CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

MỤC LỤC

Giới thiệu cấu trúc dữ liệu và giải thuật

DSA là gì?

DSA viết tắt của **Data Structures (Cấu trúc dữ liệu)** và **Algorithms (Giải thuật)**.

Cấu trúc dữ liệu: Quản lý cách dữ liệu được lưu trữ và truy cập.

Ví dụ: Mảng (Array), Danh sách liên kết (Linked List), Cây (Tree), Heap.

Giải thuật: Tập trung vào các bước xử lý dữ liệu đó để giải quyết một bài toán.

Ví dụ: Tìm kiếm nhị phân (Binary Search), Sắp xếp nhanh (Quick Sort), Sắp xếp trộn (Merge Sort).

Tại sao DSA lại quan trọng?

Nền tảng của công nghệ: Là xương sống cho hầu hết các phần mềm như GPS, Công cụ tìm kiếm (Google), AI ChatBots (ChatGPT), Game, Cơ sở dữ liệu, Ứng dụng Web...

Cơ hội nghề nghiệp: Các tập đoàn công nghệ hàng đầu (Big Tech) như Google, Microsoft, Amazon, Apple, Meta... luôn chú trọng kiểm tra kiến thức DSA trong các buổi phỏng vấn.

Tư duy lập trình: Học DSA giúp nâng cao khả năng giải quyết vấn đề và giúp bạn trở thành một lập trình viên giỏi hơn.

Lộ trình học DSA từng bước (Step-by-Step)

1. Xây dựng tư duy Logic (Logic Building): Sau khi học cơ bản về ngôn ngữ, bước đầu tiên là rèn luyện tư duy logic thông qua các bài toán lập trình cơ bản.

2. Độ phức tạp (Complexity Analysis): Để phân tích một giải thuật, chúng ta đo lường tốc độ tăng trưởng của **thời gian (Time Complexity)** hoặc **không gian (Space Complexity)** dựa trên kích thước đầu vào. Thông thường, chúng ta tập trung vào trường hợp xấu nhất (Worst case).

3. Mảng (Array): Mảng là cấu trúc dữ liệu tuyến tính, nơi các phần tử được cấp phát bộ nhớ liên tục, cho phép truy cập hằng số thời gian thông qua chỉ số.

4. Thuật toán Tìm kiếm (Searching Algorithms): Được sử dụng để tìm một giá trị cụ thể trong một tập dữ liệu lớn. Có nhiều loại thuật toán tìm kiếm với cách tiếp cận và hiệu suất khác nhau.

5. Thuật toán Sắp xếp (Sorting Algorithms): Dùng để sắp xếp các phần tử theo một thứ tự nhất định (tăng dần, giảm dần, bảng chữ cái...). Việc sắp xếp giúp việc tìm kiếm và truy cập dữ liệu hiệu quả hơn.

6. Băm (Hashing): Kỹ thuật tạo ra một giá trị có kích thước cố định (Hash value) từ dữ liệu đầu vào thông qua các hàm toán học (Hash functions). Hashing cực kỳ hữu ích cho việc tìm kiếm, thêm và xóa dữ liệu nhanh chóng.

7. Kỹ thuật Hai con trỏ (Two Pointer Technique)

Sử dụng hai biến chỉ số (thường từ hai đầu mảng) để duyệt và tìm kiếm một điểm hoặc giá trị thỏa mãn yêu cầu.

8. Kỹ thuật Cửa sổ trượt (Window Sliding Technique):

Sử dụng kết quả của mảng con trước đó để tính toán nhanh kết quả của mảng con hiện tại, giúp giảm thiểu việc tính toán lặp lại.

9. Kỹ thuật Cộng dồn (Prefix Sum Technique): Tính toán trước tổng các phần tử từ đầu mảng đến vị trí hiện tại để truy vấn nhanh tổng của bất kỳ mảng con nào.

10. Chuỗi ký tự (String): Một chuỗi các ký tự, thường là bất biến (immutable) và có một tập hợp các phần tử giới hạn (ví dụ: bảng chữ cái tiếng Anh).

