TOÁN RỜI RẠC VÀ THUẬT TOÁN

(DISCRETE MATHEMATIC AND ALGORITHMS)

Bài 2

Phân tích thuật toán

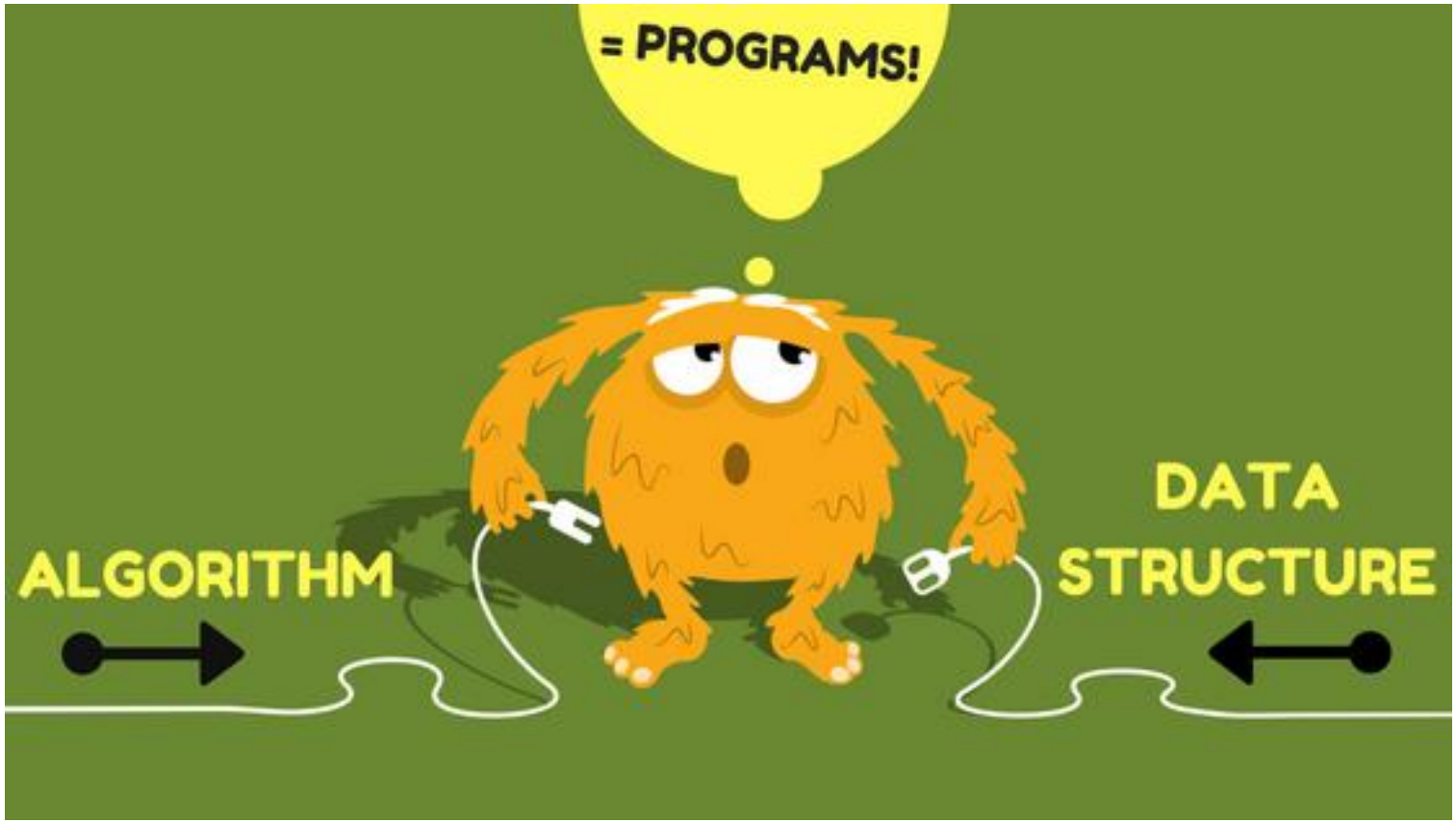
(*Algorithm Analysis*)

Nguyễn Thị Hồng Minh

minhnth@hus.edu.vn

Nội dung

- **Nhắc lại**
- **Độ phức tạp thuật toán**
- **Chứng minh tính đúng của thuật toán**
- **P & NP**

