CÂU HỎI PHỎNG VẤN DSA

THƯỜNG GẶP NHẤT (0-3 NĂM) (5-20 LPA)

1. ARRAYS – CÂU HỎI THƯỜNG GẶP NHẤT

★ Arrays là nền tảng của DSA. Phần lớn câu hỏi phỏng vấn cho người có 0-3 năm kinh nghiệm đều xoay quanh Arrays.

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Two Sum (LeetCode 1)	Hashing	O(n) time O(n) space	• Dùng HashMap để lưu number → index • Với mỗi number, kiểm tra xem target - num có tồn tại không
2	Best Time to Buy & Sell Stock (LC 121)	One Pass / Greedy	O(n) time O(1) space	• Theo dõi minimum price tính đến thời điểm hiện tại • Max profit = max(sell - min_buy)
3	Maximum Subarray (Kadane's Algo) (LC 53)	Kadane's Algorithm	O(n) time O(1) space	• Cộng dồn running sum • Nếu sum < 0, reset về 0 • Cập nhật max_sum
4	Move Zeroes (LC 283)	Two Pointers	O(n) time O(1) space	• Đưa toàn bộ phần tử khác 0 lên đầu • Điền phần còn lại bằng 0
5	Majority Element (LC 169)	Boyer-Moore Voting Algo	O(n) time O(1) space	• Đếm / triệt tiêu theo từng cặp • Candidate còn lại chính là majority
6	Merge Sorted Arrays (LC 88)	Two Pointers	O(n + m) time O(1) space	• Duyệt từ cuối mảng trở về đầu • Đặt phần tử lớn hơn vào cuối nums1
7	Product of Array Except Self (LC 238)	Prefix & Suffix	O(n) time O(1) space (ignoring output)	• Tính left product & right product • res[i] = left[i] * right[i]

★ **CÁC PATTERN QUAN TRỌNG CẦN NHẬN DIỆN**

→ Hashing Pattern
→ Subarray / Kadane's Pattern
→ Two Pointers Pattern
→ Prefix / Suffix Pattern
→ Sliding Window Pattern
→ Sorting / Greedy Pattern

2. STRINGS – CÂU HỎI THƯỜNG GẶP NHẤT

★ Câu hỏi về Strings kiểm tra kiến thức nền tảng về ký tự, hashing, con trỏ và khả năng nhận diện pattern.

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Valid Palindrome (LC 125)	Two Pointers	O(n) time O(1) space	• So sánh từ hai đầu vào giữa • Bỏ qua ký tự không phải chữ/số • So sánh không phân biệt hoa thường
2	Valid Anagram (LC 242)	Hashing (Frequency Count)	O(n) time O(1) space	• Đếm tần suất xuất hiện của từng ký tự • Hai chuỗi là anagram nếu mọi tần suất trùng khớp
3	Longest Common Prefix (LC 14)	Horizontal Scan (or Vertical)	O(S) time O(1) space	• So sánh ký tự ở cùng vị trí index • Dừng khi phát hiện không khớp • (S = tổng số ký tự)
4	Longest Substring Without Repeating Characters (LC 3)	Sliding Window	O(n) time O(k) space (k = bộ ký tự)	• Dùng HashMap / Set • Mở rộng cửa sổ, thu hẹp khi trùng lặp • Ghi nhớ độ dài lớn nhất
5	Group Anagrams (LC 49)	Hashing (Sorted String / Frequency)	O(n*k log k) time O(n*k) space (k = độ dài t.bình)	• Sắp xếp mỗi chuỗi hoặc dùng frequency map làm key • Gom nhóm các chuỗi có cùng key
6	String Compression (LC 443)	Two Pointers	O(n) time O(1) space	• Con trỏ đọc & con trỏ ghi • Đếm số ký tự liên tiếp giống nhau • Thay số đếm > 1 bằng chữ số

★ PATTERN NHẬN DIỆN – STRINGS

→ Two Pointers → Palindrome, Reverse, Compare

→ Hashing → Anagram, Frequency, Grouping

→ Sliding Window → Longest Substring problems

→ Prefix Matching → Common Prefix, Search

→ Simulation → Compression, Encoding

→ String Manipulation → Build, Transform, Parse

STRINGS – MẸO NHANH

✓ Ưu tiên xử lý ký tự trước (chữ hoa/thường, khoảng trắng, ký tự đặc biệt)

✓ Xác định xem cần hashing, two pointers hay sliding window

BẢNG ĐỘ PHỨC TẠP THỜI GIAN

O(1)	Hằng số
O(log n)	Logarit
O(n)	Tuyến tính

3. TWO POINTERS + SLIDING WINDOW – CÂU HỎI THƯỜNG GẶP NHẤT

★ Two Pointers giải quyết hiệu quả các bài toán trên mảng đã/chưa sắp xếp.
Sliding Window dùng cho các bài toán về dãy con / chuỗi con liên tiếp.