11. Đệ quy (Recursion): Kỹ thuật lập trình mà một hàm **tự gọi lại chính nó**. Thường dùng để giải quyết các bài toán có thể chia nhỏ thành các bài toán con tương tự.

12. Ma trận/Lưới (Matrix/Grid): Mảng hai chiều được sắp xếp theo hàng và cột. Mỗi phần tử nằm tại giao điểm của một hàng và một cột.

13. Danh sách liên kết (Linked List)

Cấu trúc dữ liệu tuyến tính lưu trữ dữ liệu trong các nút (nodes) được kết nối bởi các con trỏ. Các nút không nằm liên tiếp trong bộ nhớ và chỉ có thể truy cập tuần tự từ đầu danh sách (Head).

14. Ngăn xếp (Stack): Hoạt động theo nguyên lý **LIFO (Vào sau, ra trước)**. Stack quan trọng trong việc quản lý lời gọi hàm, bộ nhớ và các bài toán như Next Greater Element.

15. Hàng đợi (Queue): Hoạt động theo nguyên lý **FIFO (Vào trước, ra trước)**. Queue đóng vai trò quan trọng trong việc quản lý tác vụ, điều phối và hệ thống xử lý tin nhắn.

16. Hàng đợi hai đầu (Deque): Cho phép thêm hoặc xóa các phần tử từ cả hai đầu một cách hiệu quả.

17. Cây (Tree): Cấu trúc dữ liệu phân cấp, phi tuyến tính bao gồm các nút được kết nối bởi các cạnh. Nút trên cùng gọi là gốc (Root). Được dùng nhiều trong hệ thống tệp tin, cơ sở dữ liệu.

18. Heap: Một dạng cây nhị phân hoàn chỉnh thỏa mãn thuộc tính Heap. Thường dùng để triển khai hàng đợi ưu tiên (Priority Queue).

19. Đồ thị (Graph): Gồm một tập hợp các đỉnh (Vertices) và các cạnh (Edges) nối các cặp đỉnh đó. Graph dùng để biểu diễn các mối quan hệ thực tế như mạng xã hội, bản đồ giao thông.

20. Giải thuật Tham lam (Greedy Algorithm): Xây dựng giải pháp từng bước và luôn chọn lựa lựa chọn tốt nhất tại thời điểm hiện tại (tối ưu cục bộ) với hy vọng đạt được kết quả tối ưu toàn cục.

21. Quy hoạch động (Dynamic Programming - DP): Giải quyết các bài toán phức tạp bằng cách chia chúng thành các bài toán con đơn giản hơn. Bằng cách giải mỗi bài toán con một lần và lưu lại kết quả, DP tránh được việc tính toán lặp lại.

22. Cấu trúc dữ liệu và Giải thuật nâng cao: Các cấu trúc như **Trie, Segment Tree, Red-Black Tree** và **Binary Indexed Tree** giúp tối ưu hóa hiệu suất cho các bài toán đặc thù như truy vấn đoạn, cập nhật động hoặc xử lý dữ liệu lớn.

23. Các thuật toán khác

- Giải thuật Bitwise: Thao tác trên từng bit của con số.

- Quay lui (Backtracking): Sử dụng đệ quy để thử các trường hợp và quay lại (revert) nếu hướng đi hiện tại không dẫn đến kết quả.

- Chia để trị (Divide and Conquer): Chia bài toán thành các phần nhỏ, giải quyết chúng và kết hợp lại.

- Nhánh và Cận (Branch and Bound): Dùng trong các bài toán tối ưu hóa tổ hợp để tìm kiếm giải pháp tốt nhất một cách hệ thống.

- **Giải thuật Hình học (Geometric):** Giải quyết các bài toán liên quan đến hình dạng, điểm, đường thẳng.

- Giải thuật Ngẫu nhiên (Randomized): Sử dụng tính ngẫu nhiên để đạt được kết quả, thường đơn giản và hiệu quả hơn trong các bài toán xác suất.

