ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

THIẾT KẾ VÀ ĐÁNH GIÁ THUẬT TOÁN

Bài 2

Phân tích và đánh giá độ phức tạp thuật toán (*Algorithm Analysis*)

Nguyễn Thị Hồng Minh
minhnth@gmail.com

Nội dung

1. **Phân tích thuật toán**
2. **Đánh giá độ phức tạp thuật toán**
3. **Chứng minh tính đúng của thuật toán**
4. **Phân lớp độ phức tạp thuật toán**

Nội dung

1. **Phân tích thuật toán**
2. **Đánh giá độ phức tạp thuật toán**
3. **Chứng minh tính đúng của thuật toán**
4. **Phân lớp độ phức tạp thuật toán**

Phân tích thuật toán

- **Mô hình thuật toán**
  - Hai mô hình mô tả thuật toán: mô hình lý thuyết và mô hình thực tế
    - ***Mô hình lý thuyết***: Thuật toán tương đương máy Turing với các đặc trưng:

- Đơn vị nhớ: là một ô nhớ lưu giữ một kí hiệu trên băng.
- Đơn vị thời gian: là thời gian thực hiện bước chuyển trạng thái.
- Độ phức tạp thời gian: số bước chuyển trạng thái từ ban đầu tới kết thúc.
- Độ phức tạp không gian: Số ô nhớ trên băng sử dụng để thực hiện thuật toán.

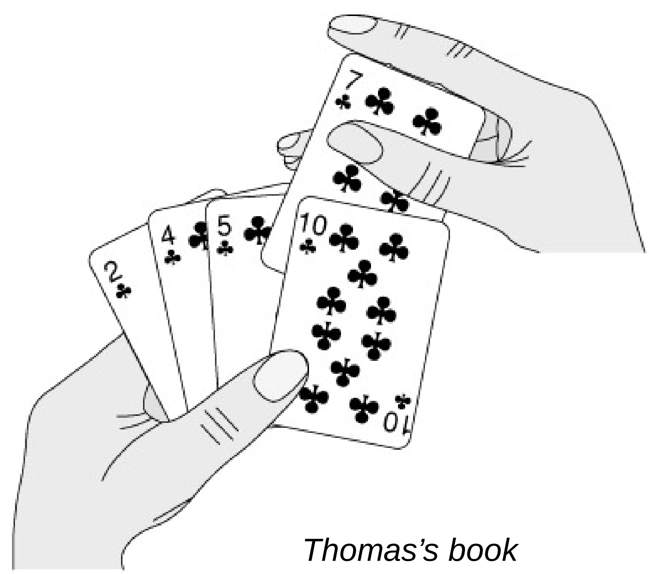
    - ***Mô hình thực tế:*** Thuật toán mô tả theo ngôn ngữ tựa Algol (Pascal):

- Đơn vị nhớ: lưu giữ trọn vẹn một dữ liệu, bất kể nó dài hay ngắn.
- Đơn vị thời gian: là thời gian thực hiện một phép tính số học, logic cơ bản hoặc thậm chí là một câu lệnh “đơn”.
- Độ phức tạp thời gian: Số đơn vị thời gian thực hiện thuật toán => **độ phức tạp tính toán: hàm phụ thuộc vào kích thước dữ liệu vào.**
- Độ phức tạp không gian: yêu cầu về bộ nhớ, băng thông, cổng logic…

Phân tích thuật toán

- **Mô hình thuật toán**
  - Ví dụ thuật toán theo mô hình thực tế: Thuật toán ***Sắp xếp chèn***