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Container With Most Water (LC 11)	Two Pointers (Opposite Ends)	O(n) time O(1) space	• Di chuyển con trỏ tại chiều cao nhỏ hơn • Area = $\min(h_1, h_2)$ x width
2	3Sum (LC 15)	Two Pointers + Sorting	$O(n^2)$ time O(1) space (chưa tính sort)	• Sắp xếp mảng • Cố định một phần tử • Dùng two pointers cho phần còn lại
3	Remove Duplicates from Sorted Array (LC 26)	Two Pointers	O(n) time O(1) space	• Con trỏ chậm giữ vị trí unique • Con trỏ nhanh quét qua mảng • Copy khi gặp phần tử unique mới
4	Longest Substring Without Repeating Characters (LC 3)	Sliding Window + Hashing	O(n) time O(k) space (k = bộ ký tự)	• Mở rộng cửa sổ để bao gồm thêm • Thu hẹp cửa sổ khi gặp trùng lặp • Ghi nhớ độ dài lớn nhất
5	Maximum Sum Subarray of Size K (LC 643)	Sliding Window	O(n) time O(1) space	• Tính tổng của cửa sổ đầu tiên • Trượt cửa sổ: cộng mới, bớt cũ • Theo dõi tổng lớn nhất
6	Minimum Window Substring (LC 76)	Sliding Window + Hashing	O(n) time O(σ) space (σ = ký tự duy nhất)	• Mở rộng để bao gồm mọi ký tự cần • Thu hẹp về cửa sổ hợp lệ nhỏ nhất • Theo dõi độ dài cửa sổ nhỏ nhất

★ KHI NÀO DÙNG CÁI GÌ?

→ Dùng **TWO POINTERS** khi:
✓ Cần tìm cặp / bộ ba (pairs/triplets)
✓ Mảng đã sắp xếp / có thể sắp xếp
✓ Bài toán yêu cầu min/max với hai giá trị

→ Dùng **SLIDING WINDOW** khi:
✓ Bài toán liên quan subarray / substring
✓ Cần tìm dài nhất/ngắn nhất thỏa điều kiện
✓ Phần tử dương / dựa trên số đếm

MẪU TWO POINTERS

```
left = 0, right = n - 1
while left < right:
    if condition:
        # xử lý logic tại đây
        do_something()
    if move_left_condition:
        left += 1
    else:
        right -= 1
```

💡 Di chuyển con trỏ một cách cẩn thận và luôn tư duy theo hướng: phần nào của mảng/chuỗi đã được xử lý xong.

MẪU SLIDING WINDOW

```
left = 0
for right in range(n):
    add(nums[right])
    # mở rộng cửa sổ
    while window_invalid:
        remove(nums[left])
        left += 1
        # thu hẹp cửa sổ
    update_answer()
```

4. HASHING + PREFIX SUM – CÂU HỎI THƯỜNG GẶP NHẤT

★ Hashing giúp tra cứu nhanh và xử lý các bài toán dựa trên tần suất.
Prefix Sum hỗ trợ các bài toán về dãy con / khoảng liên tiếp.

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Two Sum (HashMap) (LC 1)	Hashing (Map)	O(n) time O(n) space	• Lưu num → index vào HashMap • Với mỗi num, kiểm tra target - num • Trả về index nếu tìm thấy
2	Subarray Sum Equals K (LC 560)	Prefix Sum + Hashing	O(n) time O(n) space	• Cộng dồn prefix sum • Nếu (prefix - k) có trong map → tìm thấy subarray hợp lệ • Map lưu prefixSum → số lần đếm
3	Longest Consecutive Sequence (LC 128)	Hashing (Set)	O(n) time O(n) space	• Đưa toàn bộ số vào Set • Chỉ kiểm tra nếu (num-1) không có trong set • Đếm các số liên tiếp
4	Top K Frequent Elements (LC 347)	Hashing + Heap / Bucket	O(n log k) hoặc O(n) (bucket sort)	• Đếm tần suất bằng HashMap • Dùng Min-Heap kích thước k • Hoặc Bucket sort theo tần suất
5	Subarray with Given Sum (K) (LC 325)	Prefix Sum + Hashing	O(n) time O(n) space	• Lưu prefixSum → index sớm nhất • Nếu (prefix - k) tồn tại, cập nhật độ dài lớn nhất
6	Count of Nice Subarrays (LC 1248)	Prefix Sum + Hashing	O(n) time O(n) space	• Chuyển về 0/1 (số lẻ = 1) • prefixSum[i] - prefixSum[j] = k • → dùng map đếm tần suất prefixSum

★ TỐI ƯU TỪ $O(N^2)$ SANG O(N) BẰNG HASHING / PREFIX SUM

Brute Force ($O(n^2)$)
- ✓ Dùng hai vòng lặp
- ✓ Kiểm tra mọi cặp / subarray
- ✓ Gây ra Time Limit Exceeded (TLE)

→

Tối ưu (O(n))
- ✓ Dùng HashMap / Prefix Sum
- ✓ Lưu lại thông tin đã thấy trước đó
- ✓ Gộp vòng lặp lồng nhau thành một vòng

PREFIX SUM CƠ BẢN

```
prefix[i] = tổng phần tử từ 0 đến i

Tổng subarray từ i đến j:
sum(i,j) = prefix[j] - prefix[i-1]

# Giúp trả lời truy vấn range/subarray nhanh
```

HASHING CƠ BẢN

	Key	Value
HashMap / Set	Key1	Value1
	Key2	Value2

- ✓ Tra cứu / chèn gần như O(1)
- ✓ Rất hữu ích để đếm, kiểm tra tồn tại, gom nhóm

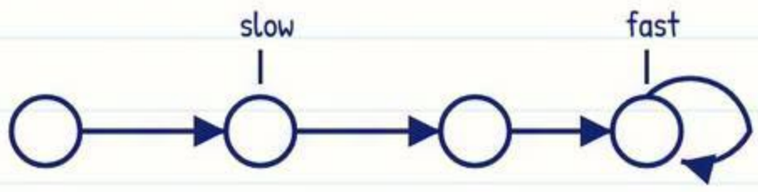
5. LINKED LIST – CÂU HỎI THƯỜNG GẶP NHẤT

★ Câu hỏi Linked List kiểm tra khả năng thao tác con trỏ và xử lý edge case.
Kỹ thuật Fast & Slow pointer được dùng trong rất nhiều bài toán.

