### 1.6. TỐI ƯU CHƯƠNG TRÌNH

Một chương trình đã chạy đúng không có nghĩa là việc lập trình đã xong, ta phải sửa đổi lại một vài chi tiết để chương trình có thể chạy nhanh hơn, hiệu quả hơn. Thông thường, trước khi kiểm thử thì ta nên đặt mục tiêu viết chương trình sao cho đơn giản, **miễn sao chạy ra kết quả đúng** là được, sau đó khi tối ưu chương trình, ta xem lại những chỗ nào viết chưa tốt thì tối ưu lại mã lệnh để chương trình ngắn hơn, chạy nhanh hơn. Không nên viết tới đâu tối ưu mã đến đó, bởi chương trình có mã lệnh tối ưu thường phức tạp và khó kiểm soát.

Việc tối ưu chương trình nên dựa trên các tiêu chuẩn sau:

**1.6.1. Tính tin cậy**

Chương trình phải chạy đúng như dự định, mô tả đúng một giải thuật đúng. Thông thường khi viết chương trình, ta luôn có thói quen kiểm tra tính đúng đắn của các bước mỗi khi có thể.

**1.6.2. Tính uyển chuyển**

Chương trình phải dễ sửa đổi. Bởi ít có chương trình nào viết ra đã hoàn hảo ngay được mà vẫn cần phải sửa đổi lại. Chương trình viết dễ sửa đổi sẽ làm giảm bớt công sức của lập trình viên khi phát triển chương trình.

**1.6.3. Tính trong sáng**

Chương trình viết ra phải dễ đọc dễ hiểu, để sau một thời gian dài, khi đọc lại còn hiểu mình làm cái gì?. Để nếu có điều kiện thì còn có thể sửa sai (nếu phát hiện lỗi mới), cải tiến hay biến đổi để được chương trình giải quyết bài toán khác. Tính trong sáng của chương trình phụ thuộc rất nhiều vào công cụ lập trình và phong cách lập trình.

**1.6.4. Tính hữu hiệu**

Chương trình phải chạy nhanh và ít tốn bộ nhớ, tức là tiết kiệm được cả về không gian và thời gian. Để có một chương trình hữu hiệu, cần phải có giải thuật tốt và những tiểu xảo khi lập trình. Tuy nhiên, việc áp dụng quá nhiều tiểu xảo có thể khiến chương trình trở nên rối rắm, khó hiểu khi sửa đổi. Tiêu chuẩn hữu hiệu nên dừng lại ở mức chấp nhận được, không quan trọng bằng ba tiêu chuẩn trên. Bởi phần cứng phát triển rất nhanh, yêu cầu hữu hiệu không cần phải đặt ra quá nặng.

Từ những phân tích ở trên, chúng ta nhận thấy rằng việc làm ra một chương trình đòi hỏi rất nhiều công đoạn và tiêu tốn khá nhiều công sức. Chỉ một công đoạn không hợp lý sẽ làm tăng chi phí viết chương trình. Nghĩ ra cách giải quyết vấn đề đã khó, biến ý tưởng đó thành hiện thực cũng không dễ chút nào.

Những cấu trúc dữ liệu và giải thuật đề cập tới trong chuyên đề này là những kiến thức rất phổ thông, một người học lập trình không sớm thì muộn cũng phải biết tới. Chỉ hy vọng rằng khi học xong chuyên đề này, qua những cấu trúc dữ liệu và giải thuật hết sức mẫu mực, chúng ta rút ra được bài học kinh nghiệm: **Đừng bao giờ viết chương trình khi mà chưa suy xét kỹ về giải thuật và những dữ liệu cần thao tác**, bởi như vậy ta dễ mắc phải hai sai lầm trầm trọng: hoặc là sai về giải thuật, hoặc là giải thuật không thể triển khai nổi trên một cấu trúc dữ liệu không phù hợp. Chỉ cần mắc một trong hai lỗi đó thôi thì nguy cơ sụp đổ toàn bộ chương trình là hoàn toàn có thể, càng cố chữa càng bị rối, khả năng hầu như chắc chắn là phải làm lại từ đầu.