Trong lĩnh vực lập trình và khoa học máy tính, “thuật toán” là tập hợp các bước xử lý có thứ tự nhằm giải quyết một bài toán cụ thể. Mặc dù không có “một danh sách duy nhất” liệt kê toàn bộ các thuật toán vì chúng vô cùng phong phú và đa dạng, nhưng có thể phân loại chúng theo nhiều tiêu chí khác nhau. Dưới đây là một số nhóm lớn của các loại thuật toán mà lập trình viên thường gặp:

**1. Phân loại theo chức năng và ứng dụng**

**a. Thuật toán sắp xếp (Sorting Algorithms)**

Các thuật toán này được thiết kế để sắp xếp dãy dữ liệu theo thứ tự (tăng dần, giảm dần, theo thứ tự từ điển, …). Ví dụ:

* **Bubble Sort (Sắp xếp nổi bọt):** So sánh và hoán đổi các phần tử liền kề cho đến khi dãy được sắp xếp.
* **Insertion Sort (Sắp xếp chèn):** Chèn dần các phần tử vào vị trí thích hợp trong mảng đã sắp xếp.
* **Selection Sort (Sắp xếp chọn):** Liên tục tìm phần tử nhỏ nhất (hoặc lớn nhất) và đưa về vị trí đầu.
* **Quick Sort (Sắp xếp nhanh):** Áp dụng mô hình “chia để trị” bằng cách chọn pivot, phân chia mảng thành các mảng con rồi sắp xếp đệ quy.
* **Merge Sort (Sắp xếp trộn):** Chia mảng thành các mảng con, sắp xếp riêng và trộn lại.
* **Heap Sort, Radix Sort, Shell Sort:** Các biến thể khác có ưu điểm riêng về hiệu năng tùy thuộc vào dữ liệu.

**b. Thuật toán tìm kiếm (Searching Algorithms)**

Được sử dụng để tìm kiếm một phần tử trong tập hợp dữ liệu:

* **Linear Search (Tìm kiếm tuyến tính):** Duyệt từng phần tử cho đến khi tìm thấy mục tiêu.
* **Binary Search (Tìm kiếm nhị phân):** Áp dụng cho mảng đã được sắp xếp, chia nhỏ khoảng tìm kiếm liên tục (độ phức tạp O(log n)).
* **Jump Search, Interpolation Search:** Các biến thể cải tiến dựa trên tính chất của dữ liệu đã sắp xếp.

**c. Thuật toán đồ thị (Graph Algorithms)**

Xử lý các vấn đề liên quan đến đồ thị (nodes và edges):

* **Thuật toán duyệt đồ thị:**
  + **Breadth-First Search (BFS):** Duyệt theo chiều rộng, thường sử dụng hàng đợi.
  + **Depth-First Search (DFS):** Duyệt theo chiều sâu, thường sử dụng đệ quy hoặc ngăn xếp.
* **Tìm đường đi ngắn nhất:**
  + **Dijkstra, Bellman–Ford, Floyd–Warshall, A\* (A-star):** Dùng để tìm đường đi tối ưu trong đồ thị.
* **Tìm cây bao trùm nhỏ nhất:**
  + **Prim, Kruskal:** Xây dựng cây kết nối các đỉnh với tổng trọng số nhỏ nhất.
* **Các bài toán khác:** Tìm thành phần liên thông, tô màu đồ thị, tìm chu trình, v.v.

**d. Thuật toán lập trình động (Dynamic Programming - DP)**

Giải quyết các bài toán phức tạp bằng cách phân chia thành các bài toán con trùng lặp và lưu lại kết quả để tái sử dụng (memoization):

* Ví dụ: Dãy Fibonacci, bài toán knapsack, chuỗi con chung dài nhất (Longest Common Subsequence), khoảng cách chỉnh sửa (Edit Distance), v.v.

**e. Thuật toán tham lam (Greedy Algorithms)**

Luôn chọn lựa giải pháp tốt nhất ở mỗi bước (tối ưu cục bộ) với hy vọng đạt được giải pháp tối ưu toàn cục:

* Ví dụ: Thuật toán Huffman cho nén dữ liệu, bài toán đổi tiền, Lập lịch công việc, Dijkstra (có tính chất tham lam), v.v.

**f. Thuật toán quay lui (Backtracking)**

Giải bài toán bằng cách thử các khả năng khác nhau và “quay lại” khi gặp trường hợp không khả thi:

* Ví dụ: Bài toán N-Queens, giải Sudoku, tạo hoán vị, tổ hợp, v.v.