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Reverse Linked List (LC 206)	Iterative (Three Pointers)	O(n) time O(1) space	• Giữ prev, curr, next • Đảo ngược liên kết từng bước • Trả về prev ở cuối
2	Linked List Cycle (LC 141)	Fast & Slow Pointers (Floyd's)	O(n) time O(1) space	• Di chuyển slow 1 bước, fast 2 bước • Nếu gặp nhau → có chu trình • Nếu fast chạm cuối → không có chu trình
3	Middle of Linked List (LC 876)	Fast & Slow Pointers	O(n) time O(1) space	• Di chuyển slow 1 bước, fast 2 bước • Khi fast chạm cuối, • slow sẽ ở giữa danh sách
4	Merge Two Sorted Lists (LC 21)	Two Pointers	O(n + m) time O(1) space	• So sánh node của cả hai danh sách • Gắn node nhỏ hơn vào kết quả • Di chuyển con trỏ của danh sách đó
5	Remove Nth Node From End (LC 19)	Two Pointers (Gap Technique)	O(n) time O(1) space	• Di chuyển fast trước n bước • Di chuyển cả hai đến khi fast chạm cuối • Xóa slow.next
6	Intersection of Two Linked Lists (LC 160)	Two Pointers (Switch Lists)	O(n + m) time O(1) space	• Duyệt qua cả hai danh sách • Chuyển sang head kia khi chạm cuối • Sẽ gặp nhau tại giao điểm hoặc NULL

★ FAST & SLOW POINTER – KHI NÀO DÙNG?

Dùng Fast & Slow khi:
- ✓ Phát hiện chu trình trong Linked List
- ✓ Tìm phần tử ở giữa
- ✓ Tìm điểm bắt đầu chu trình
- ✓ Tìm node thứ k từ cuối



Tại sao nó hoạt động?
- ✓ Fast đi 2x tốc độ, slow đi 1x
- ✓ Nếu có chu trình, fast sẽ đuổi kịp slow
- ✓ Nếu không có chu trình, fast sẽ chạm NULL

ĐỘ PHỨC TẠP CÁC PHÉP TOÁN LINKED LIST

- Traverse → O(n)
- Search → O(n)
- Insert at Head → O(1)
- Delete at Head → O(1)
- Insert at Tail → O(n)
- Delete at Tail → O(n)
- Insert at Position → O(n)
- Delete at Position → O(n)

CẤU TRÚC NODE

6. STACKS + QUEUES – CÂU HỎI THƯỜNG GẶP NHẤT

★ Bài toán Stack & Queue kiểm tra khả năng hiểu LIFO, FIFO và tư duy monotonic.

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Valid Parentheses (LC 20)	Stack	O(n) time O(n) space	• Push các dấu ngoặc mở • Với ngoặc đóng, kiểm tra đỉnh stack • Stack phải rỗng ở cuối
2	Min Stack (LC 155)	Stack (Extra Space)	O(1) time cho push, pop, top, getMin O(n) space	• Dùng thêm 1 stack lưu giá trị min • Hoặc lưu cặp (val, current_min) • Mọi thao tác đều O(1)
3	Next Greater Element I (LC 496)	Monotonic Stack	O(n) time O(n) space	• Dùng stack giảm dần • Liên tục pop các phần tử nhỏ hơn • Map phần tử lớn hơn tiếp theo
4	Daily Temperatures (LC 739)	Monotonic Stack	O(n) time O(n) space	• Lưu index vào stack • Pop khi current > nhiệt độ ở đỉnh • Đáp án = current_index - pop_index
5	Implement Queue Using Stacks (LC 232)	Two Stacks	Amortized O(1) time O(n) space	• Push → đẩy vào stack1 • Pop → chuyển sang stack2 nếu rỗng • Pop lấy từ stack2
6	Implement Stack Using Queues (LC 225)	Two Queues	O(1) time (push) O(n) time (pop) O(n) space	• Push → enqueue rồi xoay vòng • Phần tử mới nhất luôn ở đầu • Pop chỉ đơn giản là dequeue
7	Evaluate Reverse Polish Notation (LC 150)	Stack	O(n) time O(n) space	• Nếu là số → push vào stack • Nếu là toán tử → pop hai giá trị, tính toán • Push kết quả trở lại
8	Largest Rectangle in Histogram (LC 84)	Monotonic Stack	O(n) time O(n) space	• Dùng stack tăng dần • Tính diện tích khi pop phần tử • Thêm sentinel 0 ở cuối

★ CÁC PATTERN STACK / QUEUE PHỔ BIẾN

→ Monotonic Increasing Stack
→ Monotonic Decreasing Stack
→ Stack Simulation
→ Two Stack / Two Queue
→ Expression Evaluation
→ Parentheses Matching
→ Next Greater / Smaller Element
→ Sliding Window Maximum
→ BFS (Using Queue)

7. BINARY SEARCH – CÂU HỎI THƯỜNG GẶP NHẤT

★ Binary Search giảm không gian tìm kiếm từ O(n) xuống O(log n). Luôn tư duy theo dữ liệu đã sắp xếp / điều kiện đơn điệu.

