge(vector<int>& nums1, int m,
           vector<int>& nums2, int n) {
    int i = m - 1;       // cuối nums1
    int j = n - 1;       // cuối nums2
    int k = m + n - 1;   // cuối mảng gộp
    while(j >= 0) {
        if(i >= 0 && nums1[i] > nums2[j])
            nums1[k--] = nums1[i--];
        else
            nums1[k--] = nums2[j--];
    }
}
```

Thời gian: O(m + n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Điền từ cuối về đầu để tránh ghi đè lên những phần tử của nums1 chưa được xử lý.

⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

CHUỖI & SLIDING WINDOW

7 Valid Anagram

Bài toán:

Cho hai chuỗi s và t, trả về true nếu t là một anagram (đảo chữ) của s, ngược lại trả về false.

Ví dụ:

s = "listen", t = "silent"

Output: true

Cách tiếp cận:

Đếm tần suất xuất hiện của các ký tự (26 chữ cái). Nếu hai chuỗi có cùng bảng tần suất thì chúng là anagram của nhau.

Code C++:

```cpp
bool isAnagram(string s, string t) {
    if(s.size() != t.size()) return false;
    vector<int> count(26, 0);
    for(char ch : s) count[ch - 'a']++;
    for(char ch : t) count[ch - 'a']--;
    for(int x : count) {
        if(x != 0) return false;
    }
    return true;
}
```

Thời gian: O(n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Câu hỏi rất phổ biến để kiểm tra khả năng xử lý chuỗi cơ bản và kỹ thuật đếm tần suất.

8 Longest Substring Without Repeating Characters

Bài toán:

Cho chuỗi s, hãy tìm độ dài của chuỗi con (substring) dài nhất không chứa ký tự lặp lại.

Ví dụ:

s = "abcabcbb"

Output: 3 ("abc")

Cách tiếp cận:

Dùng Sliding Window kết hợp HashMap/Set để lưu vị trí xuất hiện gần nhất của mỗi ký tự.

Code C++:

```cpp
int lengthOfLongestSubstring(string s) {
    unordered_map<char, int> lastIndex;
    int left = 0, maxLen = 0;
    for(int r = 0; r < s.size(); r++) {
        if(lastIndex.count(s[r]) &&
           lastIndex[s[r]] >= left) {
            left = lastIndex[s[r]] + 1;
        }
        lastIndex[s[r]] = r;
        maxLen = max(maxLen, r - left + 1);
    }
    return maxLen;
}
```

Thời gian: O(n)

Bộ nhớ: O(min(n, k))

k = số ký tự khác nhau

Mẹo phỏng vấn:

Bài sliding window kinh điển. Chỉ cập nhật con trỏ trái khi ký tự lặp nằm bên trong cửa sổ hiện tại.

9 Longest Repeating Character Replacement

Bài toán:

Cho chuỗi s và số nguyên k. Bạn được phép thay tối đa k ký tự trong chuỗi bằng bất kỳ chữ in hoa nào. Trả về độ dài chuỗi con dài nhất chỉ gồm một loại ký tự sau khi thực hiện các thao tác trên.

Ví dụ:

s = "AABABBA", k = 1

Output: 4 ("AABA" → "AAAA")

Cách tiếp cận:

Dùng Sliding Window. Theo dõi tần suất lớn nhất trong cửa sổ. Nếu window_size − maxFreq > k thì thu nhỏ cửa sổ lại.

Code C++:

```cpp
int characterReplacement(string s, int k) {
    vector<int> freq(26, 0);
    int left = 0, maxFreq = 0, maxLen = 0;
    for(int r = 0; r < s.size(); r++) {
        freq[s[r] - 'A']++;
        maxFreq = max(maxFreq,
                      freq[s[r] - 'A']);
        while((r - left + 1) - maxFreq > k) {
            freq[s[left] - 'A']--;
            left++;
        }
        maxLen = max(maxLen, r - left + 1);
    }
    return maxLen;
}
```

Thời gian: O(n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Điểm mấu chốt: không cần tính lại maxFreq khi thu nhỏ cửa sổ, vì đáp án chỉ tăng khi maxFreq tăng.

⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

TÌM KIẾM NHỊ PHÂN

10 Binary Search

Bài toán:

Cho mảng số nguyên đã sắp xếp nums và giá trị target, trả về chỉ số của target. Nếu không tìm thấy, trả về -1.

Ví dụ:

nums = [-1, 0, 3, 5, 9, 12]
target = 9
Output: 4

Cách tiếp cận:

Dùng Binary Search. So sánh phần tử ở giữa với target rồi loại bỏ một nửa không gian tìm kiếm.

Code C++:

```
int binarySearch(vector<int>& nums,
                 int target) {
    int left = 0, right = nums.size() - 1;
    while(left <= right) {
        int mid = left + (right - left) / 2;
        if(nums[mid] == target) return mid;
        else if(nums[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}
```

Thời gian: O(log n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Luôn tính mid = left + (right – left) / 2 để tránh tràn số nguyên khi mảng rất lớn.

11 Search in Rotated Sorted Array

Bài toán:

Cho mảng đã sắp xếp bị xoay vòng nums (không có phần tử trùng) và giá trị target, trả về chỉ số của target. Nếu không tìm thấy, trả về -1.

Ví dụ:

nums = [4, 5, 6, 7, 0, 1, 2]
target = 0
Output: 4

Cách tiếp cận:

Dùng binary search có biến đổi. Ở mỗi bước, xác định nửa nào đang được sắp xếp và kiểm tra target nằm ở đâu.

Code C++:

```
int search(vector<int>& nums, int target) {
    int left = 0, right = nums.size() - 1;
    while(left <= right) {
        int mid = left + (right - left) / 2;
        if(nums[mid] == target) return mid;
        // Nửa trái đã sắp xếp
        if(nums[left] <= nums[mid]) {
            if(nums[left] <= target &&
               target < nums[mid])
                right = mid - 1;
            else left = mid + 1;
        }
        // Nửa phải đã sắp xếp
        else {
            if(nums[mid] < target &&
               target <= nums[right])
                left = mid + 1;
            else right = mid - 1;
        }
    }
    return -1;
}
```

Thời gian: O(log n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Xử lý thật cẩn thận các điều kiện biên (<=, <, >, >=). Hãy vẽ mảng ra giấy để dễ hình dung hơn.

12 First Bad Version

Bài toán:

Bạn là quản lý sản phẩm và có n phiên bản [1..n]. Hãy tìm phiên bản lỗi đầu tiên. Một phiên bản là lỗi nếu isBadVersion(i) trả về true.

Ví dụ:

n = 5, phiên bản lỗi đầu tiên = 4
Output: 4

Cách tiếp cận:

Dùng binary search. Nếu mid là phiên bản lỗi thì tìm tiếp ở nửa trái, ngược lại tìm ở nửa phải.

Code C++:

```
int firstBadVersion(int n) {
    int left = 1, right = n, ans = n;
    while(left <= right) {
        int mid = left + (right - left) / 2;
        if(isBadVersion(mid)) {
            ans = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    return ans;
}
```

Thời gian: O(log n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Đây là dạng "binary search trên đáp án" kinh điển: tìm biên đầu tiên thỏa mãn điều kiện cho trước.

⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

LINKED LIST

13 Reverse Linked List

Bài toán:

Cho head của một singly linked list, hãy đảo ngược danh sách và trả về head mới.

Ví dụ:

Input: 1 → 2 → 3 → 4 → 5 → NULL

Output: 5 → 4 → 3 → 2 → 1 → NULL

Cách tiếp cận:

Duyệt qua danh sách và đảo chiều các liên kết bằng ba con trỏ: prev, curr và next.

Code C++:

```
ListNode* reverseList(ListNode* head) {
    ListNode *prev = NULL, *curr = head;
    ListNode *next;
    while(curr != NULL) {
        next = curr->next;  // lưu node kế
        curr->next = prev;  // đảo liên kết
        prev = curr;        // tiến prev
        curr = next;        // tiến curr
    }
    return prev;            // head mới
}
```

Thời gian: O(n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Mấu chốt là cập nhật các liên kết đúng thứ tự. Đừng làm mất tham chiếu tới node kế tiếp khi đảo chiều.

14 Merge Two Sorted Lists

Bài toán:

Cho head của hai linked list đã sắp xếp list1 và list2. Hãy trộn hai danh sách thành một danh sách đã sắp xếp và trả về head.

Ví dụ:

Input:

list1: 1 → 2 → 4

list2: 1 → 3 → 4

Output: 1 → 1 → 2 → 3 → 4 → 4

Cách tiếp cận:

Dùng hai con trỏ để duyệt song song hai danh sách. Chọn node nhỏ hơn và nối dần vào danh sách kết quả.

Code C++:

```
ListNode* mergeTwoLists(ListNode* l1,
                        ListNode* l2) {
    ListNode dummy(0);
    ListNode* tail = &dummy;
    while(l1 && l2) {
        if(l1->val <= l2->val) {
            tail->next = l1;
            l1 = l1->next;
        } else {
            tail->next = l2;
            l2 = l2->next;
        }
        tail = tail->next;
    }
    tail->next = (l1) ? l1 : l2;
    return dummy.next;
}
```

Thời gian: O(m + n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

Dùng dummy node để code gọn hơn và tránh phải xử lý riêng trường hợp đầu danh sách kết quả.

15 Linked List Cycle

Bài toán:

Cho head của một linked list, hãy xác định xem danh sách có chứa chu trình hay không. Trả về true nếu có, ngược lại trả về false.

Ví dụ:

Input: head = [3, 2, 0, -4], pos = 1

[3] → [2] → [0] → [-4]

Output: true (có chu trình)

Cách tiếp cận:

Dùng thuật toán Floyd's Tortoise and Hare (con trỏ chậm và nhanh). Nếu có chu trình, hai con trỏ chắc chắn sẽ gặp nhau.

Code C++:

```
bool hasCycle(ListNode *head) {
    if(head == NULL || head->next == NULL)
        return false;
    ListNode *slow = head, *fast = head;
    while(fast && fast->next) {
        slow = slow->next;          // đi 1 bước
        fast = fast->next->next;    // đi 2 bước
        if(slow == fast) return true;
    }
    return false;
}
```

Thời gian: O(n)

Bộ nhớ: O(1)

Mẹo phỏng vấn:

fast đi 2 bước, slow đi 1 bước. Nếu chúng gặp nhau thì tồn tại chu trình; nếu fast chạm NULL thì không có.

⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

STACK & QUEUE

16 Valid Parentheses

Bài toán:

Cho chuỗi s chỉ gồm các ký tự '(', ')', '{', '}', '[' và ']', hãy kiểm tra xem chuỗi đó có hợp lệ hay không.

Ví dụ:

Input: s = "()[]{}"
Output: true

Input: s = "(]"
Output: false

Cách tiếp cận:

Dùng stack. Đẩy các dấu mở vào stack. Với dấu đóng, kiểm tra xem nó có khớp với phần tử ở đỉnh stack hay không.

Code C++:

```
bool isValid(string s) {
    stack<char> st;
    unordered_map<char, char> mp =
        {{')','('}, {'}','{'}, {']','['}};
    for(char ch : s) {
        if(mp.count(ch)) {        // dấu đóng
            if(st.empty() ||
               st.top() != mp[ch]) return false;
            st.pop();
        } else {                  // dấu mở
            st.push(ch);
        }
    }
    return st.empty();
}
```

Thời gian: O(n)

Bộ nhớ: O(n)

Mẹo phỏng vấn:

Chỉ push các dấu mở. Với dấu đóng, chỉ cần kiểm tra đỉnh stack xem có khớp hay không.

17 Min Stack

Bài toán:

Thiết kế một stack hỗ trợ các thao tác push, pop, top và lấy phần tử nhỏ nhất trong thời gian hằng số.

Ví dụ:

Input: ["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]
Output: [null,null,null,null,-3,null,0,-2]

Cách tiếp cận:

Dùng hai stack: một stack cho giá trị thường và một stack cho các giá trị nhỏ nhất.

Code C++:

```
class MinStack {
    stack<int> st, minSt;
public:
    void push(int val) {
        st.push(val);
        if(minSt.empty() ||
           val <= minSt.top()) minSt.push(val);
    }
    void pop() {
        if(st.top() == minSt.top())
            minSt.pop();
        st.pop();
    }
    int top() { return st.top(); }
    int getMin() { return minSt.top(); }
};
```

Thời gian: O(1) mọi thao tác

Bộ nhớ: O(n)

Mẹo phỏng vấn:

Chỉ lưu vào min stack khi xuất hiện giá trị nhỏ nhất mới, và chỉ pop khi chính giá trị đó bị lấy ra.

18 Daily Temperatures

Bài toán:

Cho mảng temperatures, trả về mảng answer với answer[i] là số ngày phải chờ kể từ ngày i để có nhiệt độ ấm hơn. Nếu không có ngày nào như vậy, answer[i] = 0.

Ví dụ:

temps = [73, 74, 75, 71, 69, 72, 76, 73]
Output: [1, 1, 4, 2, 1, 1, 0, 0]

Cách tiếp cận:

Dùng stack lưu chỉ số các ngày có nhiệt độ giảm dần. Khi gặp ngày ấm hơn, pop ra và tính hiệu chỉ số.

Code C++:

```
vector<int> dailyTemperatures(
                    vector<int>& temps) {
    int n = temps.size();
    vector<int> ans(n, 0);
    stack<int> st;          // lưu chỉ số
    for(int i = 0; i < n; i++) {
        while(!st.empty() &&
              temps[i] > temps[st.top()]) {
            int idx = st.top(); st.pop();
            ans[idx] = i - idx;
        }
        st.push(i);
    }
    return ans;
}
```

Thời gian: O(n)

Bộ nhớ: O(n)

Mẹo phỏng vấn:

Monotonic decreasing stack chính là chìa khóa cho mọi bài dạng "Next Greater Element".

⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

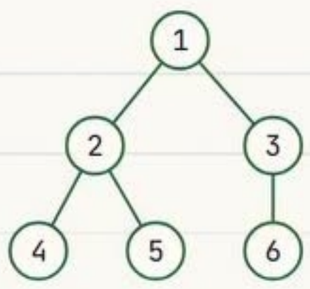
CÂY NHỊ PHÂN (TREES)

19 Maximum Depth of Binary Tree

Bài toán:

Cho node gốc của một cây nhị phân, hãy trả về độ sâu lớn nhất của cây.

Ví dụ:



Output: 3

Cách tiếp cận:

Dùng đệ quy.

Depth = 1 + max(độ sâu cây con trái, độ sâu cây con phải)

Code C++:

```
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x),
        left(NULL), right(NULL) {}
};

int maxDepth(TreeNode* root) {
    if(root == NULL) return 0;
    int leftDepth = maxDepth(root->left);
    int rightDepth = maxDepth(root->right);
    return 1 + max(leftDepth, rightDepth);
}
```

Thời gian: O(n)

Bộ nhớ: O(h)

(recursion stack)

Mẹo phỏng vấn:

Bài cây rất cơ bản. Luôn xử lý trường hợp NULL trước tiên để tránh lỗi runtime.

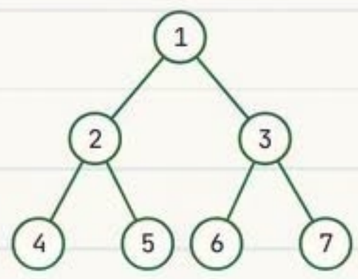
20 Invert Binary Tree

Bài toán:

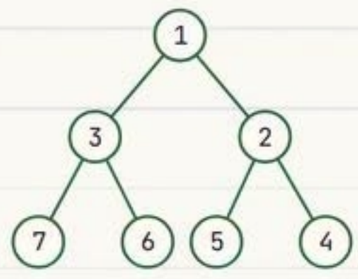
Cho node gốc của một cây nhị phân, hãy đảo ngược cây và trả về gốc của nó.

Ví dụ:

Input:



Output:



Cách tiếp cận:

Hoán đổi con trái và con phải của mọi node một cách đệ quy.

Code C++:

```
TreeNode* invertTree(TreeNode* root) {
    if(root == NULL) return NULL;
    // hoán đổi hai con
    TreeNode* temp = root->left;
    root->left = root->right;
    root->right = temp;

    invertTree(root->left);
    invertTree(root->right);
    return root;
}
```

Thời gian: O(n)

Bộ nhớ: O(h)

(recursion stack)

Mẹo phỏng vấn:

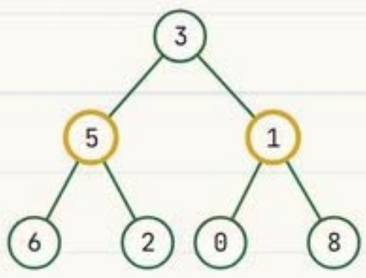
Có thể giải bằng đệ quy DFS. Hãy thử giải theo cách lặp bằng queue (BFS) như một câu hỏi mở rộng.

21 Lowest Common Ancestor

Bài toán:

Cho một cây nhị phân, hãy tìm tổ tiên chung thấp nhất (LCA) của hai node p và q.

Ví dụ:



Output: 3 (p = 5, q = 1)

Cách tiếp cận:

Nếu node hiện tại là NULL thì trả về NULL. Nếu node hiện tại là p hoặc q thì trả về chính nó. Đệ quy sang trái và phải: nếu cả hai bên đều khác NULL thì node hiện tại là LCA, ngược lại trả về bên khác NULL.

Code C++:

```
TreeNode* lowestCommonAncestor(
        TreeNode* root, TreeNode* p,
        TreeNode* q) {
    if(root == NULL || root == p ||
       root == q) return root;
    TreeNode* left = lowestCommonAncestor(
                        root->left, p, q);
    TreeNode* right = lowestCommonAncestor(
                        root->right, p, q);
    if(left && right) return root;
    return left ? left : right;
}
```

Thời gian: O(n)

Bộ nhớ: O(h)

(recursion stack)

Mẹo phỏng vấn:

Mẫu hình này (trả node hoặc NULL ngược lên trên đệ quy) được dùng trong rất nhiều bài toán về cây.

⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

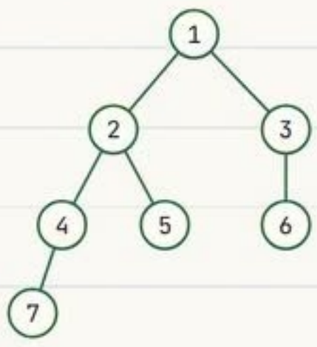
CÂY NHỊ PHÂN (TREES)

22 Diameter of Binary Tree

Bài toán:

Cho node gốc của một cây nhị phân, hãy trả về đường kính của cây (độ dài đường đi dài nhất giữa hai node bất kỳ).

Ví dụ:



Output: 4

(Đường dài nhất: 7 – 4 – 2 – 1 – 3)

Cách tiếp cận:

Dùng DFS. Tại mỗi node:

1. Lấy chiều cao của cây con trái và phải.
2. Cập nhật diameter = max(diameter, left + right).
3. Trả về height = 1 + max(left, right).

Code C++:

```cpp
class Solution {
public:
    int diameter = 0;
    int height(TreeNode* node) {
        if(node == NULL) return 0;
        int left = height(node->left);
        int right = height(node->right);
        diameter = max(diameter,
                       left + right);
        return 1 + max(left, right);
    }
    int diameterOfBinaryTree(TreeNode* root) {
        height(root);
        return diameter;
    }
};
```

Thời gian: O(n)

Bộ nhớ: O(h)

(recursion stack)

Mẹo phỏng vấn:

Đường kính có thể không đi qua gốc. Vì vậy luôn kiểm tra left + right tại mọi node.

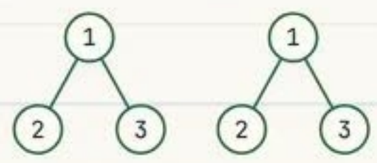
23 Same Tree

Bài toán:

Cho gốc của hai cây nhị phân p và q, hãy xác định xem chúng có giống hệt nhau về cấu trúc và giá trị các node hay không.

Ví dụ:

p: q:



Output: true

Cách tiếp cận:

So sánh đệ quy cả hai cây:

1. Nếu cả hai node đều NULL → true.
2. Nếu chỉ một node NULL → false.
3. So sánh giá trị rồi kiểm tra tiếp trái & phải.

Code C++:

```cpp
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if(p == NULL && q == NULL) return true;
        if(p == NULL || q == NULL) return false;
        if(p->val != q->val) return false;
        return isSameTree(p->left, q->left)
            && isSameTree(p->right, q->right);
    }
};
```

Thời gian: O(n)

Bộ nhớ: O(h)

(recursion stack)

Mẹo phỏng vấn:

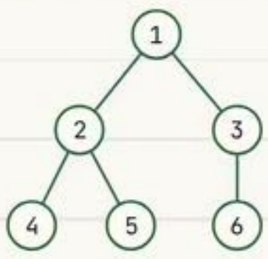
Luôn xử lý các trường hợp NULL trước khi truy cập giá trị của node.

24 Serialize and Deserialize Binary Tree

Bài toán:

Hãy thiết kế thuật toán mã hóa một cây nhị phân thành chuỗi và giải mã chuỗi đó trở lại đúng cây ban đầu.

Ví dụ:



Chuỗi mã hóa:

1,2,3,4,5,6,#,#,#,#,#,#,#

Cách tiếp cận:

Duyệt cây theo từng mức (BFS).

1. serialize: lưu giá trị node, dùng '#' cho NULL.
2. deserialize: dựng lại cây từ chuỗi đã lưu.

Code C++:

```cpp
class Codec {
public:
    // Mã hóa cây thành chuỗi (BFS)
    string serialize(TreeNode* root) {
        if(!root) return "";
        string s; queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()) {
            TreeNode* nd = q.front(); q.pop();
            if(!nd) { s += "#,"; continue; }
            s += to_string(nd->val) + ",";
            q.push(nd->left); q.push(nd->right);
        }
        return s;
    }
    // Giải mã chuỗi thành cây
    TreeNode* deserialize(string data) {
        if(data.empty()) return NULL;
        stringstream ss(data); string v;
        getline(ss, v, ',');
        TreeNode* root = new TreeNode(stoi(v));
        queue<TreeNode*> q; q.push(root);
        while(!q.empty()) {
            TreeNode* nd = q.front(); q.pop();
            if(getline(ss,v,',') && v != "#")
                q.push(nd->left = new TreeNode(stoi(v)));
            if(getline(ss,v,',') && v != "#")
                q.push(nd->right = new TreeNode(stoi(v)));
        }
        return root;
    }
};
```

Thời gian: O(n)

Bộ nhớ: O(n)

(queue + output)

Mẹo phỏng vấn:

Luôn thêm ký hiệu NULL (#) khi serialize; nếu không, bạn sẽ không thể dựng lại cây một cách duy nhất.

⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

ĐỒ THỊ (GRAPHS)

25 Number of Islands

Bài toán:

Cho lưới 2D gồm các ô '1' (đất) và '0' (nước), hãy trả về số lượng hòn đảo. Một đảo được bao quanh bởi nước và tạo thành từ các ô đất kề nhau theo chiều ngang hoặc dọc.

Ví dụ:

Input:

1	1	0	0	0
1	0	0	0	1
0	0	1	1	0
0	0	0	0	1

Output: 3

Cách tiếp cận:

Dùng DFS. Với mỗi ô có giá trị '1', chạy DFS và đánh dấu toàn bộ vùng đất liền kề là đã thăm, rồi đếm số đảo.

Code C++:

```cpp
class Solution {
public:
    int n, m;
    void dfs(vector<vector<char>>& g,
             int i, int j) {
        if(i < 0 || j < 0 || i >= n ||
           j >= m || g[i][j] != '1') return;
        g[i][j] = '0';
        dfs(g, i+1, j); dfs(g, i-1, j);
        dfs(g, i, j+1); dfs(g, i, j-1);
    }
    int numIslands(vector<vector<char>>& g) {
        n = g.size(); if(n == 0) return 0;
        m = g[0].size();
        int count = 0;
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
                if(g[i][j] == '1') {
                    dfs(g, i, j); count++;
                }
        return count;
    }
};
```

Thời gian: O(n × m)

Bộ nhớ: O(n × m)
(recursion stack)

Mẹo phỏng vấn:

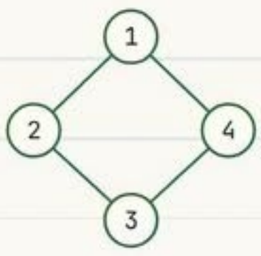
Hãy đánh dấu các ô đã thăm (đổi thành '0' hoặc dùng mảng visited) để không đếm trùng cùng một hòn đảo.

26 Clone Graph

Bài toán:

Cho tham chiếu tới một node trong đồ thị vô hướng liên thông, hãy trả về bản sao sâu (deep copy) của đồ thị đó.

Ví dụ:



Output: Bản sao sâu của đồ thị

Cách tiếp cận:

Dùng DFS kèm một map lưu ánh xạ node cũ → node mới. Nếu node đã được sao chép thì trả về luôn bản sao.

Code C++:

```cpp
class Solution {
public:
    unordered_map<Node*, Node*> mp;
    Node* cloneGraph(Node* node) {
        if(!node) return NULL;
        if(mp.count(node)) return mp[node];
        Node* copy = new Node(node->val);
        mp[node] = copy;
        for(Node* nei : node->neighbors)
            copy->neighbors.push_back(
                cloneGraph(nei));
        return copy;
    }
};
```

Thời gian: O(V + E)

Bộ nhớ: O(V)
(recursion stack + map)

Mẹo phỏng vấn:

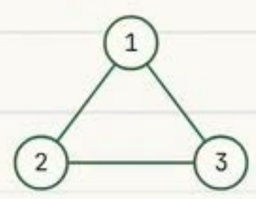
Luôn kiểm tra xem node đã được sao chép hay chưa trước khi sao chép tiếp, để tránh lặp vô hạn.

27 Detect Cycle in Undirected Graph

Bài toán:

Cho một đồ thị vô hướng, hãy xác định xem đồ thị có chứa chu trình hay không.

Ví dụ:



Output: true (1–2–3–1)

Cách tiếp cận:

Dùng BFS hoặc DFS. Trong lúc duyệt, nếu gặp một node đã thăm mà không phải node cha thì tồn tại chu trình.

Code C++:

```cpp
class Solution {
public:
    bool dfs(int node, int parent,
             vector<int> adj[],
             vector<int>& vis) {
        vis[node] = 1;
        for(int nei : adj[node]) {
            if(!vis[nei]) {
                if(dfs(nei, node, adj, vis))
                    return true;
            }
            else if(nei != parent) return true;
        }
        return false;
    }
    bool isCycle(int V, vector<int> adj[]) {
        vector<int> vis(V, 0);
        for(int i = 0; i < V; i++)
            if(!vis[i] && dfs(i, -1, adj, vis))
                return true;
        return false;
    }
};
```

Thời gian: O(V + E)

Bộ nhớ: O(V)
(recursion stack)

Mẹo phỏng vấn:

Luôn truyền node cha khi DFS/BFS. Nếu không, cạnh vô hướng u–v sẽ bị nhận nhầm thành chu trình.

⭐ Top 30 Câu Hỏi Phỏng Vấn DSA </>

kèm Lời Giải C++

Cách tiếp cận | Độ phức tạp thời gian | Độ phức tạp bộ nhớ | Mẹo phỏng vấn

QUY HOẠCH ĐỘNG (DP)

28 Fibonacci Number

Bài toán:

Cho số nguyên n, hãy trả về số Fibonacci thứ n.

Ví dụ:

Input: n = 6

Output: 8

Giải thích: Dãy Fibonacci:

