# Môn PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

## Chủ đề 4
## PHƯƠNG PHÁP THIẾT KẾ THUẬT TOÁN COMPLETE SEARCH BRUTE FORCE

**Mục tiêu:**

Sinh viên vận dụng được phương pháp thiết kế để giải bài toán cụ thể.

**Nội dung:**

- Đặc điểm bài toán,
- Dạng thuật toán phổ quát (template/framework),
- Bài toán như thế nào thì cần/không cần dùng Brute Force,
- Ưu điểm, nhược điểm,
- Bài tập minh họa (thích hợp và không thích hợp),
- Bài tập về nhà (Wecode),
- Quiz để đánh giá mức độ hiểu bài.

### NỘI DUNG

### 1. ĐẶC ĐIỂM BÀI TOÁN.

a. *Brute Force là gì?*

Brute Force là cách tiếp cận đơn giản để giải quyết bài toán, thường trực tiếp dựa trên đề bài và định nghĩa của các khái niệm liên quan. Brute Force vét cạn và chạy tất cả các trường hợp có thể để giải quyết bài toán, chạy qua cả trường hợp đúng lẫn trường hợp sai.

> Ví dụ, trong phép tính lũy thừa, cách tiếp cận Brute Force đưa ra cách tính là : Thực hiện phép nhân với số lần bằng số mũ.

b. *Độ phức tạp.*

Độ phức tạp của Brute Force phụ thuộc vào giải thuật áp dụng nó.

> Nếu áp dụng Brute Force, tìm string con trong string sẽ có độ phức tạp thời gian là O(m * n) ; tìm phần tử lớn nhất trong mảng sẽ có độ phức tạp thời gian là O(n).
> Giải thuật MonkeySort/BogoSort có độ phức tạp thời gian là O(n * n!).

c. *Vì sao Brute Force lại quan trọng ?*

Cách tiếp cận Brute Force dù rất đơn giản và có độ phức tạp lớn, nhưng nhìn chung vẫn không nên bị xem nhẹ vì

(1) Brute Force được sử dụng cho mục đích nghiên cứu và giáo dục (đánh giá hiệu năng của các giải thuật khác).
(2) Đối với các bài toán như sắp xếp, tìm kiếm và nhân ma trận, Brute Force giúp đưa ra thuật toán hợp lý có một số giá trị thực tiễn, mà không bị giới hạn bởi kích thước đối tượng.
(3) Không cần phải thiết kế thuật toán có hiệu năng cao hơn, nếu bài toán không quá lớn; Brute

Force vẫn có thể giải quyết bài toán đó trong khoảng thời gian phù hợp.

(4) Dù có hiệu năng không cao, Brute Force vẫn có hiệu quả với những bài toán nhỏ.

(5) Brute Force được áp dụng rộng rãi để giải nhiều bài toán.

## 2. DẠNG THUẬT TOÁN PHỔ QUÁT (FRAMEWORK)

Phương pháp Brute Force có thể được biểu diễn bằng pseudo-code như sau :

```
c ← first(P)
while c not Λ do
    if valid(P, c)
    then output(P, c)
    end if
    c ← next(P, c)
end while
```

Trong đó,

P là bài toán,

c là từng *ứng viên* khả thi để giải quyết P,

`valid(P, c)` kiểm tra xem *ứng viên* c có phải lời giải cho P hay không,

`output(P, c)` dùng lời giải c của P tùy theo ứng dụng (chẳng hạn, in c ra màn hình rồi dừng chương trình),

`first(P)` trả về giá trị c đầu tiên được xét đến của bài toán P

và `next(P, c)` trả về giá trị c tiếp theo được xét đến của bài toán P, ngay sau giá trị c được cung cấp trong đối số.

Trong thực tế, `next` cần thông báo khi không còn ứng viên nào sau giá trị c hiện tại. Người ta thường dùng "ứng viên rỗng" cho mục đích này (dưới đây ký hiệu là Λ, lambda). Không ứng viên khả thi nào có thể mang giá trị của Λ. Tương tự, `first` cũng cần trả về Λ nếu ngay từ đầu, không có ứng viên nào cho bài toán P.

*Ví dụ :* Xét bài toán "Tìm vị trí xuất hiện đầu tiên của số X trong mảng số nguyên A và in kết quả ra màn hình."

- P là "Tìm vị trí xuất hiện đầu tiên của số X trong mảng số nguyên A và in kết quả ra màn hình."
- c là từng index của các giá trị trong mảng.
- `valid(P, c)` là kết quả của phép so sánh `A[c] == X`.
- `output(P, c)` là chuỗi hành động in c ra màn hình và kết thúc chương trình.
- Λ là n (n là kích thước mảng A. Đây là giá trị index mà không phần tử nào của A có được nếu `A[0..n - 1]`).
- `first(P)` trả về 0, là giá trị index đầu tiên cần xét đến.
- `next(P, c)` trả về `c + 1`, là giá trị index tiếp theo cần xét đến. Nếu c đã đạt đến cuối mảng (c = n - 1) thì `next(P, c)` sẽ trả về n, và không thỏa mãn điều kiện c khác Λ nữa.