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Binary Search (LC 704)	Binary Search	O(log n) O(1) space	• Tìm kiếm trong mảng đã sắp xếp • So sánh mid và di chuyển phạm vi
2	Find First and Last Position (LC 34)	Binary Search (Left & Right)	O(log n) O(1) space	• Tìm vị trí trái nhất và phải nhất • Dùng binary search cải tiến
3	Search in Rotated Sorted Array (LC 33)	Binary Search (Rotated Array)	O(log n) O(1) space	• Tìm nửa đã sắp xếp, sau đó tìm kiếm • Xử lý điểm xoay (pivot) chính xác
4	Find Peak Element (LC 162)	Binary Search (Peak)	O(log n) O(1) space	• So sánh mid với phần tử lân cận • Đỉnh (peak) lớn hơn các phần tử lân cận
5	Square Root of x (LC 69)	Binary Search (Answer Space)	O(log n) O(1) space	• Tìm căn bậc hai nguyên • Kiểm tra mid*mid <= x
6	Search in Answer Space (Binary Search on Range)	Binary Search (Answer Space)	O(log R) O(1) space	• Binary search trên khoảng đáp án • Hữu ích trong bài toán tối ưu hoá

★ MẪU BINARY SEARCH

```
left = start, right = end
while left <= right:
    mid = left + (right-left)//2
    if condition(mid):
        # đáp án khả dĩ
        move left/right
    else:
        move the other way
return final answer
```

KHI NÀO DÙNG BINARY SEARCH

✓ Mảng đã được sắp xếp
✓ Có thể định nghĩa điều kiện đơn điệu
✓ Khi không gian đáp án có thứ tự (min/max khả dĩ)

MẸO THƯỜNG DÙNG

✓ Left = mid + 1 (tìm ở nửa phải)
✓ Right = mid - 1 (tìm ở nửa trái)
✓ Cẩn thận điều kiện để tránh vòng lặp vô hạn
✓ Dùng long/64-bit khi phạm vi số lớn

EDGE CASES

✓ Mảng rỗng
✓ Mảng chỉ có một phần tử
✓ Tất cả phần tử giống nhau
✓ Target nhỏ hơn/lớn hơn mọi phần tử

8. TREES & BST – CÂU HỎI THƯỜNG GẶP NHẤT

★ Trees là cấu trúc dữ liệu phân cấp. BST có tính chất đặc biệt: trái < gốc < phải.

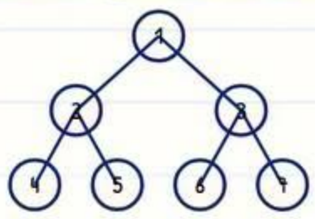
1 Tree Traversals (Pre, In, Post Order)
2 Maximum Depth of Binary Tree
3 Level Order Traversal
5 Check if Balanced Binary Tree
6 Validate Binary Search Tree
7 Lowest Common Ancestor (LCA)

8. TREES & BST – CÂU HỎI THƯỜNG GẶP NHẤT (CHI TIẾT)

★ Trees là cấu trúc phân cấp. BST tuân theo: trái < gốc < phải. Nắm vững các phép duyệt cây + pattern cây phổ biến.

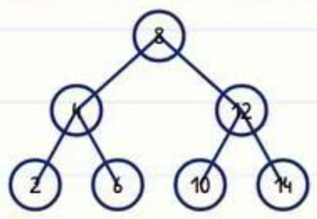
#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Tree Traversals (Pre, In, Post Order) (LC 144,145,94)	DFS (Recursion / Stack)	O(n) time O(h) space	• Pre: Root → Left → Right • In: Left → Root → Right • Post: Left → Right → Root
2	Maximum Depth of Binary Tree (LC 104)	DFS (Recursion)	O(n) time O(h) space	• Height = 1 + max(left, right) • Base: null → 0
3	Level Order Traversal (LC 102)	BFS (Queue)	O(n) time O(w) space (w = độ rộng lớn nhất)	• Dùng queue • Đẩy các con theo từng cấp • Theo dõi kích thước mỗi cấp
4	Diameter of Binary Tree (LC 543)	DFS (Post Order)	O(n) time O(h) space	• Diameter = đường đi dài nhất • Truyền height lên từ node con • Cập nhật max toàn cục
5	Check if Balanced Binary Tree (LC 110)	DFS (Bottom Up)	O(n) time O(h) space	• Chênh lệch height trái & phải phải <= 1 • Nếu subtree nào mất cân bằng, trả -1
6	Validate Binary Search Tree (LC 98)	DFS (Inorder)	O(n) time O(h) space	• Inorder của BST luôn được sắp xếp • Dùng con trỏ prev để kiểm tra current > prev
7	Lowest Common Ancestor (LCA) (LC 236)	DFS	O(n) time O(h) space	• Nếu root null → trả về null • Nếu root == p hoặc q → trả về root • Nếu trái & phải đều khác null → trả root

MINH HOẠ CÁC PHÉP DUYỆT CÂY



Preorder (Root, Left, Right)
→ 1, 2, 4, 5, 3, 6, 7
Inorder (Left, Root, Right)
→ 4, 2, 5, 1, 6, 3, 7
Postorder (Left, Right, Root)
→ 4, 5, 2, 6, 7, 3, 1
Level Order (BFS)
→ 1, 2, 3, 4, 5, 6, 7

TÍNH CHẤT CỦA BST



Với mọi node:

✓ Trái < Node

✓ Phải > Node

Inorder của BST luôn cho kết quả đã được sắp xếp.