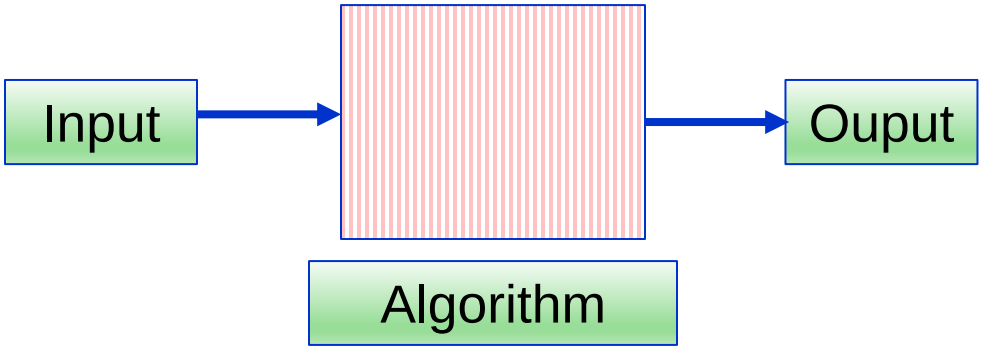


"Thuật toán + Cấu trúc dữ liệu = Chương trình"
(Algorithms + Data Structures = Programs)

(Niklaus Wirth)

Thuật toán

- **Thuật toán** – Các bước hữu hạn để giải bài toán



- **Diễn đạt thuật toán**
  - Pseudo-code: Sử dụng ngôn ngữ tự nhiên tựa ngôn ngữ lập trình
  - Tính logic: hiểu rõ tinh thần, các bước thuật toán
  - Tính có thể cài đặt: Khả năng có thể lập trình

Phân tích thuật toán

- **Phân tích để đánh giá:**
  - Là bước thực hiện sau khi thiết kế thuật toán nhằm trả lời:
    - Thuật toán có **đúng** không?
    - Thuật toán có **hiệu quả** không?
- **Tính hiệu quả:**
  - Thời gian (số lượng các bước tính toán).

    ⇨ Độ phức tạp thời gian
  - Không gian (tài nguyên sẽ được sử dụng khi thuật toán thi hành).

    ⇨ Độ phức tạp không gian

Chứng minh tính đúng

- **Các chiến lược chứng minh tính đúng (correctness)**
  - ***Kiểm thử (testing):*** Chạy thử thuật toán với các dữ liệu vào cụ thể

    Ưu điểm: Dễ thực hiện

    Nhược điểm: Có thể không phát hiện hết các lỗi (tiềm ẩn)
  - ***Chứng minh tính đúng (correctness proof):*** chứng minh bằng toán học

    Ưu điểm: Tổng quát

    Nhược điểm: khó hơn, và có thể vẫn có lỗi
  - ***Kết hợp kiểm thử và chứng minh tính đúng*** => hiệu quả hơn
- **Các phương pháp chứng minh tính đúng**
  - ***Đối với thuật toán đệ quy:*** Dùng quy nạp
  - ***Đối với thuật toán không đệ quy:*** tính đúng nằm ở các vòng lặp, sử dụng bất biến vòng lặp (loop invariant)

Chứng minh tính đúng

- **Chứng minh tính đúng đối với thuật toán đệ quy**
  - Phương pháp quy nạp:
    - Chứng minh tính đúng của thuật toán theo kích thước dữ liệu vào
    - Cơ sở của quy nạp: trường hợp suy biến của đệ quy
    - Giả thiết quy nạp: thuật toán đúng với dữ liệu kích thước n
    - Tổng quát: với dữ liệu kích thước n+1 thuật toán cho ra đúng output

Chứng minh tính đúng

- **Chứng minh tính đúng đối với thuật toán đệ quy**
  - ***Ví dụ:*** Thuật toán tìm max của một dãy số $A_1, A_2, \ldots, A_n$ có n phần tử

```
Maximum(n)≡    //Tìm max của dãy số có n phần tử
    if (n==1) return (A1)
    else return(max(Maximum(n-1),An);
End.
```

Chứng minh thuật toán trả lại đúng giá trị lớn nhất trong các phần tử $A_1, \ldots, A_n$

Cơ sở: $n=1 \Rightarrow A_1$ là phần tử duy nhất và lớn nhất

Giả thiết quy nạp: `Maximum`(n) trả lại giá trị lớn nhất $A_1, \ldots, A_n$

Tổng quát: `Maximum`(n+1) trả lại max(`Maximum`(n),A_{n+1})=max($A_1, \ldots, A_n, A_{n+1}$)

Chứng minh tính đúng

- **Chứng minh tính đúng đối với thuật toán không đệ quy**
  - Phương pháp bất biến vòng lặp (*loop invariant*)
    - Bất biến vòng lặp là *biểu thức logic (của các biến được sử dụng vòng lặp)* có giá trị không đổi trong quá trình lặp
    - Sử dụng bất biến vòng lặp để chỉ ra thuật toán lặp dừng và cho output
    - Chứng minh đối với thuật toán có 1 vòng lặp, nếu có vòng lặp lồng nhau thì phải bắt đầu từ các vòng lặp bên trong.
  - Các đặc trưng của bất biến vòng lặp: Khởi tạo, Duy trì, Kết thúc
    - Khởi tạo: bất biến của vòng lặp phải đúng trước lần lặp đầu tiên.
    - Duy trì: Nếu nó đúng trước một vòng lặp, nó vẫn còn đúng trước vòng lặp tiếp theo.
    - Kết thúc: Khi vòng lặp kết thúc, bất biến này cho chúng ta một tính chất hữu ích giúp chứng minh được thuật toán là đúng đắn.

