##### Để trở thành một lập trình viên giỏi, đặc biệt trong các lĩnh vực như lập trình cạnh tranh (competitive programming) hoặc phát triển phần mềm, bạn cần nắm vững một số thuật toán quan trọng. Dưới đây là danh sách các thuật toán cơ bản và nâng cao mà bạn nên học và hiểu rõ:

# Thuật toán sắp xếp (Sorting Algorithms)

##### Sắp xếp là nền tảng cho nhiều thuật toán khác. Khi dữ liệu được sắp xếp, việc tìm kiếm và xử lý sẽ trở nên dễ dàng hơn.

## Thuật toán cơ bản:

### Bubble Sort:

#### Độ phức tạp O(n²).

#### Thực hiện nhiều lần đổi chỗ các phần tử liền kề nếu chúng không theo thứ tự.

#### Ít dùng trong thực tế vì kém hiệu quả, nhưng có thể dùng khi muốn hiểu cách đổi chỗ các phần tử.

#### Mô phỏng: https://www.tiktok.com/@28tech/video/7127603324878097690

### Selection Sort:

#### Độ phức tạp O(n²).

#### Chọn phần tử nhỏ nhất từ danh sách và đặt nó vào vị trí đầu tiên.

#### Hiệu quả hơn Bubble Sort nhưng vẫn không tốt cho dữ liệu lớn.

#### Mô phỏng: https://www.tiktok.com/@28tech/video/7126824842627927323?q=%EF%83%98%09Selection%20Sort&t=1727964849142

### Insertion Sort:

#### Độ phức tạp O(n²).

#### Thích hợp khi dữ liệu gần như đã sắp xếp, vì với mỗi lần chèn, nó chỉ cần duyệt một phần nhỏ của danh sách.

#### Mô phỏng: https://www.tiktok.com/@28tech/video/7128271813649501466?q=%EF%83%98%09Insertion%20Sort%3A%20&t=1727964889315

## Thuật toán nâng cao:

### Merge Sort:

#### Độ phức tạp O(nlog n) - Sử dụng phương pháp chia để trị.

#### Thích hợp cho các tập dữ liệu lớn vì có thời gian chạy ổn định.

#### Không gian (O(n)), nó rất hiệu quả.

#### Mô phỏng: https://www.tiktok.com/@28tech/video/7128342546379590939?q=%EF%83%98%09Merge%20Sort%3A%20&t=1727964911713

### Quick Sort:

#### Độ phức tạp O(nlog n) - Sử dụng phương pháp chia để trị, nhưng có thể tệ nhất là O(n²) nếu chọn pivot không tối ưu.

#### Quick Sort thường nhanh hơn Merge Sort trong thực tế vì giảm thiểu sử dụng bộ nhớ.

#### Mô phỏng: https://www.tiktok.com/@28tech/video/7135396547214298394?q=quick%20sort&t=1727964940447

### Heap Sort:

#### Độ phức tạp O(n log n).

#### Sử dụng heap để sắp xếp, đặc biệt hiệu quả khi cần giới hạn bộ nhớ.

#### Không yêu cầu không gian bổ sung như Merge Sort.

#### Mô phỏng: https://www.tiktok.com/@ongchaulaptrinh/video/7360660365232901396?q=heap%20sort&t=1727964964754

## Sắp xếp tuyến tính:

### Counting Sort:

#### Độ phức tạp O(n+k) với k là giá trị lớn nhất trong mảng.

#### Dùng khi có phạm vi giá trị hẹp (như điểm số thi hoặc xếp hạng).

#### Không so sánh các phần tử mà đếm số lần xuất hiện.

### Radix Sort:

#### Độ phức tạp O(d(n+k)), trong đó d là số chữ số lớn nhất của các phần tử

#### Dùng cho số nguyên lớn, thích hợp khi cần tốc độ nhanh với dãy có phạm vi hẹp.

# Tìm kiếm (Searching Algorithms)

### Linear Search: O(n) - Dùng cho dữ liệu nhỏ hoặc không sắp xếp.

### Binary Search: O(log n) - Phổ biến khi tìm kiếm trên mảng/chuỗi đã sắp xếp, rất quan trọng trong tối ưu hóa tìm kiếm.