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
             A[j+1] = A[j]
             j = j - 1
      }
      A[j+1] = key
  }
}
```



Thomas's book

Phân tích thuật toán

- **Phân tích thuật toán:**
  - Là việc dự đoán:
    - Tài nguyên mà thuật toán đó đòi hỏi
    - Thời gian thực hiện thuật toán: Khó đánh giá cụ thể bằng số lượng các đơn vị thời gian => Đánh giá qua biểu diễn tiệm cận (Asymptotic Performance) trên các mô hình thực tế của thuật toán.
  - Mỗi phân tích thuật toán được thực hiện với mô hình tính toán xác định.
  - Sử dụng mô hình máy truy cập ngẫu nhiên đơn bộ xử lí (generic uniprocessor random-access machine –RAM) với đặc trưng:
    - Thời gian truy cập các ô nhớ là như nhau
    - Các thao tác được thực hiện tuần tự, không có thao tác đồng thời
    - Các lệnh thực hiện với cùng đơn vị thời gian trừ lời gọi hàm, vòng lặp
    - Kích thước một đơn dữ liệu là đồng nhất (1 biến int = 1 biến float) (ngoại trừ một số tính toán cụ thể tới *bit* dữ liệu)

Phân tích thuật toán

- **Kích thước dữ liệu vào (input size)**
  - Độ phức tạp thời gian và không gian của thuật toán là hàm của kích thước dữ liệu vào (có thể là số lượng hoặc giá trị)
  - Ví dụ việc mô tả sự phụ thuộc kích thước dữ liệu vào của các thuật toán
    - Sắp xếp mảng: Số các phần tử mảng cần sắp
    - Nhân số học: Tổng số bit dữ liệu
    - Tính giai thừa: Số cần tính giai thừa
    - Tháp Hà Nội: Số tầng tháp
    - Tìm cây bao trùm của đồ thị: Số đỉnh và cạnh của đồ thị
    - …

Phân tích thuật toán

- **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" (như lệnh gán, tăng, giảm biến…)
  - Các câu lệnh sau được xem như được thực hiện với cùng một thời gian
    - **`y = m * x + b`**
    - **`c = 5 / 9 * (t - 32 )`**
    - **`z = f(x) + g(y)`**
  - Đánh giá thời gian thực hiện của đoạn lệnh sau ?