Chương 1: TƯ DUY LOGIC

Sau khi học cơ bản về ngôn ngữ, bước đầu tiên là rèn luyện tư duy logic thông qua các bài toán lập trình cơ bản.

Tư duy logic trong lập trình chính là kỹ năng thiết lập quy trình giải quyết vấn đề một cách hệ thống và nhất quán thông qua những nguyên lý cơ bản. Đây được coi là nền tảng cốt lõi, cho phép người lập trình phân tích, biện luận và tìm ra phương án xử lý tối ưu cho mọi bài toán thực tế.

Để rèn luyện khả năng tư duy này, bạn có thể áp dụng lộ trình sau:

1. Phân tích đề bài: Đừng vội bắt tay vào code, hãy dành thời gian để thấu hiểu bản chất yêu cầu.

2. Cụ thể hóa dữ liệu: Thử nghiệm bài toán với nhiều bộ dữ liệu giả định khác nhau để hình dung rõ luồng xử lý.

3. Tìm kiếm quy luật: Từ các ví dụ thực tế, hãy đúc kết ra những điểm chung hoặc mô hình lặp lại.

4. Ưu tiên giải pháp đơn giản: Luôn bắt đầu bằng một phương án khả thi nhất (Brute-force), sau đó mới cải tiến và tối ưu hiệu suất.

Bài tập

Bài tập cơ bản

Bài 1: Kiểm tra Số Chẵn hay Số Lẻ

Yêu cầu bài toán: Cho một số nguyên n, hãy kiểm tra xem số đó là số chẵn hay số lẻ. Trả về giá trị true nếu là số chẵn và false nếu là số lẻ.

Ví dụ 1:

- **Đầu vào (Input):** $n = 15$
- **Đầu ra (Output):** false
- **Giải thích:** Vì $15 \pmod 2 = 1$ (15 chia 2 dư 1), nên 15 là số lẻ.

Ví dụ 2:

- **Đầu vào (Input):** $n = 44$
- **Đầu ra (Output):** true
- **Giải thích:** Vì $44 \pmod 2 = 0$ (44 chia hết cho 2), nên 44 là số chẵn.

Bài tập 2: Viết chương trình in bảng cửu chương

Bài tập 3: Tổng n số tự nhiên

$$S(n)=1+2+3+...+n=\frac{n(n+1)}{2}$$

n	1	5	10	15	100	200
Kết quả	1	15	55	120	5050	20100

Bài tập 4: Tổng n số bình phương đầu tiên

$$S(n)=1^2+2^2+3^2+...+n^2=\frac{n(n+1)(2n+1)}{6}$$

n	1	5	10	15	100	200
Kết quả	1	55	385	1240	338 350	2 686 700

Bài tập 5: Tổng n số lập phương đầu tiên

$$S(n)=1^3+2^3+3^3+...+n^3=\frac{n^2(n+1)^2}{4}$$

n	1	5	10	15	100	200
Kết quả	1	225	3025	14 400	25 502 500	404 010 000

Bài tập 6: Cho số hạng thứ nhất, thứ 2 của cấp số cộng. Tìm số hạng thứ n

Công thức: $u_n=u_1+(n-1)d$

Bài tập 7: Tổng các chữ số

n	1234	55467	1350	15675	100123	122200
Kết quả	10	27	9	24	7	7

Cách 1: Dùng Digit Extraction: $T(n)=O(\log_{10} n)$; $S(n)=O(1)$.

Cách 2: Dùng Recursion

Cách 3: Dùng Converse String

Bài tập 8: Tìm cặp số (a, b) để $a^3+b^3=n$

Cách 1: Duyệt $T(n)=O(n^2)$; $S(n)=O(1)$.

Cách 2: Cải tiến $T(n)=O\left(n^{\frac{1}{3}}\right)$; $S(n)=O(1)$.