Chứng minh tính đúng

- **Chứng minh tính đúng đối với thuật toán không đệ quy**
  - ***Ví dụ:*** Thuật toán tìm max của một dãy số $A_1, A_2,\ldots, A_n$ có n phần tử

```
Maximum(n)≡    //Tìm max của dãy số có n phần tử
    m=A1;
    for(i=2;i<=n;i++)
        if (m < Ai) m=Ai;
    return (m)
End.
```

Bất biến vòng lặp: $m_j = \max(A_1,\ldots,A_j)$

Khởi tạo: $m_1 = A_1 = \max(A_1)$ - đúng

Duy trì: $m_j=\max(A_1,\ldots A_j)$ thì $m_{j+1} = \max(m_j,A_{j+1}) = \max(A_1,\ldots A_{j+1})$

Kết thúc: $i=n+1$ sau t lần lặp ($t = n+1-2+1=n$)

$m_t = \max(A_1,\ldots A_t) = \max(A_1,\ldots A_n)$

Độ phức tạp thời gian

- **Thời gian thực hiện thuật toán (running time)**
  - Là số các thao tác cơ bản được thực hiện: phép tính số học, logic cơ bản hoặc một câu lệnh "đơn"...
  - Phụ thuộc vào kích thước dữ liệu vào
    - Trường hợp tốt nhất
    - Trường hợp xấu nhất
    - Trường hợp trung bình
- **Đánh giá thời gian thực hiện thuật toán**
  - Hai phương pháp chính:
    - *Phương pháp thực nghiệm*
    - *Phương pháp lí thuyết*

Độ phức tạp thời gian

- **Đánh giá thuật toán**
  - ***Phương pháp thực nghiệm***

    Lập trình, chạy thử nhiệm, ghi nhận các số đo về độ phức tạp.

    Ưu điểm: Có thể đánh giá được tất cả các thuật toán khả trình.

    Nhược điểm: Bị hạn chế và phụ thuộc vào máy tính thực hiện.
  - ***Phương pháp lí thuyết***

    Xác định quan hệ về thời gian với kích thước dữ liệu bằng hàm toán học.

    Ưu điểm: không phụ thuộc ngôn ngữ lập trình, loại máy tính; đánh giá được với dữ liệu có kích thước lớn.

    Nhược điểm: Khó vì biểu diễn các ràng buộc về mặt toán học

Biểu diễn tiệm cận

- **Biểu diễn tiệm cận (**Asymptotic Performance)
  - Thuật toán sẽ thế nào khi kích thước của bài toán trở nên rất lớn?
    - Thời gian thực hiện
    - Bộ nhớ yêu cầu, các tài nguyên khác (băng thông, nguồn, cổng logic…)
  - Biểu diễn tiệm cận
    - Sử dụng các kí pháp quy ước để xác định mối quan hệ giữa các hàm đánh giá độ phức tạp (lớn hơn, nhỏ hơn, xấp xỉ)

      Ví dụ: $f(n) = O(g(n))$
    - Hệ thống hóa một số hàm đánh giá độ phức tạp cơ bản

      Ví dụ: $f_1(n) = O(n)$

      $f_2(n) = O(n \log n)$

      $f_3(n) = O(2^n)$

Biểu diễn tiệm cận

- **Kí pháp biểu diễn tiệm cận** (Asymptotic Notation)
  - ***Cận trên: Ký pháp O (đọc là O lớn)***
    - **Định nghĩa:**

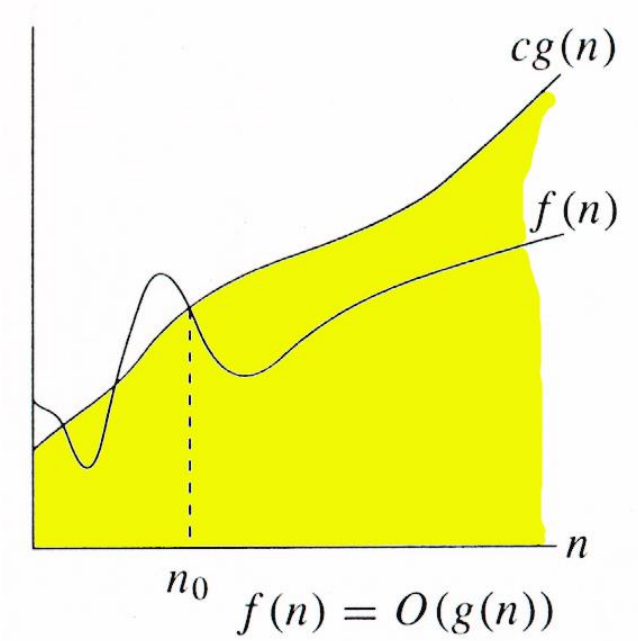
f(n) = O(g(n)) nếu tồn tại các hằng số dương c và n_0 sao cho $f(n) \le c \cdot g(n)$ với mọi $n \ge n_0$