```
0, 1, 1, 2, 3, 5, 8, ...
idx: 0 1 2 3 4 5 6
```

Cách tiếp cận:

Dùng DP theo hướng bottom-up (tabulation).

dp[i] = dp[i−1] + dp[i−2]

Code C++:

```cpp
int fib(int n) {
    if(n <= 1) return n;
    vector<long long> dp(n + 1, 0);
    dp[0] = 0;
    dp[1] = 1;
    for(int i = 2; i <= n; ++i) {
        dp[i] = dp[i-1] + dp[i-2];
    }
    return (int)dp[n];
}
```

Thời gian: O(n)

Bộ nhớ: O(n)

(có thể tối ưu về O(1))

Mẹo phỏng vấn:

Hãy nhắc tới lời giải đệ quy ngây thơ trước và giải thích vì sao nó bị lặp bài toán con, rồi mới chuyển sang DP.

29 0/1 Knapsack Problem

Bài toán:

Cho khối lượng và giá trị của n món đồ cùng một chiếc balo sức chứa W, hãy trả về tổng giá trị lớn nhất có thể bỏ vào balo.

Ví dụ:

Weights = [1, 3, 4, 5]

Values = [1, 4, 5, 7]

W = 7

Output: 9 (chọn món có wt 3 và 4)

Cách tiếp cận:

Dùng DP. dp[i][w] = giá trị lớn nhất khi dùng i món đầu tiên với sức chứa w.

Code C++:

```cpp
int knapSack(int W, vector<int>& wt,
             vector<int>& val, int n) {
    vector<vector<int>> dp(n + 1,
                vector<int>(W + 1, 0));
    for(int i = 1; i <= n; ++i) {
        for(int w = 0; w <= W; ++w) {
            if(wt[i-1] <= w) {
                dp[i][w] = max(val[i-1] +
                    dp[i-1][w - wt[i-1]],
                    dp[i-1][w]);
            } else {
                dp[i][w] = dp[i-1][w];
            }
        }
    }
    return dp[n][W];
}
```

Thời gian: O(n × W)

Bộ nhớ: O(n × W)

Mẹo phỏng vấn:

Hãy giải thích rõ lựa chọn cho từng món: lấy (nếu vừa balo) hoặc không lấy. Vẽ bảng DP với ví dụ nhỏ sẽ rất hiệu quả.

30 Longest Common Subsequence (LCS)

Bài toán:

Cho hai chuỗi text1 và text2, hãy trả về độ dài dãy con chung dài nhất của chúng.

Ví dụ:

text1 = "abcde"

text2 = "ace"

Output: 3 ("ace")

Cách tiếp cận:

Dùng DP. Nếu hai ký tự cuối khớp nhau thì cộng 1 + dp[i−1][j−1], ngược lại lấy max của dp[i−1][j] và dp[i][j−1].

Code C++:

```cpp
int lcs(string &a, string &b) {
    int n = a.size(), m = b.size();
    vector<vector<int>> dp(n + 1,
                vector<int>(m + 1, 0));
    for(int i = 1; i <= n; ++i) {
        for(int j = 1; j <= m; ++j) {
            if(a[i-1] == b[j-1])
                dp[i][j] = 1 + dp[i-1][j-1];
            else
                dp[i][j] = max(dp[i-1][j],
                               dp[i][j-1]);
        }
    }
    return dp[n][m];
}
```

Thời gian: O(n × m)

Bộ nhớ: O(n × m)

Mẹo phỏng vấn:

Nêu rõ khác biệt giữa subsequence (không liên tiếp) và substring (liên tiếp), rồi giải thích cách bảng DP được xây dựng.