### Hashing: O(1) trung bình - Rất phổ biến trong thực tế, giúp tra cứu và chèn dữ liệu nhanh chóng (hash table, hash set).

## Nâng cao:

### Ternary Search: O(log₃ n) - Ít dùng trong thực tế, nhưng có thể áp dụng cho các bài toán tối ưu hàm đơn điệu.

### Exponential Search: Áp dụng khi cần tìm kiếm trong dãy có kích thước không rõ, nó kết hợp binary search với exponential growth.

# Đệ quy và chia để trị (Recursion and Divide and Conquer)

### Merge Sort và Quick Sort: Sử dụng chia để trị để giải quyết vấn đề sắp xếp.

### Binary Search: Áp dụng đệ quy để tìm kiếm hiệu quả trong mảng đã sắp xếp.

### Backtracking: Dùng đệ quy trong việc giải các bài toán tổ hợp (ví dụ: N-Queens, Sudoku).

### Tìm tổ hợp và hoán vị: Áp dụng trong các bài toán tìm dãy con, tổ hợp.

##### Đệ quy là công cụ mạnh mẽ, nhưng phải cẩn thận để tránh stack overflow. Các bài toán phức tạp như **N-Queens**, **Sudoku**, và **Tìm đường đi mê cung** đều có thể được giải bằng đệ quy.

# Cấu trúc dữ liệu (Data Structures)

### Array/Vector: O(1) truy cập - Cấu trúc cơ bản nhất, sử dụng rộng rãi.

### Linked List: O(n) tìm kiếm, O(1) thêm/xóa ở đầu/cuối - Phù hợp khi cần thêm/xóa phần tử thường xuyên.

### Stack: O(1) thêm/xóa - Sử dụng trong các bài toán duyệt DFS, kiểm tra dấu ngoặc, hủy bỏ hành động.

### Queue/Deque: O(1) thêm/xóa - Dùng trong các thuật toán BFS, xử lý hàng đợi.

### Priority Queue/Heap: O(log n) - Quan trọng trong các thuật toán tìm đường đi ngắn nhất (Dijkstra), xử lý sự kiện.

### Fenwick Tree (Binary Indexed Tree): O(log n) - Tương tự Segment Tree nhưng đơn giản hơn để cài đặt, ứng dụng trong các bài toán về truy vấn mảng.

## Nâng cao:

### Hash Table:

#### O(1) - Tra cứu nhanh, ứng dụng rộng rãi trong tìm kiếm và quản lý dữ liệu.

#### Tuy nhiên, khi xảy ra đụng độ, hiệu suất có thể giảm.

### Heap (Priority Queue): Là cấu trúc dữ liệu giúp tìm phần tử lớn nhất hoặc nhỏ nhất một cách nhanh chóng. Nó rất hữu ích trong thuật toán Dijkstra để tìm đường đi ngắn nhất trong đồ thị.

### Segment Tree: O(log n) - Dùng trong các bài toán truy vấn và cập nhật mảng động (tìm min/max, tổng).

### Trie: O(m), với m là độ dài từ - Sử dụng trong các bài toán về chuỗi (tìm kiếm từ điển, autocomplete).

# Lý thuyết đồ thị (Graph Theory)

##### Đồ thị rất quan trọng trong lập trình, dùng để mô hình hoá nhiều bài toán như mạng lưới, đường đi, cây bao trùm...

### ****BFS (Breadth-First Search)****: Duyệt đồ thị theo chiều rộng, sử dụng trong tìm kiếm đường ngắn nhất trong đồ thị không trọng số.

### ****DFS (Depth-First Search)****: Duyệt đồ thị theo chiều sâu, thường dùng trong bài toán tìm thành phần liên thông, chu trình trong đồ thị.

### ****Dijkstra****: Độ phức tạp O((V+E) log V) - Tìm đường đi ngắn nhất từ một đỉnh đến các đỉnh khác trong đồ thị không có trọng số âm.

### ****Floyd-Warshall****: O(V³) - Tìm đường đi ngắn nhất giữa mọi cặp đỉnh trong đồ thị.

### ****Bellman-Ford****: O(VE) - Tìm đường đi ngắn nhất trong đồ thị có cạnh trọng số âm.

### ****Kruskal và Prim****: O(E log E) và O(V²) - Tìm cây khung nhỏ nhất (MST) trong đồ thị.

# Lập trình động (Dynamic Programming - DP)

##### Lập trình động là công cụ mạnh mẽ để giải quyết các bài toán tối ưu. DP giúp lưu trữ các kết quả trung gian để tránh tính toán lại.