Hình thức:

$O(g(n)) = \{ f(n): \exists$ các số $c, n_0 > 0$ sao cho $f(n) \le c \cdot g(n) \ \forall \ n \ge n_0\}$

- **Ý nghĩa:**

Tập các hàm có tốc độ tăng trưởng luôn nhỏ hơn hàm g(n)



f(n) < g(n)

Biểu diễn tiệm cận

- **Kí pháp biểu diễn tiệm cận (**Asymptotic Notation)
  - ***Cận trên: Ký pháp O (đọc là O lớn)***
    - **Ví dụ:**

- Độ phức tạp thuật toán Sắp xếp chèn $f(n) = an^2+bn+c = O(n^2)$

Chứng minh :

$$an^2 + bn + c \leq (a + b + c)n^2 + (a + b + c)n + (a + b + c)$$
$$\leq 3(a + b + c)n^2 \quad \text{với} \quad n \geq 1$$

Chọn $c' = 3(a + b + c)$ và $n_0 = 1$ => đpcm

- Độ phức tạp thuật toán Sắp xếp chèn $f(n) = an^2+bn+c = O(n^3)$???

Kí pháp cận trên đánh giá trường hợp xấu nhất, quan tâm tới hàm nhỏ nhất!

Biểu diễn tiệm cận

- **Kí pháp biểu diễn tiệm cận (**Asymptotic Notation)
  - ***Cận dưới: Ký pháp Ω (đọc là Omega)***
    - **Định nghĩa:**

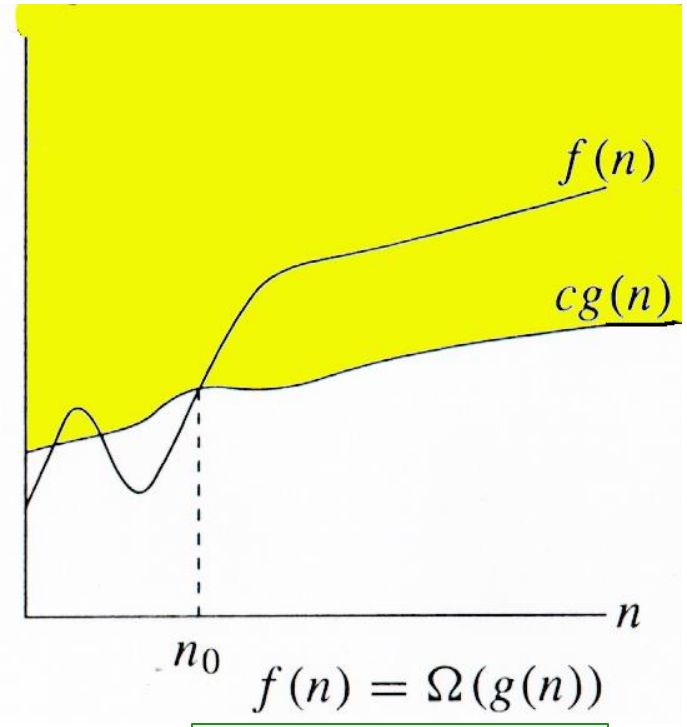
f(n) = Ω (g(n)) nếu tồn tại các hằng số dương c và n_0 sao cho f(n) ≥ c · g(n) với mọi n ≥ n_0

Hình thức:

O(g(n)) = { f(n): ∃ các số c,n_0 >0 sao cho f(n) ≥ c · g(n) ∀ n ≥ n_0}

    - **Ý nghĩa:**

Tập các hàm có tốc độ tăng trưởng luôn lớn hơn hàm g(n)



f(n) > g(n)

Biểu diễn tiệm cận

- **Kí pháp biểu diễn tiệm cận (**Asymptotic Notation)
  - ***Cận dưới: Ký pháp* Ω**
    - **Ví dụ:**

- Độ phức tạp thuật toán Sắp xếp chèn $f(n) = an^2+bn+c = \Omega(n)$

Chứng minh :

$an^2 + bn + c \geq bn + c \geq bn$ với $n \geq 1$

Chọn $c' = b$ và $n_0 = 1$ => đpcm

Kí pháp cận dưới đánh giá trường hợp tốt nhất, quan tâm tới hàm lớn nhất!

Biểu diễn tiệm cận

- **Kí pháp biểu diễn tiệm cận** (Asymptotic Notation)
  - ***Cận chặt: Ký pháp Θ (đọc là Theta)***
    - **Định nghĩa:**

