Quá trình phân tích và thiết kế thuật toán

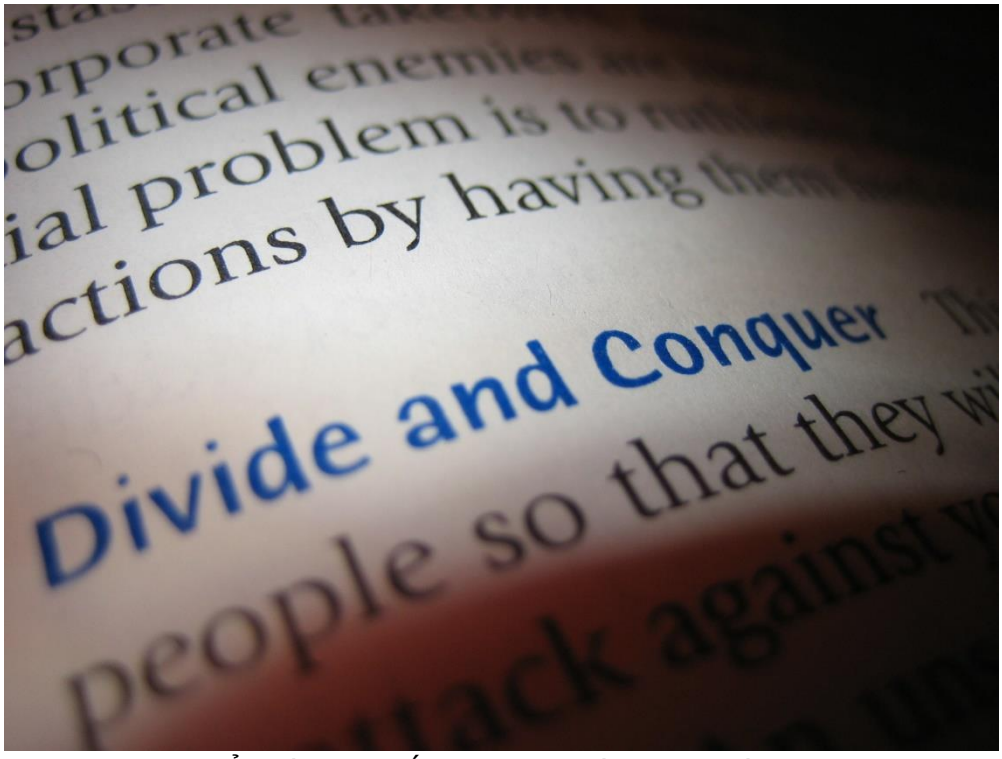
Sau đây là các bước xây dựng một thuật toán hoàn chỉnh.

#1 Devising algorithm - phân tích và phác thảo

Tạo ra một thuật toán trong thực tế là cả một nghệ thuật, trong đó, lập trình viên chính là nghệ sĩ.

Để hoàn thành được bước phân tích và phác thảo, ta cần đến khá nhiều kiến thức căn bản của môn **Cấu trúc dữ liệu và giải thuật**, cụ thể đó là các **chiến thuật thiết kế thuật toán**, gồm có 5 đại diện tiêu biểu:

1. **Chia để trị - divide and conquer**
2. **Giải thuật tham ăn - Greedy Method**
3. **Lập trình quy hoạch động - Dynamic Programming**
4. **Back Tracking**
5. **Branch and Bound**



Chia để trị là một chiến thuật được áp dụng khá rộng rãi

#2 Algorithm Validation - Kiểm tra thuật toán

Một khi thuật toán đã được thiết kế xong, ta cần kiểm tra tính đúng đắn của nó bằng cách nhét thuật toán vào máy tính và đưa cho nó một đống input - dữ liệu đầu vào. Tất nhiên các dữ liệu này phải tồn tại dưới định dạng phù hợp. Mục tiêu của việc Validate thuật toán là để đảm bảo nó sẽ hoạt động trơn tru trên mọi ngôn ngữ lập trình.

#3 Algorithm Analysis - Đánh giá thuật toán

Khi thuật toán chạy, nó cần tiêu tốn 2 thứ: thời gian xử lý của CPU để thực hiện các phép toán và bộ nhớ để chứa dữ liệu + chương trình thực thi. Do đó, ở giai đoạn này, ta đánh giá hiệu năng thuật toán theo 2 yếu tố: thời gian thực thi, bộ nhớ sử dụng.

*Để hiểu sâu hơn mối quan hệ giữa thuật toán và tài nguyên máy tính, hãy tham khảo cuốn **Programming From Ground Up** của Jonathan Bartlett.*

#4 Program Testing- Test chương trình

Việc test chương trình được chia thành 2 giai đoạn: **debugging** và **profiling**.

Debugging là quá trình thực thi chương trình dựa trên một tập dữ liệu mẫu để xác định các lỗi xảy ra (loại lỗi, vị trí lỗi,...) và khắc phục chúng. Tuy nhiên Debugging hầu như không bao giờ phát hiện được 100% lỗi. Đó là lý do các bản cập nhật và bản vá lỗi (patch) ra đời.