**g. Mô hình “Chia để trị” (Divide and Conquer)**

Không phải là thuật toán cụ thể mà là một mô hình giải bài toán bằng cách:

* **Divide (Chia nhỏ):** Chia bài toán ban đầu thành các bài toán con độc lập.
* **Conquer (Giải quyết):** Giải từng bài toán con.
* **Combine (Kết hợp):** Hợp nhất các kết quả của bài toán con để ra lời giải tổng thể.
* Ví dụ: Binary Search, Merge Sort, Quick Sort, Strassen’s Matrix Multiplication.

**h. Thuật toán hashing (Hashing Algorithms)**

Sử dụng hàm băm để chuyển đổi dữ liệu thành chỉ số (index) giúp tra cứu và lưu trữ nhanh:

* Ví dụ: Hash table, các kỹ thuật giải quyết va chạm như chaining và open addressing.

**i. Thuật toán xử lý chuỗi (String Algorithms)**

Dành cho các bài toán liên quan đến xử lý, tìm kiếm và phân tích chuỗi ký tự:

* **String Matching:** Naive search, Knuth-Morris-Pratt (KMP), Boyer-Moore, Rabin-Karp.
* Các thuật toán xử lý chuỗi khác như tạo suffix tree, suffix array.

**j. Thuật toán hình học tính toán (Computational Geometry)**

Giải quyết các bài toán về hình học và không gian:

* Ví dụ: Tìm bao lồi (Convex Hull – Graham Scan, Jarvis March), kiểm tra giao nhau của đường thẳng, tìm khoảng cách giữa các điểm, v.v.

**k. Thuật toán lý thuyết số (Number Theory Algorithms)**

Áp dụng vào các bài toán liên quan đến số học:

* Ví dụ: Thuật toán Euclid (GCD), Sàng Eratosthenes để tìm số nguyên tố, nhân số lớn (Karatsuba, Toom–Cook), kiểm tra số nguyên tố (Miller–Rabin, AKS), phân tích số nguyên thành thừa số nguyên tố.

**l. Các thuật toán ngẫu nhiên và tối ưu hóa**

* **Thuật toán ngẫu nhiên (Randomized Algorithms):** Sử dụng các yếu tố ngẫu nhiên để đưa ra lời giải (Monte Carlo, Las Vegas).
* **Thuật toán tối ưu hóa:** Các phương pháp như Gradient Descent, Simulated Annealing, Genetic Algorithms, Differential Evolution… được sử dụng trong các bài toán tối ưu và học máy.

**2. Phân loại theo cách tiếp cận giải bài toán (Paradigms)**

Ngoài việc phân loại theo chức năng, các thuật toán còn được chia theo cách tiếp cận giải quyết bài toán:

* **Đệ quy (Recursion):** Nhiều thuật toán (như DP, DFS, Backtracking) sử dụng đệ quy để lặp lại quá trình giải bài toán.
* **Tham lam (Greedy):** Luôn chọn lựa giải pháp tối ưu cục bộ tại mỗi bước.
* **Chia để trị (Divide and Conquer):** Chia bài toán thành các phần nhỏ hơn, giải quyết và kết hợp lại.
* **Quay lui (Backtracking):** Thử các khả năng và loại bỏ dần các trường hợp không khả thi.

**Tổng kết**

Không có một danh sách “tất cả” các thuật toán cố định vì lĩnh vực này luôn phát triển với rất nhiều biến thể và ứng dụng khác nhau. Tuy nhiên, việc nắm vững các nhóm thuật toán chính – từ sắp xếp, tìm kiếm, xử lý chuỗi, đồ thị, lập trình động, tham lam cho đến các mô hình giải bài toán như chia để trị và quay lui – sẽ giúp bạn xây dựng nền tảng vững chắc để giải quyết hầu hết các bài toán trong lập trình.

Mỗi nhóm thuật toán có những ưu nhược điểm riêng về độ phức tạp về thời gian (Big-O), yêu cầu bộ nhớ và phù hợp với từng kiểu dữ liệu hoặc bài toán cụ thể. Việc lựa chọn đúng thuật toán không chỉ giúp chương trình chạy nhanh, hiệu quả mà còn góp phần cải thiện khả năng bảo trì và mở rộng của phần mềm.

Hy vọng bài tổng hợp trên đã giúp bạn có cái nhìn toàn diện về “các loại thuật toán trong lập trình” và định hướng học tập, áp dụng một cách hiệu quả trong công việc của mình.