*Chú ý :* Đối với mỗi ứng viên là lời giải của P, thuật toán sẽ gọi `output`. Các thuật toán có thể cài đặt sao cho output làm dừng chương trình sau khi tìm ra lời giải đầu tiên, hoặc sau một số lời giải nhất định, hoặc sau khi kiểm tra một số ứng viên nhất định, hoặc sau một khoảng thời gian nhất định dùng CPU.

**3. KHI NÀO CẦN DÙNG BRUTE FORCE** (Bao gồm cả "Khi nào không cần dùng Brute Force")

Bài toán có thể được áp dụng Brute Force nếu có các đặc điểm sau :

- Có kích thước nhỏ (mảng có không quá 100 phần tử, vân vân).

  Khi sắp xếp mảng có không quá 100 phần tử, ta không cần thiết phải thiết kế thuật toán có hiệu năng cao hơn. Brute Force lúc này có thể giải quyết bài toán trong khoảng thời gian có thể chấp nhận được, nhất là với khả năng của máy tính hiện nay.

- Khi cần tìm tất cả các giải pháp khả thi để giải quyết bài toán.

  Khi cần tìm giá trị lớn nhất trong một mảng không được sắp xếp, ta cần duyệt qua tất cả phần tử để so sánh và tìm ra giá trị lớn nhất.

Tuy nhiên không nên dùng Brute Force cho các bài toán :

- Có kích thước rất lớn (mảng có hơn 1000 phần tử, vân vân).

  Khi ta cân nhắc độ phức tạp thời gian, có thể thấy các bài toán lớn sẽ cần một khoảng thời gian khủng khiếp để giải quyết nếu sử dụng Brute Force.

- Khi có thể tìm ra lời giải mà không phải đi qua tất cả các giải pháp khả thi.

  Khi tìm một phần tử trong mảng đã sắp xếp theo giá trị cho trước, ta có thể áp dụng giải thuật tìm kiếm nhị phân trong thời gian O(log n) mà không phải duyệt qua tất cả các phần tử, từ đầu đến cuối.

**4. ƯU ĐIỂM VÀ NHƯỢC ĐIỂM**

Phương pháp Brute Force có nhiều ưu điểm, có thể kể đến như :

- Brute Force được áp dụng mà không bị giới hạn trong một chủ đề nào.
- Đây là phương pháp lý tưởng cho các bài toán nhỏ đơn giản.
- Brute Force được so sánh với các thuật toán hiệu quả hơn, giúp nhấn mạnh tính ưu việt của những thuật toán đó.
- Brute Force giúp tìm ra tất cả các giải pháp khả thi để giải bài toán.

Bên cạnh đó, Brute Force cũng có một số nhược điểm làm ta phải cẩn trọng khi dùng :

- Brute Force nhìn chung không hiệu quả và chậm.
- Các thuật toán áp dụng Brute Force thường phụ thuộc vào hiệu suất máy tính hơn là thiết kế thuật toán.
- Cách tiếp cận Brute Force không mấy sáng tạo so với nhiều thuật toán khác.

**5. BÀI TOÁN MINH HỌA** (Bao gồm một bài thích hợp và một bài không thích hợp. Mỗi bài toán bao gồm độ phức tạp.)

*a. Bài toán thích hợp : Kiểm tra tính nguyên tố.*

Kiểm tra xem một số có phải là số nguyên tố hay không bằng phép chia thử với độ chính xác tuyệt đối.

```
function is_prime(x):
for f = 2 to x – 1 do
   if x mod f = 0
   then return 'not prime'
end for
return 'prime'
```

Độ phức tạp: Theta(n) nếu n là số nguyên tố, Theta(ước số nhỏ nhất của n) nếu n là số nguyên tố. Nếu ta xét số bit của input, độ phức tạp cho trường hợp tệ nhất là Theta(2^n) cho số nguyên n bit.

Để cải thiện thuật toán, ta có thể giảm số ước số để chia thử (nội dung đó nằm ngoài phạm vi của chủ để này).

*b. Bài toán không thích hợp : Duyệt đồ thị.*

Cho một đồ thị vô hướng liên thông có trọng số. Bắt đầu từ đỉnh 1, hãy chỉ ra chu trình Hamilton có đường đi ngắn nhất.

## 6. BÀI TẬP VỀ NHÀ

Một slide thông báo bài tập về nhà trên Wecode, bao gồm đường liên kết và thời gian làm bài (từ lúc nào đến lúc nào). Các nội dung này sẽ được đăng lại trên trang môn học.

**7. QUIZ** (Đánh giá mức độ hiểu bài)

Có 7 câu hỏi trắc nghiệm nhanh.