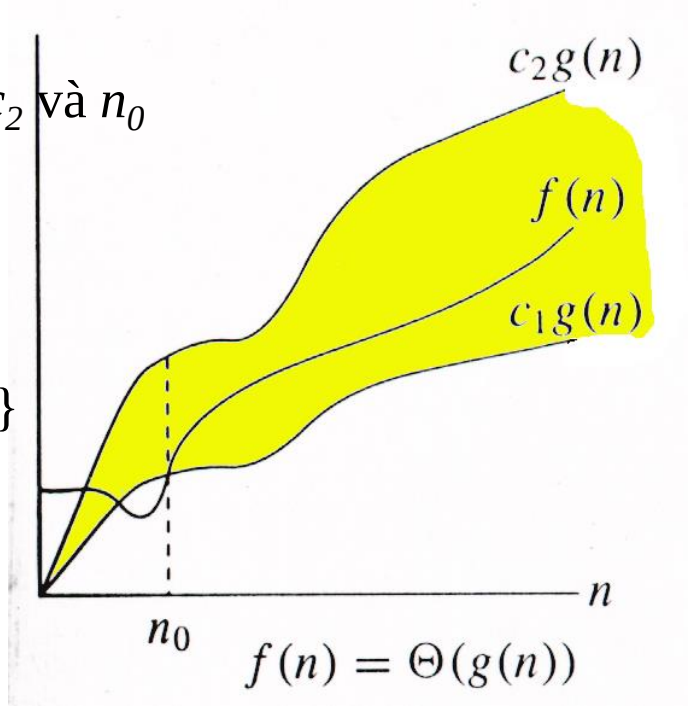
f(n) = Θ(g(n)) nếu tồn tại các hằng số dương c_1, c_2 và n_0 sao cho $c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$ với mọi $n \ge n_0$

Hình thức:

$\Theta(g(n)) = \{ f(n): \exists$ các số $c_1, c_2, n_0 > 0$ sao cho $c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n) \ \forall \ n \ge n_0\}$

- **Ý nghĩa:**

Tập các hàm có tốc độ tăng trưởng tương đương với hàm g(n)



$f(n) \sim g(n)$

Biểu diễn tiệm cận

- **Kí pháp biểu diễn tiệm cận (**Asymptotic Notation)
  - ***Cận chặt: Ký pháp Θ***
    - **Ví dụ:**

- Độ phức tạp thuật toán Sắp xếp chèn $f(n) = an^2+bn+c = \Theta(n^2)$

Chứng minh :

Chọn $c_1 = a/4$, $c_2 = 7a/4$ và $n_0=2.\max(b/a,\text{sqrt}(c/a))$ => đpcm

Kí pháp cận chặt đánh giá trường hợp trung bình

Cùng với kí pháp cận trên cho đánh giá chung về độ phức tạp thuật toán.

- ***Một số kí pháp khác: o (o nhỏ), ω (omega nhỏ), θ (theta nhỏ)***

Định nghĩa tương tự các kí pháp lớn tương ứng, thay ≤, ≥ bằng <, > tương ứng

Biểu diễn tiệm cận

- **Các tính chất của quan hệ tiệm cận**
  - ***Tính bắc cầu :***

$$f(n) = \Theta(g(n)) \;\&\; g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$$
$$f(n) = O(g(n)) \;\&\; g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$$
$$f(n) = \Omega(g(n)) \;\&\; g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$$

  - **Tính phản xạ**:

$$f(n) = \Theta(f(n))$$
$$f(n) = O(f(n))$$
$$f(n) = \Omega(f(n))$$

  - **Tính đối xứng và đối xứng bắc cầu :**

$$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$$
$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

Lớp hàm đánh giá

- **Phân lớp một số hàm đánh giá độ phức tạp tính toán cơ bản**

  - **Hàm 1 (hằng số O(1))**

Số phép tính/thời gian chạy/dung lượng bộ nhớ không phụ thuộc vào độ lớn đầu vào.

Thuật toán với số hữu hạn các thao tác được thực hiện 1 hoặc 1 vài lần.

Ví dụ: thuật toán giải phương trình bậc nhất, bậc hai…

  - **Hàm log*n* (logarit – O(log*n*))**

Các thuật toán có thời gian thực hiện tăng theo kích thước dữ liệu vào với tốc độ hàm logarit.

Ví dụ thuật toán tìm kiếm trên mảng được sắp, thuật toán thao tác trên nhánh con của cây nhị phân đầy đủ…

Lớp hàm đánh giá

- **Phân lớp một số hàm đánh giá độ phức tạp tính toán cơ bản**

  - **Hàm *n* (tuyến tính – O(*n*))**

Các thuật toán có thời gian thực hiện tăng theo kích thước dữ liệu vào với tốc độ tuyến tính. Thường là một số hữu hạn các thao tác với tất cả các dữ liệu vào.

Ví dụ thuật toán tìm kiếm (phần tử, max, min…) trên mảng.

  - **Hàm *n*log*n* (tuyến tính logarit – O(*n*log*n*))**

Các thuật toán để giải các bài toán bằng cách chia thành các bài toán nhỏ hơn, giải một cách độc lập rồi hợp lại để nhận được kết quả của bài toán lớn.

Ví dụ thuật toán sắp xếp nhanh, sắp xếp vun đống.