Bài tập 9: In ra số Fibonacci thứ n

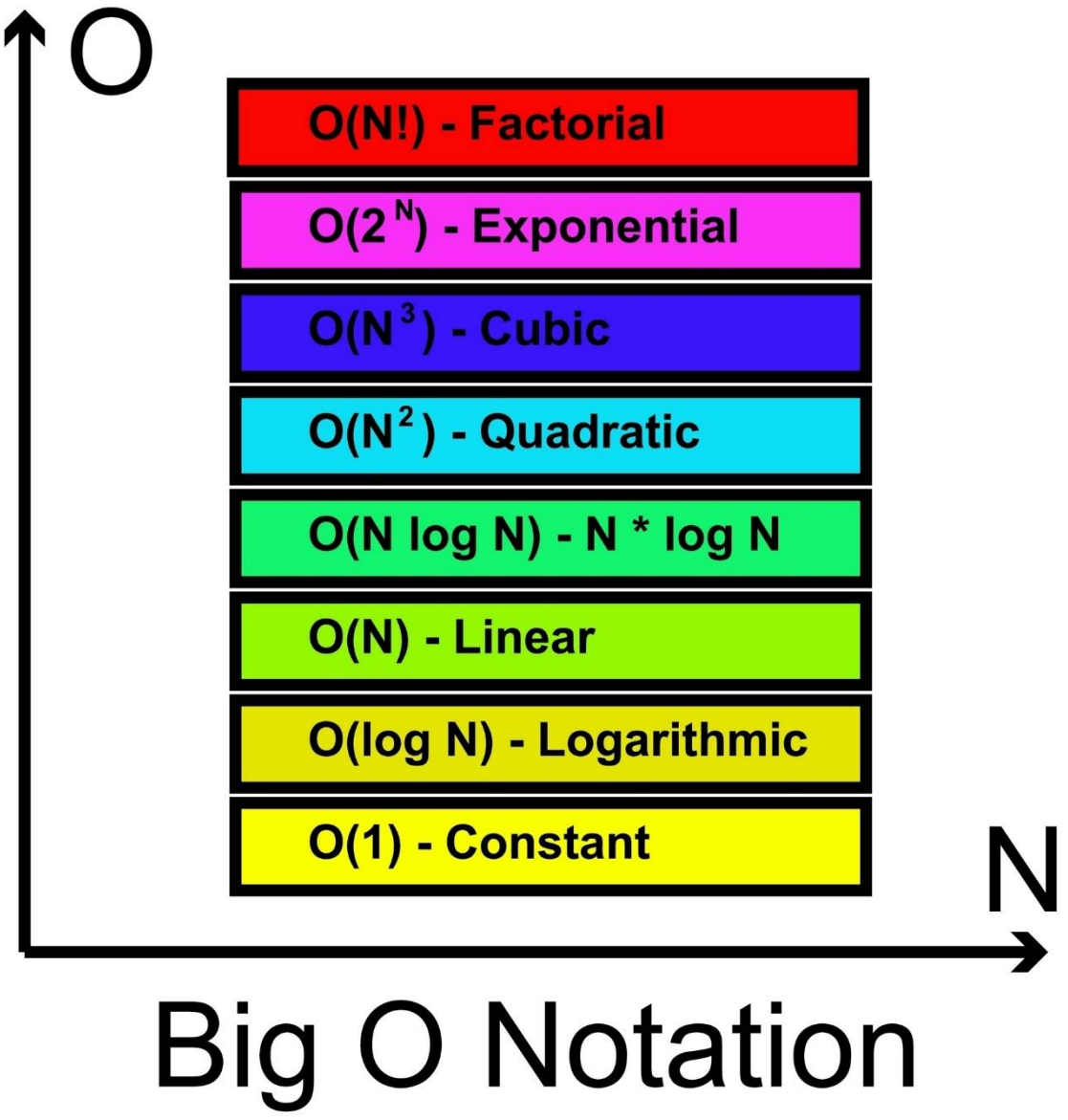
Bài tập 10: Chuyển đổi hệ thập phân sang hệ nhị phân

Bài tập mức khá

Bài tập khó

Chương 2: ĐỘ PHỨC TẠP THUẬT TOÁN

Để phân tích một giải thuật, chúng ta đo lường tốc độ tăng trưởng của **thời gian (Time Complexity)** hoặc **không gian (Space Complexity)** dựa trên kích thước đầu vào. Thông thường, chúng ta tập trung vào trường hợp xấu nhất (Worst case).