DFS VS BFS

DFS (Depth First Search)

✓ Đi sâu trước

✓ Dùng Stack / Đệ quy

✓ Tốt cho path, components, backtracking, LCA

BFS (Breadth First Search)

✓ Duyệt theo từng cấp

✓ Dùng Queue

✓ Tốt cho shortest path, level order, minimum steps

BẢNG ĐỘ PHỨC TẠP THỜI GIAN & KHÔNG GIAN

n = số lượng node

h = chiều cao của cây (worst case O(n), best case O(log n))

CÁC PATTERN CÂY PHỔ BIẾN

✓ Traverse everything → DFS / BFS

✓ Search / validate → DFS

✓ Level-based → BFS

✓ Path / Ancestor → DFS

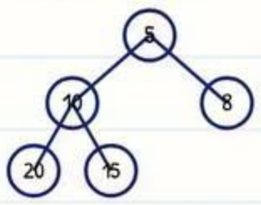
9. HEAPS (PRIORITY QUEUE) - CÂU HỎI THƯỜNG GẶP NHẤT

★ Heap là cây nhị phân hoàn chỉnh thỏa mãn tính chất heap.
Dùng để hiện thực Priority Queue một cách hiệu quả.

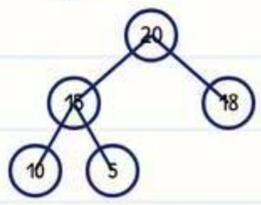
#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Kth Largest Element in Array (LC 215)	Heap (Min Heap)	O(n log k) time O(k) space	• Dùng Min Heap kích thước k • Push k phần tử đầu tiên • Với phần còn lại, thay nếu lớn hơn • Đỉnh heap là kth largest
2	K Closest Points to Origin (LC 973)	Heap (Max Heap)	O(n log k) time O(k) space	• Dùng Max Heap kích thước k • Push k điểm gần nhất theo khoảng cách • Thay nếu điểm mới gần hơn
3	Merge K Sorted Lists (LC 23)	Heap (Min Heap)	O(N log k) time O(k) space (N = tổng số node)	• Push head của mọi list vào Min Heap • Pop min, push tiếp phần tử kế của list đó • Lặp lại đến khi heap rỗng
4	Find Median from Data Stream (LC 295)	Two Heaps (Max + Min)	O(log n) time O(n) space	• Max Heap cho nửa nhỏ hơn • Min Heap cho nửa lớn hơn • Cân bằng kích thước, lấy trung vị
5	Top K Frequent Elements (LC 347)	Heap (Min Heap)	O(n log k) time O(k) space	• Đếm tần suất bằng Map • Push (freq, phần tử) vào Min Heap • Giữ kích thước = k
6	Sort Nearly Sorted Array (LC 1046)	Heap (Min Heap)	O(n log k) time O(k) space	• Các phần tử lệch vị trí tối đa k • Dùng Min Heap kích thước k+1 • Pop min và push phần tử tiếp theo

CÁC LOẠI HEAP

MIN HEAP
- ✓ Parent ≤ Child
- ✓ Phần tử nhỏ nhất ở gốc



MAX HEAP
- ✓ Parent ≥ Child
- ✓ Phần tử lớn nhất ở gốc



ĐỘ PHỨC TẠP CÁC THAO TÁC HEAP

Insert (push)	**O(log n)**
Delete Top (pop)	**O(log n)**
Peek / Top	**O(1)**
Build Heap	**O(n)**
Heapify	**O(log n)**

KHI NÀO DÙNG HEAP?

- ✓ K phần tử lớn nhất/nhỏ nhất
- ✓ Merge K danh sách đã sắp xếp
- ✓ Xử lý theo độ ưu tiên
- ✓ Trung vị của luồng dữ liệu (streaming)
- ✓ Lập lịch / mô phỏng
- ✓ Thuật toán Dijkstra
- ✓ Mã hoá Huffman

HIỆN THỰC HEAP BẰNG MẢNG (ARRAY)

Với node tại index i (0-indexed):

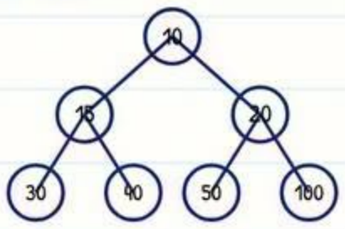
Left Child = 2i + 1

Right Child = 2i + 2

Parent = (i - 1) / 2

Index	0	1	2	3	4	5	6
Value	10	15	20	30	40	50	100

BIỂU DIỄN DẠNG CÂY



10. GRAPHS – CÂU HỎI THƯỜNG GẶP NHẤT

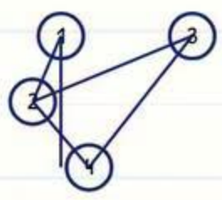
★ Graphs dùng để biểu diễn mối quan hệ giữa các thực thể. Các loại: Directed / Undirected, Weighted / Unweighted

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Number of Islands (LC 200)	DFS / BFS (Grid Graph)	O(m x n) time O(m x n) space	• Duyệt cả 4 hướng • Đánh dấu ô đã thăm
2	Clone Graph (LC 133)	DFS / BFS (Graph Traversal)	O(V + E) time O(V) space	• Dùng HashMap lưu node đã clone • Tránh thăm lại node cũ
3	Detect Cycle in Undirected Graph (LC 547)	DFS / Union Find	O(V + E) time O(V) space	• DFS: kiểm tra node cha • Union Find: chu trình khi union trùng
4	Course Schedule (LC 207)	BFS (Topological Sort - Kahn's Algo)	O(V + E) time O(V) space	• Xây mảng in-degree • Nếu mọi node được xử lý → khả thi, ngược lại không
5	Minimum Spanning Tree (Prim's/Kruskal's)	Greedy Algorithm	O(E log V) (dùng PQ)	• MST kết nối mọi node với • tổng trọng số nhỏ nhất
6	Dijkstra's Algorithm (Shortest Path)	Greedy + PQ	O((V + E) log V)	• Dùng Min Heap • Chỉ hoạt động với trọng số không âm
7	Topological Sort (LC 210)	BFS / DFS	O(V + E) time O(V) space	• Dùng cho DAG (Directed Acyclic Graph) • Thứ tự của các môn học / tác vụ