## Các bài toán phổ biến:

### Knapsack Problem: Tối ưu hóa việc chọn đồ sao cho tổng trọng lượng không vượt quá giới hạn, nhưng tổng giá trị là cao nhất.

### Longest Common Subsequence (LCS): Tìm dãy con chung dài nhất giữa hai chuỗi.

### Longest Increasing Subsequence (LIS): Tìm dãy con tăng dài nhất.

### Fibonacci: Tính số Fibonacci với việc lưu trữ kết quả để tránh tính toán lại.

### Matrix Chain Multiplication: Tìm cách tối ưu để tính tích của chuỗi ma trận.

### Subset Sum Problem: Tìm xem có tồn tại tập con nào có tổng bằng một giá trị cho trước.

# Xử lí chuỗi

### KMP (Knuth-Morris-Pratt): Tìm kiếm chuỗi con trong thời gian O(n), sử dụng bảng prefix function để tránh lặp lại so khớp.

### Rabin-Karp: Tìm kiếm chuỗi con bằng cách sử dụng hàm băm, độ phức tạp trung bình O(n).

### Z Algorithm: Tìm tất cả các vị trí xuất hiện của một chuỗi con trong chuỗi chính.

### Suffix Array và Suffix Tree: Dùng trong các bài toán tìm kiếm nhanh, tìm dãy con chung dài nhất giữa hai chuỗi, so sánh chuỗi con.

# Thuật toán tham lam (Greedy Algorithms)

##### Các thuật toán tham lam giải quyết bài toán bằng cách đưa ra quyết định tối ưu cục bộ tại từng bước, với hy vọng đạt được kết quả tối ưu toàn cục.

## Các bài toán phổ biến:

### Activity Selection: Chọn các hoạt động sao cho chúng không chồng lấn nhau và tổng thời gian là lớn nhất.

### Huffman Coding: Tạo mã hóa cho các ký tự theo tần suất xuất hiện, ứng dụng trong nén dữ liệu.

### Kruskal và Prim: Tìm cây khung nhỏ nhất (MST) trong đồ thị.

### Dijkstra: Tìm đường đi ngắn nhất trong đồ thị không có cạnh trọng số âm.

# Số học (Number Theory)

### Greatest Common Divisor (GCD), Least Common Multiple (LCM): Ước số chung lớn nhất và bội số chung nhỏ nhất.

### Sàng Eratosthenes:

#### Thuật toán tìm tất cả các số nguyên tố nhỏ hơn N.

#### Mô phỏng: https://www.tiktok.com/@data\_engineer\_master/video/7138033407191256346?q=s%C3%A0ng%20eratosthenes&t=1727965210484

### Modular Arithmetic: Các phép toán với số dư.

### Exponentiation by Squaring: Tính lũy thừa nhanh.

### Euclidean Algorithm: Tìm GCD bằng phương pháp Euclid.

### Chinese Remainder Theorem, Fermat's Little Theorem: Các bài toán lý thuyết số nâng cao.

# Kỹ thuật và chiến lược tối ưu (Optimization Techniques)

### Two Pointers: Kỹ thuật hai con trỏ trong việc duyệt mảng hoặc xâu.

### Sliding Window: Kỹ thuật cửa sổ trượt để tối ưu việc tìm kiếm hoặc tính toán trên dãy.

### Divide and Conquer: Chia để trị, dùng trong các thuật toán như Merge Sort, Quick Sort.

### Greedy vs. Dynamic Programming: Phân biệt giữa hai phương pháp giải quyết bài toán, hiểu khi nào nên áp dụng từng kỹ thuật.

# Thuật toán hình học (Computational Geometry)

### Convex Hull (Graham Scan, Jarvis March): Tìm bao lồi của một tập hợp điểm.

### Line Intersection: Kiểm tra giao nhau giữa hai đoạn thẳng.

### Closest Pair of Points: Tìm cặp điểm gần nhau nhất.