Chương 3: Mảng

Mảng là cấu trúc dữ liệu tuyến tính, nơi các phần tử được cấp phát bộ nhớ liên tục, cho phép truy cập hằng số thời gian thông qua chỉ số.

- [Hướng dẫn về Mảng] | [Trắc nghiệm về Mảng]

4. Thuật toán Tìm kiếm (Searching Algorithms)

Được sử dụng để tìm một giá trị cụ thể trong một tập dữ liệu lớn. Có nhiều loại thuật toán tìm kiếm với cách tiếp cận và hiệu suất khác nhau.

- [Hướng dẫn Tìm kiếm] | [Trắc nghiệm Tìm kiếm]

5. Thuật toán Sắp xếp (Sorting Algorithms)

Dùng để sắp xếp các phần tử theo một thứ tự nhất định (tăng dần, giảm dần, bảng chữ cái...). Việc sắp xếp giúp việc tìm kiếm và truy cập dữ liệu hiệu quả hơn.

- [Hướng dẫn Sắp xếp] | [Trắc nghiệm Sắp xếp]

6. Băm (Hashing)

Kỹ thuật tạo ra một giá trị có kích thước cố định (Hash value) từ dữ liệu đầu vào thông qua các hàm toán học (Hash functions). Hashing cực kỳ hữu ích cho việc tìm kiếm, thêm và xóa dữ liệu nhanh chóng.

- [Hướng dẫn Hashing] | [Trắc nghiệm Hashing]

7. Kỹ thuật Hai con trỏ (Two Pointer Technique)

Sử dụng hai biến chỉ số (thường từ hai đầu mảng) để duyệt và tìm kiếm một điểm hoặc giá trị thỏa mãn yêu cầu.

- [Hướng dẫn Hai con trỏ] | [Trắc nghiệm Hai con trỏ]

8. Kỹ thuật Cửa sổ trượt (Window Sliding Technique)

Sử dụng kết quả của mảng con trước đó để tính toán nhanh kết quả của mảng con hiện tại, giúp giảm thiểu việc tính toán lặp lại.

- [Hướng dẫn Cửa sổ trượt] | [Trắc nghiệm Cửa sổ trượt]

9. Kỹ thuật Cộng dồn (Prefix Sum Technique)

Tính toán trước tổng các phần tử từ đầu mảng đến vị trí hiện tại để truy vấn nhanh tổng của bất kỳ mảng con nào.

- [Hướng dẫn Prefix Sum] | [Trắc nghiệm Prefix Sum]

10. Chuỗi ký tự (String)

Một chuỗi các ký tự, thường là bất biến (immutable) và có một tập hợp các phần tử giới hạn (ví dụ: bảng chữ cái tiếng Anh).

- [Hướng dẫn về Chuỗi] | [Trắc nghiệm về Chuỗi]

11. Đệ quy (Recursion)

Kỹ thuật lập trình mà một hàm **tự gọi lại chính nó**. Thường dùng để giải quyết các bài toán có thể chia nhỏ thành các bài toán con tương tự.

- [Hướng dẫn Đệ quy] | [Trắc nghiệm Đệ quy]

12. Ma trận/Lưới (Matrix/Grid)

Mảng hai chiều được sắp xếp theo hàng và cột. Mỗi phần tử nằm tại giao điểm của một hàng và một cột.

- [Hướng dẫn Ma trận] | [Trắc nghiệm Ma trận]

13. Danh sách liên kết (Linked List)

Cấu trúc dữ liệu tuyến tính lưu trữ dữ liệu trong các nút (nodes) được kết nối bởi các con trỏ. Các nút không nằm liên tiếp trong bộ nhớ và chỉ có thể truy cập tuần tự từ đầu danh sách (Head).