BIỂU DIỄN ĐỒ THỊ

- ✓ Adjacency List: tiết kiệm bộ nhớ
- ✓ Adjacency Matrix: tra cứu nhanh hơn

1 → [2,3] 2 → [1,4]
3 → [1] 4 → [2,3]



	1	2	3	4
1	0	1	1	0
2	1	0	0	1
3	1	0	0	0
4	0	1	1	0

CÁC PHÉP DUYỆT ĐỒ THỊ PHỔ BIẾN

BFS (Level Order)
Dùng Queue
Duyệt theo từng cấp

DFS (Depth First)
Dùng Stack / Đệ quy
Đi sâu hết mức có thể

PHÁT HIỆN CHU TRÌNH (ĐỒ THỊ VÔ HƯỚNG)

Cách tiếp cận DFS:

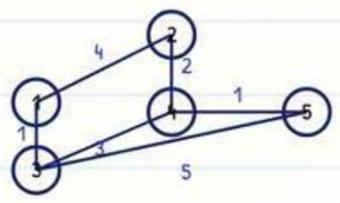
- ✓ Nếu node kề đã thăm không phải cha → có chu trình
- ✓ Theo dõi node cha trong lúc DFS

TOPOLOGICAL SORT (THUẬT TOÁN KAHN)

- ✓ Tính in-degree của mọi node
- ✓ Đẩy các node có in-degree 0 vào queue
- ✓ Pop khỏi queue, thêm vào kết quả
- ✓ Giảm in-degree của các node kề
- ✓ Lặp lại đến khi queue rỗng

DIJKSTRA'S ALGORITHM (ĐƯỜNG ĐI NGẮN NHẤT)

- ✓ Khởi tạo distance mọi node = ∞, source = 0
- ✓ Dùng Min Heap chọn node có distance nhỏ nhất
- ✓ Cập nhật distance của các node kề
- ✓ Lặp lại đến khi mọi node đã xử lý



Khoảng cách ngắn nhất từ node 1:
1 → 0, 2 → 2 (1 → 3 → 2), 3 → 1 (1 → 3), 4 → 4 (1 → 3 → 2 → 4)

11. DYNAMIC PROGRAMMING (DP) – CÂU HỎI THƯỜNG GẶP NHẤT

★ DP là kỹ thuật giải bài toán bằng cách chia nhỏ thành các bài toán con chồng lấn. Các bước: Xác định state → Viết công thức truy hồi → Chọn cách tiếp cận (Top-Down/Bottom-Up)

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Climbing Stairs (LC 70)	1D DP (Fibonacci)	O(n) time O(1)/O(n) space	• dp[i] = số cách để đến bậc i • dp[i] = dp[i-1] + dp[i-2]
2	House Robber (LC 198)	1D DP	O(n) time O(1) space	• Ở mỗi bước, chọn hoặc bỏ qua • dp[i] = max(dp[i-1], dp[i-2]+nums[i])
3	Longest Increasing Subsequence (LC 300)	1D DP / Binary Search	O(n²) time O(n log n)	• dp[i] = độ dài LIS kết thúc tại i • Tối ưu về O(n log n)
4	Edit Distance (LC 72)	2D DP (Strings)	O(m x n) time O(m x n) space	• Insert / Delete / Replace • dp[i][j] = min của 3 phép toán
5	0/1 Knapsack (LC 416)	2D / 1D DP	O(n x W) time O(W) space	• dp[i][w] = giá trị lớn nhất dùng • i vật đầu tiên với sức chứa w
6	Coin Change 2 (LC 518)	2D DP	O(n x amount) O(amount) space	• Đếm số cách tạo thành amount • Thứ tự không quan trọng
7	Longest Common Subsequence (LC 1143)	2D DP (Strings)	O(m x n) time O(m x n) space	• Nếu ký tự khớp → +1 • Ngược lại → max của top/left
8	Partition Equal Subset Sum (LC 416)	Subset Sum DP	O(n x sum) time O(sum) space	• Kiểm tra xem sum/2 có khả thi • Boolean DP (true/false)

CÁC CÁCH TIẾP CẬN DP

1. Top-Down (Memoization)

- ✓ Dùng đệ quy + mảng memo
- ✓ Dễ viết & dễ hiểu hơn
- ✓ Phù hợp với sparse states

2. Bottom-Up (Tabulation)

- ✓ Xây dựng từ base case
- ✓ Thường nhanh hơn (không đệ quy)
- ✓ Được ưu tiên trong phỏng vấn

CÁC PATTERN DP PHỔ BIẾN

- ✓ 1D DP: Dãy tuyến tính, lựa chọn
- ✓ 2D DP: Lưới, chuỗi, khoảng
- ✓ Knapsack Pattern: Tối đa hoá giá trị
- ✓ Subsequence Pattern: LIS, LCS
- ✓ Partition Pattern: Subset Sum, Equal Partition
- ✓ DP trên cây: Tree DP, Path DP

VÍ DỤ 1: CLIMBING STAIRS (DP)

Bài toán: Cho n bậc thang, mỗi bước được leo 1 hoặc 2 bậc. Tìm số cách để lên đến bậc thứ n.