Lớp hàm đánh giá

- **Phân lớp một số hàm đánh giá độ phức tạp tính toán cơ bản**

  - **Hàm n^2 (đa thức – O(n^m))**

Các thuật toán với các thao tác được thực hiện với trong các vòng lặp lồng nhau. Trường hợp tổng quát n^m. Thông thường đánh giá thuật toán đến n=3,4

Ví dụ thuật toán sắp xếp nổi bọt, nhân ma trận.

  - **Hàm 2^n (lũy thừa – O(m^n))**

Đây là lớp thuật toán có độ phức tạp lớn. Thông thường là các thuật toán đệ quy với lượng dữ liệu đầu vào lớn. Khi *n* đủ lớn, có thể xem như bài toán không giải được theo nghĩa là không nhận được lời giải trong một thời gian hữu hạn.

Lớp hàm đánh giá

- **Tốc độ tăng trưởng các hàm đánh giá độ phức tạp cơ bản**
  - **Bằng số**

$\log n$	n	$n \log n$	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65,536
5	32	160	1,024	32,768	4,294,967,296

  - **Bằng biểu đồ/đồ thị ???**

Quan hệ tiệm cận

- **Chứng minh quan hệ tiệm cận**

  Cho hai hàm xác định độ phức tạp tính toán của thuật toán là $f(n)$ và $g(n)$.
  Cần xác định quan hệ tiệm cận $f(n) = *(g(n))$ với * là O, Ω, Θ

  - Các phương pháp chứng minh quan hệ
    - **Dùng định nghĩa**: Tìm các hằng số c, n_0 thỏa mãn điều kiện
    - **Dùng phương pháp quy nạp**:

      - Ví dụ: log n= O(n) tức là $\log(n) \le c.n$

        Cơ sở quy nạp: $n = 1 \Rightarrow 0 < 1$ đúng

        Giả thiết quy nạp: $\log(n) \le n$ với $n>1$

        Tổng quát: $\log(n+1) \le \log(n+n) = \log(2n) = \log n + 1 \le n+1$
    - **Dùng quan hệ giới hạn** (khi cho n $\to\infty$)

Quan hệ tiệm cận

- **Chứng minh quan hệ tiệm cận**
  - **Dùng quan hệ giới hạn** (khi cho $n \to \infty$)

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \begin{cases} 0 & \Rightarrow f(n) = O(g(n)) \\ \infty & \Rightarrow f(n) = \Omega(g(n)) \\ const & \Rightarrow f(n) = \Theta(g(n)) \\ kxd & \Rightarrow \text{không có quan hê} \end{cases}$$

Ví dụ:

$$\text{Xét} \quad f(n) = n\sqrt{n} \quad \text{và} \quad g(n) = n^2 - n$$

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{n\sqrt{n}}{n^2-n} = \lim_{n\to\infty}\frac{\sqrt{n}}{n-1} = 0$$

$$\Rightarrow f(n) = O(g(n))$$

Quan hệ tiệm cận

- **Quy tắc đánh giá độ phức tạp thuật toán**
  - ***Quy tắc hằng***

    $T(n) = O(c_1.f(n)) = O(f(n))$
  - ***Quy tắc cộng***

    $P = (P1;P2) \Rightarrow T_P(n) = \text{Max}(T_{P1}(n), T_{P2}(n))$

    Ví dụ: Tìm min bằng cách sắp xếp dãy tăng rồi lấy phần tử đầu tiên

    => độ phức tạp là $T_{\min}(n) = \text{Max}(T_{\text{sort}}(n),1) = O(n^2)$.
  - ***Quy tắc nhân***

    $P = (P^*)^n \Rightarrow T_P(n) = n \, . \, T_{P^*}(n)$

    Ví dụ:

    P ≡ *for i=1.. n do P**

    => độ phức tạp $T_P(n) = nT_{P^*}(n))$.

Ví dụ thuật toán

- Các thuật toán sắp xếp
  - Sắp xếp "đơn giản"
    - Bubble Sort
    - Selection Sort
    - Insertion Sort
  - Sắp xếp hiệu quả
    - Quick Sort
    - Merge Sort
    - Heap Sort

https://visualgo.net/en/sorting

Ví dụ thuật toán

- Đánh giá các thuật toán sắp xếp

Algorithm	Worst Case	Best Case	In-place	Stable
Selection Sort	$O(n^2)$	$O(n^2)$	Yes	No
Insertion Sort	$O(n^2)$	O(n)	Yes	Yes
Bubble Sort	$O(n^2)$	$O(n^2)$	Yes	Yes
Bubble Sort 2	$O(n^2)$	$O(n)$	Yes	Yes
Quick Sort	$O(n^2)$	$O(n\log n)$	Yes	No
Merge Sort	$O(n\log n)$	$O(n\log n)$	No	Yes
Heap Sort	$O(n\log n)$	$O(n\log n)$	Yes	Yes

Bài toán “dễ” và “khó”

- **Phân lớp bài toán theo độ phức tạp**

  Có hay không ***một giải thuật với thời gian đa thức*** để giải một bài toán?