Profiling cũng là quá tình thực thi chương trình trên một tập dữ liệu mẫu nhưng lần này, người ta sẽ đo thời gian cũng như dung lượng bộ nhớ. Khoan đã, vậy thì khác gì bước Analysis ở trên?

#5 Algorithm Analysis và Performance Analysis?

Như chúng tôi đề cập ở trên, một khi thuật toán được chạy, nó sẽ sử dụng CPU để thực hiện các phép toán cũng như bộ nhớ trong để chưa dữ liệu + chương trình. Vì lí do đó, thời gian và không gian bộ nhớ được chọn để đánh giá hiệu năng của thuật toán.

Ta có thể đánh giá hiệu năng của thuật toán bằng phương pháp thực nhiệm thông qua việc cài đặt thuật toán rồi chọn các tập dữ liệu mẫu để chạy thử. Sau đó thống kê các thông số nhận được để có đánh giá về thuật toán.

Tuy nhiên phương pháp thực nghiệm này gặp một số hạn chế khi áp dụng trên thực tế:

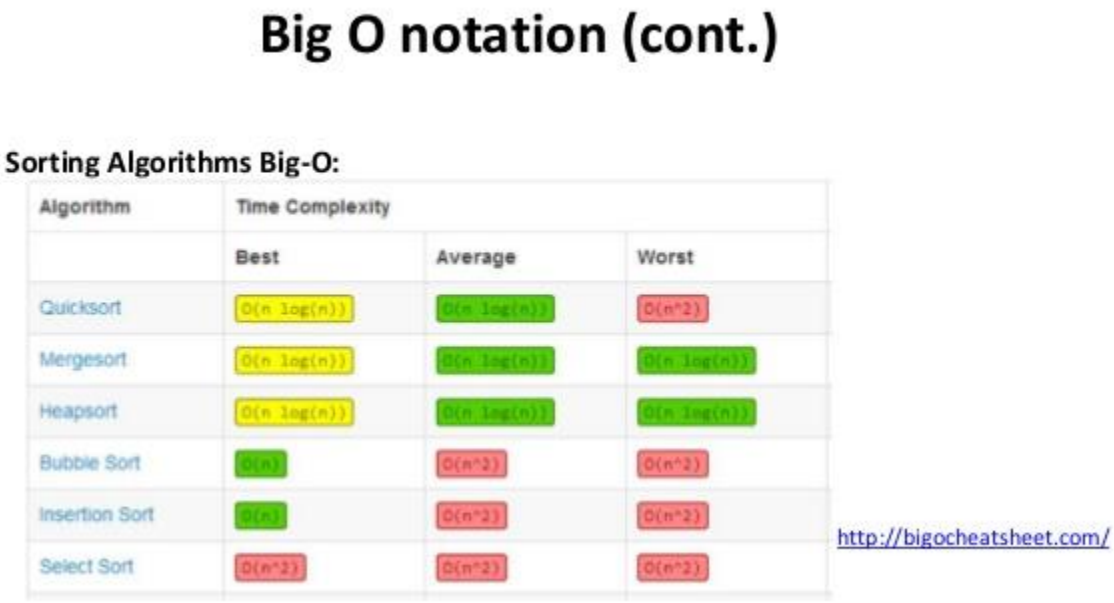
- **Phần cứng** khác nhau
- Thời gian thực thi của **mã máy**
- **Compiler** - trình biên dịch
- Giới hạn của **ngôn ngữ lập trình** sử dụng để cài đặt
- Khó chọn lựa các **bộ dữ liệu mẫu** một cách tối ưu.

Như vậy, việc so sánh các thuật toán bằng phương pháp thực nghiệm hiếm khi đạt được độ chính xác mong muốn.

Vì những lý do trên, người ta đã tìm kiếm những phương pháp đánh giá thuật toán ít phụ thuộc môi trường hơn. Một trong số đó là phương pháp đánh giá thời gian thực hiện thuật toán theo hướng xấp xỉ tiệm cận qua khái niệm O() - gọi là Big O. (Hay còn gọi là độ phức tạp của thuật toán)

Thông thường các vấn đề mà chúng ta giải quyết có một kích thước tự nhiên (số lượng dữ liệu được xử lý) mà chúng ta sẽ gọi là N. Như vậy, để đánh giá thuật toán theo yếu tố thời gian, chúng ta sẽ biểu diễn thời gian cần thiết thông qua một hàm số của N. Chúng ta quan tầm đến 2 trường hợp: xấu nhất và trung bình.

Việc xác định thời gian trong trường hợp trung bình thường được quan tâm nhiều nhất vì nó đại diện cho đa số trường hợp sử dụng thuật toán. Tuy nhiên, với một số bài toán đặc thù, việc xác định thời gian trong trường hợp xấu nhất là rất quan trọng. Ví du: các bài toán trong y tế, hàng không, quân sự, ...



Bảng tổng hợp về độ phức tạp của các thuật toán sắp xếp

Tham khảo bài viết gốc tại CodingSec