Công thức: dp[i] = dp[i-1] + dp[i-2]
Base: dp[0] = 1, dp[1] = 1
Đáp án: dp[n]

Bottom-Up (Tabulation)

```
dp[0] = 1, dp[1] = 1
for i from 2 to n:
    dp[i] = dp[i-1] + dp[i-2]
return dp[n]
```

i	0	1	2	3	4	5
dp[i]	1	1	2	3	5	8

Dry Run (n = 5) → Đáp án = 8

VÍ DỤ 2: EDIT DISTANCE (DP)

Bài toán: Số thao tác tối thiểu để chuyển word1 thành word2 (Insert/Delete/Replace)

word1 = "horse", word2 = "ros"

```
if word1[i-1] == word2[j-1]:
    dp[i][j] = dp[i-1][j-1]
else:
    dp[i][j] = 1 + min(
        dp[i-1][j],   # xoá
        dp[i][j-1],   # thêm
        dp[i-1][j-1]  # thay
```

	-	r	o	s
-	0	1	2	3
h	1	1	2	3
o	2	2	1	2
r	3	2	2	2
s	4	3	3	2
e	5	4	4	3

12. GRAPH ALGORITHMS – CÂU HỎI THƯỜNG GẶP NHẤT

★ Graph algorithms giúp tìm đường đi ngắn nhất, chi phí tối thiểu, tính liên thông. Cần biết khi nào dùng thuật toán nào!

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Shortest Path in Unweighted Graph (LC 1091)	BFS	O(V + E) time O(V) space	• BFS từ node nguồn • Lần đầu thăm = đường đi ngắn nhất
2	Shortest Path in Weighted Graph (LC 743)	Dijkstra's Algorithm (Min Heap)	O((V+E) log V) O(V) space	• Trọng số không âm • Greedy: luôn chọn node có distance nhỏ nhất
3	Detect Negative Cycle (LC 209)	Bellman-Ford Algorithm	O(V x E) time O(V) space	• Relax mọi cạnh V-1 lần • Relax thêm 1 lần mà vẫn cải thiện → có chu trình
4	Find All Bridges in Graph (LC 1192)	Tarjan's Algorithm (DFS + Low Time)	O(V + E) time O(V) space	• Cạnh (u,v) là cầu nếu • low[v] > disc[u]
5	Find Articulation Points (LC 1125)	Tarjan's Algorithm (DFS + Low Time)	O(V + E) time O(V) space	• Đỉnh mà khi xoá làm tăng • số thành phần liên thông
6	Minimum Spanning Tree (MST) (LC 1584)	Kruskal's / Prim's Algorithm	O(E log E) hoặc O((V+E) log V)	• MST kết nối mọi node với • tổng trọng số nhỏ nhất
7	Check if Graph is Bipartite (LC 785)	BFS / DFS (Coloring)	O(V + E) time O(V) space	• Thử tô màu bằng 2 màu • Nếu xung đột → không phải bipartite

ĐƯỜNG ĐI NGẮN NHẤT – TỔNG QUAN NHANH

- ✓ Đồ thị không trọng số → BFS
- ✓ Có trọng số (không âm) → Dijkstra
- ✓ Có trọng số âm → Bellman-Ford
- ✓ Chỉ cần biết có thể đến được → DFS/BFS

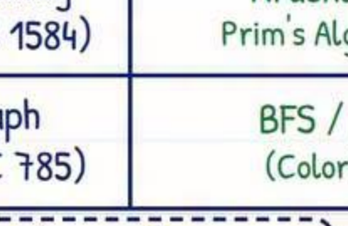
DIJKSTRA (MIN HEAP) – Ý TƯỞNG

- ✓ Lưu mảng distance[]
- ✓ Min Heap lưu (dist, node)
- ✓ Pop node có dist nhỏ nhất
- ✓ Relax mọi node kề

BELLMAN-FORD – Ý TƯỞNG

- ✓ Relax mọi cạnh V-1 lần
- ✓ Nếu vẫn relax được ở lần V → có chu trình âm
- ✓ Hoạt động với trọng số âm

VÍ DỤ DIJKSTRA



Nguồn = A — Init: A=0, B=∞, C=∞, D=∞, E=∞

Chọn A(0) → relax: B=4, C=1

Chọn C(1) → relax: B=3, E=5

Chọn B(3) → relax: D=4

Chọn D(4) → relax: E=5 (không đổi)

Chọn E(5) → xong

Distance cuối: A=0, B=3, C=1, D=4, E=5

KRUSKAL VS PRIM

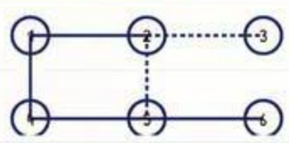
Kruskal (dựa trên cạnh)

- ✓ Sắp xếp mọi cạnh
- ✓ Thêm cạnh nhỏ nhất
- ✓ Bỏ qua nếu tạo thành chu trình
- ✓ Dùng Disjoint Set (DSU)

Prim (dựa trên node)

- ✓ Bắt đầu từ node bất kỳ
- ✓ Mở rộng MST bằng cạnh nhỏ nhất
- ✓ Dùng Min Heap

TARJAN – TÌM CẦU (BRIDGE)

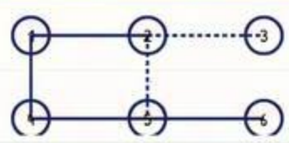


Cạnh (u,v) là cầu nếu:

low[v] > disc[u]

Ở đây, cạnh (1,4) và (3,6) là cầu

TARJAN – ĐIỂM KHỚP (ARTICULATION)



Node u là điểm khớp nếu:

low[v] ≥ disc[u]

(v là con của u trong cây DFS)

13. GREEDY ALGORITHMS – CÂU HỎI THƯỜNG GẶP NHẤT

★ Greedy chọn lựa chọn tốt nhất ở mỗi bước với hy vọng đạt tối ưu toàn cục. Hoạt động khi bài toán có tính chất Greedy Choice + Optimal Substructure.