  - ***Có***
    - Có thuật toán để giải bài toán với thời gian thực hiện của giải thuật thuộc lớp hàm đa thức.
  - ***Không***
    - Vì mọi thuật toán để giải đều thuộc lớp hàm mũ.
    - Vì không tồn tại thuật toán để giải bài toán.
  - ***Không biết***
    - Nhưng nếu tìm được thuật toán như vậy, có thể sẽ cung cấp một phương thức chung để giải nhiều bài toán khác với thời gian đa thức.

Bài toán "dễ" và "khó"

- **Tính "dễ xử lí"**
  - Một thuật toán giải bài toán trong thời gian đa thức nếu thời gian để thực hiện trong trường hợp xấu nhất thuộc lớp hàm đa thức.
  - ***Bài toán "dễ" (tractable)***: có thể giải được trong thời gian đa thức. Cận trên là hàm đa thức.
    - Ví dụ: Tìm kiếm trong danh sách được sắp ($O(\log n)$); sắp xếp danh sách ($O(n \log n)$).
  - ***Bài toán "khó" (intractable)***: không thể giải trong thời gian đa thức. Cận dưới hoặc cận chặt là hàm mũ.
    - Ví dụ: Thuật toán chuyển tháp Hà Nội ($\Theta(2^n)$); liệt kê các hoán vị khác nhau của n số ($\Theta(n!)$).

Bài toán “dễ” và “khó”

- **Xử lí các bài toán “khó” (intractable)**
  - Tìm kiếm những cải thiện càng nhiều càng tốt (đối với thuật toán), hi vọng sự khả thi. Ví dụ thuật toán quay lui.
  - Giải bài toán trong một số trường hợp đơn giản, đặc biệt nào đó, tránh độ phức tạp hàm mũ.
  - Sử dụng thuật toán xác suất thời gian đa thức, giúp có câu trả lời đúng với xác suất cao nhất. Nghĩa là “thỏa hiệp” tính đúng và quan tâm tới tốc độ.
  - Với các bài toán tối ưu, sử dụng thuật toán xấp xỉ với thời gian đa thức, tuy nhiên không đảm bảo là tìm được lời giải tối ưu nhất.

P và NP

- **Hai kiểu của bài toán cơ bản**
  - ***Bài toán tối ưu***: tìm lời giải làm sao cho một hàm mục tiêu đạt giá trị (nào đó/*max/min)*.
  - ***Bài toán quyết định***: bài toán có 1 trong hai phương án trả lời, YES hoặc NO (hoặc ***true*** hoặc ***false***, hoặc 0 hoặc 1).
  - Nhiều bài toán có thể có cả hai kiểu tối ưu và quyết định.
    - Ví dụ: Bài toán người du lịch; Xếp balo; Tô màu đồ thị…
  - Với nhiều bài toán không phải quyết định, có thể có phát biểu dạng quyết định
    - Ví dụ: Bài toán TSP tối ưu, thêm tham số k, được bài toán: Có hay không đường đi có chi phí *k*.

P và NP

- **Độ phức tạp lớp P**
  - ***Định nghĩa***: Lớp P (*Polynomial*) là tập các bài toán quyết định có thể giải bằng thuật toán với độ phức tạp đa thức.
  - Phát biểu khác: Một bài toán thuộc lớp P nếu
    - Nó là bài toán quyết định
    - Tồn tại thuật toán giải bài toán với độ phức tạp $O(n^k)$, trong đó n là kích thước dữ liệu vào, k là hằng số.
  - Như vậy, P là tập các bài toán quyết định có tính "dễ" (tractable decision problems).

- **Độ phức tạp lớp NP ?**

P và NP

- **Thuật toán không đơn định**
  - NP (Non-deterministically Polynomial) - Không đơn định đa thức.
  - ***Thuật toán không đơn định (non-deterministic algorithm)***: Giải một trường hợp *I* cụ thể của một bài toán quyết định thông qua hai bước:
    - Bước đề xuất (*Guessing stage*): *S* được đề xuất như nghiệm của bài toán (S được sinh ra một cách không đơn định).
    - Bước kiểm chứng (*Verification stage*): Một thuật toán xác định kiểm tra xem *S* có phải là nghiệm của trường hợp *I* của bài toán không (*YES* or *NO*).

P và NP

- **Độ phức tạp lớp NP**
  - Định nghĩa không hình thức: NP là lớp các bài toán quyết định mà
    - "Khó" có thể đưa ra lời giải, nhưng
    - "Dễ" kiểm tra lời giải đã được đề xuất.
  - Định nghĩa hình thức: NP là lớp các bài toán quyết định được giải bằng các thuật toán không đơn định với thời gian đa thức.
  - Bài toán NP
    - Ví dụ: tìm các nhân tử nguyên tố của 13.717.421 – Phức tạp
    - Nhưng dễ kiểm tra rằng 3.607 x 3.803 = 13.717.421