```
if (a > b)
  a = a – b;
else
  b = b –a;
```

Phân tích thuật toán

- **Phân tích thời gian thực hiện thuật toán *Sắp xếp chèn***

Câu lệnh	Thời gian
`InsertionSort(A, n) {`	
`for i = 2 to n {`	$c_1 n$
`key = A[i]`	$c_2(n\text{-}1)$
`j = i - 1;`	$c_3(n\text{-}1)$
`while (j > 0) and (A[j] > key) {`	$c_4 T$
`A[j+1] = A[j]`	$c_5(T\text{-}(n\text{-}1))$
`j = j - 1`	$c_6(T\text{-}(n\text{-}1))$
`}`	
`A[j+1] = key`	$c_7(n\text{-}1)$
`}`	
`}`	

$T = t_2 + t_3 + \ldots + t_n$ với t_i là số lần biểu thức của câu lệnh **while** được đánh giá ở lần lặp thứ i

Phân tích thuật toán

- **Phân tích thời gian thực hiện thuật toán *Sắp xếp chèn***

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 T + c_5(T - (n-1)) + c_6(T - (n-1)) + c_7(n-1)$$

$$= c_8 T + c_9 n + c_{10}$$

- *T* có thể là ?
  - Trường hợp tốt nhất: thân vòng lặp while không thực hiện lần nào
    - o $t_i = 1$ ➔ $T(n)$ là hàm tuyến tính
  - Trường hợp xấu nhất: thân vòng lặp while thực hiện ở mọi lần lặp
    - o $t_i = i$ ➔ $T(n)$ là hàm bậc 2 vì $T=2+3+4+\ldots+n$
  - Trường hợp trung bình
    - o ???

Phân tích thuật toán

- **Trường hợp tốt nhất**
  - Thời gian thực hiện thuật toán nhanh nhất với bộ dữ liệu đầu vào “lí tưởng”
- **Trường hợp xấu nhất**
  - Thời gian thực hiện thuật toán lâu nhất với bộ dữ liệu đầu vào “tồi nhất”
    - Sắp xếp theo thứ tự ngược lại
    - Tìm kiếm một phần tử không xuất hiện
    - …
  - Đánh giá cận trên của độ phức tạp
- **Trường hợp trung bình**
  - Thời gian thực hiện thuật toán với bộ dữ liệu tổng quát
  - Sử dụng các công cụ xác suất để đánh giá thời gian thực hiện
  - Độ phức tạp tính toán của thuật toán

Nội dung

1. Phân tích thuật toán
2. **Đánh giá độ phức tạp thuật toán**
3. Chứng minh tính đúng của thuật toán
4. Phân lớp độ phức tạp thuật toán

Độ phức tạp thuật toán

- **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…)
  - Tiêu chí đánh giá thuật toán: Xét thuật toán A tính toán trên dữ liệu D
    - Giá về bộ nhớ ($s_A(d)$) là số đơn vị nhớ cần thiết thực hiện thuật toán với một bộ dữ liệu đầu vào kích thước d.
    - Giá về thời gian ($t_A(d)$) là số đơn vị thời gian thực hiện thuật toán với bộ dữ liệu vào đầu vào kích thước d
    - Độ phức tạp của bộ nhớ trong trường hợp xấu nhất
      - $S_A(n) = \max \{s_A(d) \mid d \leq n\}$ $\quad (n = \max |D|)$
    - Độ phức tạp về thời gian trong trường hợp xấu nhất:
      - $T_A(n) = \max \{ t_A(d) \mid d \leq n \}$ $\quad (n = \max |D|)$

Độ phức tạp thuật toán

- **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) ≤ c · g(n) với mọi n ≥ n_0

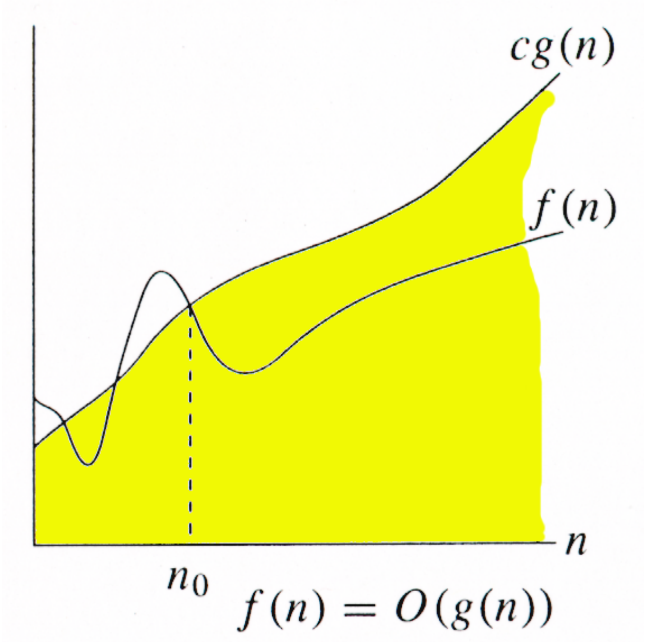
Hình thức:

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

$f(n) \leq c \cdot g(n)\ \forall\ n \geq n_0\}$

- **Ý nghĩa:**

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



Độ phức tạp thuật toán

- **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 :

$an^2 + bn + c \le (a + b + c)n^2 + (a + b + c)n + (a + b + c)$

$\le 3(a + b + c)n^2$ với $n \ge 1$