#	CÂU HỎI	PATTERN	ĐỘ PHỨC TẠP DỰ KIẾN	Ý TƯỞNG CHÍNH / GỢI Ý
1	Activity Selection (LC 435)	Greedy (Sort by End Time)	O(n log n) time O(1) space	• Sắp xếp hoạt động theo thời gian kết thúc • Chọn hiện tại, rồi chọn tiếp start >= last end
2	Fractional Knapsack (LC 1029)	Greedy (Sort by Value/Weight)	O(n log n) time O(1) space	• Sắp xếp theo tỉ lệ value/weight giảm dần • Lấy phần lẻ nếu vật không vừa hết
3	Minimum Number of Coins (LC 322)	Greedy (Sort Coins Desc)	O(n + amount) O(1) space	• Chọn đồng xu lớn nhất <= amount • Lặp lại cho phần còn lại
4	Jump Game II (LC 45)	Greedy (Level by Level)	O(n) time O(1) space	• Mở rộng phạm vi xa nhất có thể đến ở mỗi bước • Đếm số bước nhảy
5	Merge Intervals (LC 56)	Greedy (Sort by Start)	O(n log n) time O(n) space	• Sắp xếp theo điểm bắt đầu • Gộp lại nếu chồng lấn
6	Minimum Cost to Connect Ropes (LC 1167)	Greedy + Min Heap	O(n log n) time O(n) space	• Luôn nối 2 sợi dây ngắn nhất trước • Đẩy tổng trở lại heap
7	Gas Station (LC 134)	Greedy	O(n) time O(1) space	• Nếu tổng gas < tổng cost → -1 • Ngược lại, tìm điểm bắt đầu hợp lệ

1. VÍ DỤ ACTIVITY SELECTION

Hoạt động (start, end):
(1,4) (3,5) (0,6) (5,7) (3,9) (5,10)
Sắp xếp theo end:
(1,4) (3,5) (0,6) (6,10) (3,9) (5,9)
Chọn: (1,4) → (5,7) → (8,11)
Số hoạt động tối đa = 3

2. VÍ DỤ FRACTIONAL KNAPSACK

Capacity = 50
Item 1: w=10 v=60 (tỉ lệ 6.0)
Item 2: w=20 v=100 (tỉ lệ 5.0)
Item 3: w=30 v=120 (tỉ lệ 4.0)
Lấy item 1 (10) → val=60
Lấy item 2 (20) → val=100 (tổng w=30)
Lấy 20/30 của item 3 → val = 80
Tổng giá trị = 60+100+80 = 240

4. VÍ DỤ JUMP GAME II

nums = [2,3,1,1,4]
Index: 0 1 2 3 4
Nhảy 1: từ 0 → đến được [1,2]
Nhảy 2: từ [1,2] → đến được [3,4]
Nhảy 3: từ [3,4] → chạm đích
Số bước nhảy tối thiểu = 2

5. VÍ DỤ MERGE INTERVALS

Intervals: [1,3] [2,6] [8,10] [15,18]
Sắp xếp theo start: (đã sắp xếp)
Gộp: [1,3]+[2,6] → [1,6]
[8,10] và [15,18] không chồng lấn
Kết quả: [1,6] [8,10] [15,18]

6. VÍ DỤ CONNECT ROPES

Ropes: [4, 3, 2, 6]
Bước 1: lấy 2+3=5 (chi phí=5) → heap [4,5,6]
Bước 2: lấy 4+5=9 (chi phí=14) → heap [6,9]
Bước 3: lấy 6+9=15 (chi phí=29)
Tổng chi phí tối thiểu = 29

7. VÍ DỤ GAS STATION

gas = [1, 2, 3, 4, 5]
cost = [3, 4, 5, 1, 2]
Index: 0 1 2 3 4
Diff: -2 -2 -2 3 3
Tổng = 0 → Khả thi
Điểm bắt đầu = index 3

GREEDY CHOICE PROPERTY

- ✓ Lựa chọn cục bộ tốt nhất dẫn đến lời giải tối ưu toàn cục
- ✓ Các lựa chọn không ảnh hưởng lẫn nhau

KHI NÀO DÙNG GREEDY?

- ✓ Activity / Interval Scheduling
- ✓ Bài toán tối ưu hoá đơn giản
- ✓ Có thể sắp xếp theo tiêu chí rõ ràng

MẸO GREEDY

- ✓ Chứng minh tính đúng đắn (nếu có thể)
- ✓ Sắp xếp trước theo tiêu chí phù hợp
- ✓ Cẩn thận với các trường hợp phản ví dụ

CHECKLIST NHANH

- ✓ Có thể đưa ra lựa chọn tốt nhất cục bộ không?
- ✓ Bài toán có Optimal Substructure không?
- ✓ Đã thử phản ví dụ (counter-example) chưa?