P và NP

- **Độ phức tạp lớp NP**
  - **Ví dụ 1:** Thuật toán không đơn định tìm chu trình Hamilton
    - *Guessing stage*: sinh dãy $v_1\ v_2\ \ldots\ v_n$, với mỗi v_i là một cạnh của đồ thị $G=(V,E)$, $(n = |V|)$.
    - *Verification stage*: kiểm tra xem $v_1\ v_2\ \ldots\ v_n$ có là một đường đi không, nghĩa là $(v_i,v_{i+1})\in E$,

    Dễ thấy bước kiểm chứng có độ phức tạp thời gian là $O(n)$

P và NP

- **Độ phức tạp lớp NP**
  - **Ví dụ 2:** Thuật toán không đơn định tô màu đồ thị
    - *Guessing stage*: sinh một dãy *s* các kí tự $c_1\ c_2\ \ldots\ c_q$, là dãy giả thiết các màu được tô cho đồ thị.
    - *Verification stage*: kiểm tra xem mỗi màu c_i có tô được cho cạnh v_i không.

    Dễ thấy bước kiểm chứng có độ phức tạp thời gian là $O(n^2)$

P và NP

- **Độ phức tạp lớp NP**
  - **Ví dụ 3:** Bài toán thỏa được dạng chuẩn hội (CNF
    - Bài toán: Cho một công thức logic biểu diễn ở dạng chuẩn hội (conjunctive normal form-CNF), có thể tìm dãy giá trị true/false của các biến để CNF là đúng?

      $(A \vee \neg B \vee \neg C) \wedge (\neg A \vee B) \wedge (\neg B \vee D \vee E) \wedge (F \vee \neg D)$

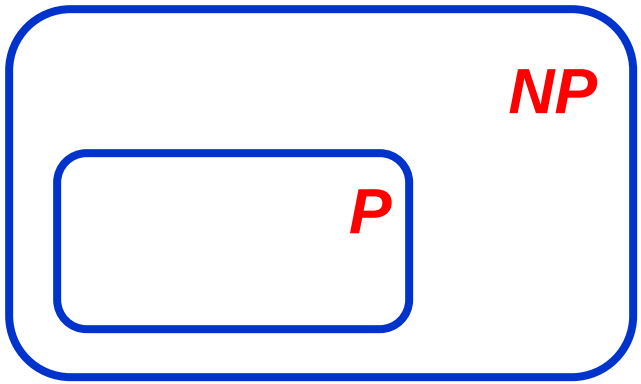
      True or False
    - Đây là bài toán thuộc lớp *NP*.
    - Thuật toán không đơn định:
      - Sinh dãy giá trị true/false của các biến.
      - Kiểm tra với dãy được sinh thì CNF có đúng không.

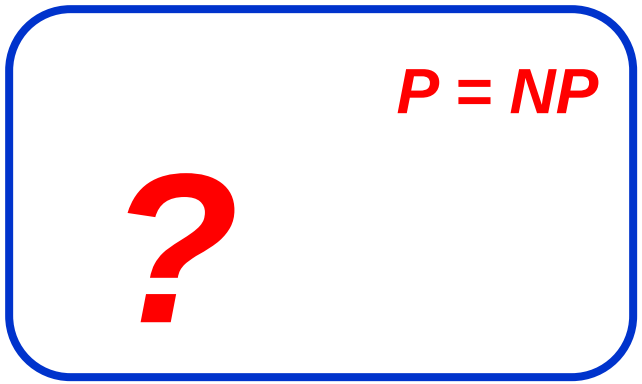
P và NP

- **P và NP**
  - $P \subseteq NP$
    - Mọi bài toán quyết định được giải bằng thuật toán với thời gian đa thức cũng có thể giải được bằng thuật toán không đơn định thời gian đa thức.
  - Chứng minh: bất kì thuật toán đơn định nào đều có thể được sử dụng làm thuật toán để kiểm chứng (verification stage) của một thuật toán không đơn định.
    - Nếu $I \in P$ và A là thuật toán đơn định thời gian đa thức để giải I, sử dụng A như thuật toán kiểm chứng mà bỏ qua bước đề xuất.

P và NP

- **P và NP**
  - *P = NP* - MỘT CÂU HỎI THÁCH THỨC?
    - Các bài toán thuộc lớp NP có thể giải trong thời gian đa thức?
    - Sơ đồ quan hệ:





P và NP

- **P và NP**
  - P = NP? - MỘT CÂU HỎI THÁCH THỨC?
    - Thách thức thế kỉ của lĩnh vực Toán-Tin. Giải thưởng 1TRIỆU USD của hội Toán học Mỹ (claymath) cho khẳng định P=NP hoặc P#NP
      http://www.claymath.org/millennium/
    - Tại sao bài toán lại quan trọng như vậy?
      - *"Nếu P=NP, Một mặt, điều này sẽ giải quyết được rất nhiều vấn đề tin học ứng dụng trong công nghiệp; nhưng mặt khác lại sẽ phá hủy sự bảo mật của toàn bộ các giao dịch tài chính thực hiện qua Internet"*.
      - Mọi "ngân hàng" và các hệ thống bảo mật đều hoảng sợ trước vấn đề lôgic nhỏ bé và cơ bản này!