- [Hướng dẫn Danh sách liên kết] | [Trắc nghiệm Danh sách liên kết]

14. Ngăn xếp (Stack)

Hoạt động theo nguyên lý **LIFO (Vào sau, ra trước)**. Stack quan trọng trong việc quản lý lời gọi hàm, bộ nhớ và các bài toán như Next Greater Element.

- [Hướng dẫn Stack] | [Trắc nghiệm Stack]

15. Hàng đợi (Queue)

Hoạt động theo nguyên lý **FIFO (Vào trước, ra trước)**. Queue đóng vai trò quan trọng trong việc quản lý tác vụ, điều phối và hệ thống xử lý tin nhắn.

- [Hướng dẫn Queue] | [Trắc nghiệm Queue]

16. Hàng đợi hai đầu (Deque)

Cho phép thêm hoặc xóa các phần tử từ cả hai đầu một cách hiệu quả.

- [Hướng dẫn Deque] | [Trắc nghiệm Deque]

17. Cây (Tree)

Cấu trúc dữ liệu phân cấp, phi tuyến tính bao gồm các nút được kết nối bởi các cạnh. Nút trên cùng gọi là gốc (Root). Được dùng nhiều trong hệ thống tệp tin, cơ sở dữ liệu.

- [Hướng dẫn về Cây] | [Trắc nghiệm về Cây]

18. Heap

Một dạng cây nhị phân hoàn chỉnh thỏa mãn thuộc tính Heap. Thường dùng để triển khai hàng đợi ưu tiên (Priority Queue).

- [Hướng dẫn Heap] | [Trắc nghiệm Heap]

19. Đồ thị (Graph)

Gồm một tập hợp các đỉnh (Vertices) và các cạnh (Edges) nối các cặp đỉnh đó. Graph dùng để biểu diễn các mối quan hệ thực tế như mạng xã hội, bản đồ giao thông.

- [Hướng dẫn Đồ thị] | [Trắc nghiệm Đồ thị]

20. Giải thuật Tham lam (Greedy Algorithm)

Xây dựng giải pháp từng bước và luôn chọn lựa lựa chọn tốt nhất tại thời điểm hiện tại (tối ưu cục bộ) với hy vọng đạt được kết quả tối ưu toàn cục.

- [Hướng dẫn Greedy] | [Trắc nghiệm Greedy]

21. Quy hoạch động (Dynamic Programming - DP)

Giải quyết các bài toán phức tạp bằng cách chia chúng thành các bài toán con đơn giản hơn. Bằng cách giải mỗi bài toán con một lần và lưu lại kết quả, DP tránh được việc tính toán lặp lại.

- [Hướng dẫn DP] | [Trắc nghiệm DP]

22. Cấu trúc dữ liệu và Giải thuật nâng cao

Các cấu trúc như **Trie, Segment Tree, Red-Black Tree** và **Binary Indexed Tree** giúp tối ưu hóa hiệu suất cho các bài toán đặc thù như truy vấn đoạn, cập nhật động hoặc xử lý dữ liệu lớn.

- [Luyện tập nâng cao]

23. Các thuật toán khác

- **Giải thuật Bitwise:** Thao tác trên từng bit của con số.

- **Quay lui (Backtracking):** Sử dụng đệ quy để thử các trường hợp và quay lại (revert) nếu hướng đi hiện tại không dẫn đến kết quả.
- **Chia để trị (Divide and Conquer):** Chia bài toán thành các phần nhỏ, giải quyết chúng và kết hợp lại.
- **Nhánh và Cận (Branch and Bound):** Dùng trong các bài toán tối ưu hóa tổ hợp để tìm kiếm giải pháp tốt nhất một cách hệ thống.
- **Giải thuật Hình học (Geometric):** Giải quyết các bài toán liên quan đến hình dạng, điểm, đường thẳng.
- **Giải thuật Ngẫu nhiên (Randomized):** Sử dụng tính ngẫu nhiên để đạt được kết quả, thường đơn giản và hiệu quả hơn trong các bài toán xác suất.