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

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

- Độ phức tạp thuật toán tìm kiếm $f(n)= an+b = O(n)$???

- Độ phức tạp thuật toán tìm kiếm $f(n)= an+b = O(n^2)$???

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

Độ phức tạp thuật toán

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

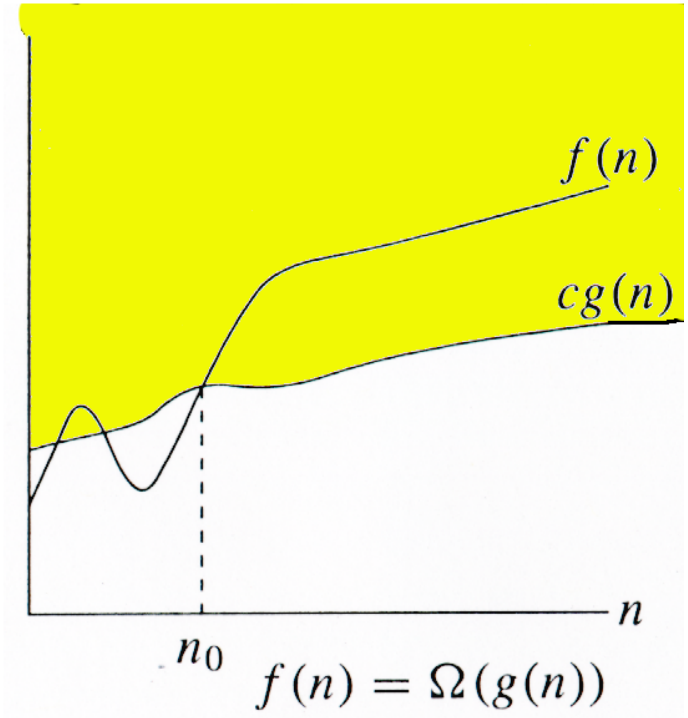
f(n) = Ω (g(n)) nếu tồn tại các hằng số dương c và n_0 sao cho f(n) ≥ c · g(n) với mọi n ≥ n_0

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)



Độ phức tạp thuật toán

- **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!

Độ phức tạp thuật toán

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

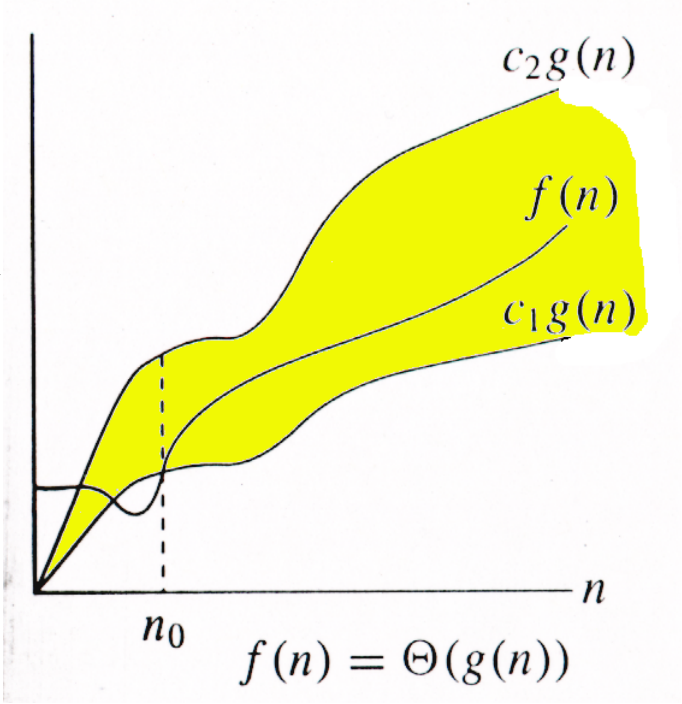
f(n) = Θ(g(n))   nếu tồn tại các hằng số dương c_1, c_2 và n_0

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)

Độ phức tạp thuật toán

- **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 $n_0=2.\max(b/a,\mathrm{sqrt}(c/a))$ => 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

Xem thêm Thomas's book, chương 3, mục 3.1

Độ phức tạp thuật toán

- **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 :**

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

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

Độ phức tạp thuật toán

- **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…

Độ phức tạp thuật toán

- **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.

Độ phức tạp thuật toán

- **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.

Độ phức tạp thuật toán

- **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 ???**

Độ phức tạp thuật toán

- **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 $n \to \infty$)

Độ phức tạp thuật toán

- **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:

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

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

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

Độ phức tạp thuật toán

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

Nếu thuật toán P có độ phức tạp $T(n)= O(c_1.f(n))$ thì P có độ phức tạp $O(f(n))$

  - ***Quy tắc cộng***

Nếu thuật toán P được thực hiện bằng Pl; P2 thì P có độ phức tạp bằng độ phức tạp lớn nhất trong hai thuật toán Pl và P2:

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

Ví dụ: Tìm max bằng cách sắp xếp dãy giảm rồi lấy phần tử đầu tiên

=> độ phức tạp là $O(n^2)$.

  - ***Quy tắc nhân***

Nếu thuật toán P được thực hiện bằng cách thực hiện *n* lần thuật toán P* thì độ phức tạp của thuật toán P bằng tích *n* lần độ phức tạp của thuận toán P*.

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

Ví dụ: P là *for i=1.. n do P** thì $T_P(n)= nT_{P*}(n))$.

Chứng minh tính đúng của thuật toán

- **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)

Xem thêm Thomas's book, mục 2.1

Chứng minh tính đúng của thuật toán

- **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 của thuật toán

- **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 => 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 của thuật toán

- **Chứng minh tính đúng đối với thuật toán không đệ quy**
  - Phương pháp bất biến vòng lặp (*loop invariant*)
    - 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.
    - 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
  - 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 của thuật toán

- **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 ($t = n+1-2+1=n$)

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

Phân lớp bài toán theo độ phức tạp

- **Lớp P và NP**
  - Lớp P: Là lớp các bài toán giải được bằng thuật toán đơn định (*deterministic*) trong thời gian đa thức. Đây là lớp các bài toán thực tế giải được.

    Ví dụ các thuật toán thuộc lớp P: Sắp xếp; Nhân ma trận; Tìm đường đi ngắn nhất; Cây bao trùm tối thiểu, …
  - Lớp NP: Là lớp các bài toán giải được bằng thuật toán không đơn định (*nondeterministic)* trong thời gian đa thức. còn nếu giải bằng thuật toán đơn định thì độ phức tạp của nó là hàm mũ (trên đa thức)

    Ví dụ các thuật toán thuộc lớp NP: Xếp balo, tháp Hà Nội, bài toán phân hoạch: cho $a_1, ..., a_n$. ? $\exists$ T $\subset$ {1 , 2, ..., n}: $\sum_{i \in T} a_i = \frac{1}{2} \sum_{i=1}^{n} a_i$,…

Phân lớp bài toán theo độ phức tạp

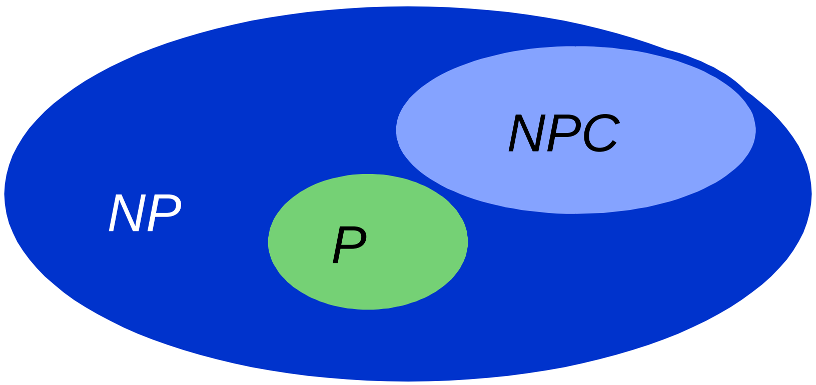
- **Khái niệm dẫn được đa thức**
  - Bài toán A dẫn được đa thức từ bài toán B (A $\propto$ B), nếu một khi B giải được bằng thuật toán đơn định đa thức thì A cũng giải được bằng thuật toán đơn định đa thức - ***A dễ hơn B***
  - Bài toán A tương đương đa thức với B (A ~ B)nếu A $\propto$ B và B $\propto$ A.

- **Lớp NPH, NPC**
  - Lớp NP-khó (NP-Hard/NPH): Bài toán A được gọi là thuộc lớp NP-khó nếu mọi bài toán thuộc lớp NP đều dẫn được đa thức từ nó.
  - Lớp NP-đầy đủ (NP-Complete/NPC): Bài toán A được gọi là bài toán NP-đầy đủ nếu A thuộc lớp NP và A là NP-khó.

  Lớp NPC nói chung là lớp các bài toán thực tế không giải được.

Phân lớp bài toán theo độ phức tạp

- Quan hệ các lớp P, NP, NPC



- Lớp NPC là lớp con của những bài toán khó nhất trong lớp NP.
- Đóng góp của vấn đề nghiên cứu NP-đầy đủ: cung cấp một cơ chế để xác định một bài toán mới trong một lĩnh vực là “dễ” hay “khó”.

Xem thêm Thomas’s book, chương 34

Phân lớp bài toán theo độ phức tạp

- Một số kỹ thuật để đối phó với những bài toán NP-đầy đủ:
  - Dùng “giải thuật xấp xỉ “(approximation algorithm) để tìm lời giải xấp xỉ gần tối ưu (near-optimal).
  - Phát triển một giải thuật để tìm ra lời giải trong một số trường hợp cụ thể, xác định nào đó.
  - Sử dụng những giải thuật có độ phức tạp hàm mũ nhưng hữu hiệu, ví dụ như giải thuật quay lui.
  - Đưa heuristic (kinh nghiệm) vào giải thuật để tăng thêm hiệu quả của giải thuật.
- Một số lĩnh vực có những bài toán NP-đầy đủ:
  - Giải tích số (numerical analysis),
  - Xử lý dòng ký tự (string processing),
  - Mô hình hóa hình học (geometry modeling)
  - Xử lý đồ thị (graph processing).
  - …