Độ phức tạp

Khoa Khoa học máy tính

Nôi dung

- Khái niệm độ phức tạp
- Độ phức tạp: lý thuyết và thực tế
- Đánh giá độ phức tạp: ba trường hợp
- Các hàm tiệm cận
- Độ phức tạp thực tế

Độ phức tạp

- Độ phức tạp
  - Độ phức tạp của một thuật toán là đo lường số các thao tác cơ bản mà thuật toán thực hiện trên một bộ dữ liệu vào
- Phụ thuộc vào dữ liệu vào
  - Độ phức tạp là một hàm phụ thuộc vào kích thước n của bộ dữ liệu vào
  - Độ phức tạp có ý nghĩa khi n lớn

- Đánh giá độ phức tạp thuật toán nhằm
  - Nghiên cứu hoạt động của thuật toán
    - Tối ưu hay không
    - Phát hiện những phần phức tạp, làm chậm thuật toán
  - So sánh các giải pháp khác nhau trong cùng ngữ cảnh
    - vấn đề
    - ngôn ngữ
    - máy tính

thời gian và bộ nhớ

- Đánh giá độ phức tạp
  - thời gian thực thi
  - bộ nhớ sử dụng
  - ➔ thường mâu thuẩn nhau
- Chúng ta thường tìm cách giảm độ phức tạp về mặt thời gian
  - bỏ qua độ phức tạp về mặt bộ nhớ
- Dẫn đến
  - thường làm tăng độ phức tạp về mặt bộ nhớ
  - làm tăng « độ phức tạp trí tuệ » của thuật toán
    - Thời gian phát triển
    - Rủi ro xảy ra lỗi
    - Chi phí bảo trì

lý thuyết và thực tế

- Hai phương pháp đánh giá độ phức tạp về mặt thời gian
  - Phương pháp lý thuyết
  - Phương pháp thực tế

- Phương pháp lý thuyết
  - đánh giá hoạt động của thuật toán khi kích thước dữ liệu n rất lớn
  - hoạt động của thuật toán được đánh giá bởi các hàm của n
  - không tính đến các chi tiết của thuật toán

lý thuyết và thực tế

- Phương pháp thực tế
  - Đánh giá một cách chi tiết thuật toán
  - Tất cả các câu lệnh đều được xem xét
  - Đánh giá riêng rẽ các hoạt động khác nhau
    - phép cộng/trừ số nguyên
    - phép nhân số nguyên
    - phép so sánh
    - thao tác đọc/ghi
    - truy cập bộ nhớ
    - …
  - Mang lại các kết quả rất chi tiết
    - nhưng thường phức tạp
    - thường khó để so sánh (do đánh giá riêng rẽ)
  - Phát hiện các vùng phức tạp
  - Hỗ trợ tối ưu chương trình

lý thuyết và thực tế

- Quan hệ giữa phương pháp lý thuyết và phương pháp thực tế
  - Thường chỉ đánh giá tập các **thao tác cơ bản**
    - thời gian thực thi thuật toán tỷ lệ thuận với số các thao tác này
    - Ví dụ
      - Số phép so sánh và trao đổi dữ liệu đối với thuật toán sắp xếp
      - Số các phép cộng và phép nhân đối với thuật toán nhân hai ma trận
      - Số các truy cập đĩa đối với thuật toán thao tác trên CSDL
  - Các thao tác cơ bản được đánh giá riêng rẽ
  - Sự lựa chọn tập các thao tác cơ bản phụ thuộc
    - kích thước dữ liệu
    - chi tiết cần đánh giá
    - khả năng phân tích
    - …

Tính độ phức tạp

- Dựa vào ngữ pháp của thuật toán, chúng ta có các nguyên tắc tính độ phức tạp như sau:
  - Đối với các thao tác cơ bản là các lệnh tuần tự thì cộng dồn số các thao tác
  - Đối với lệnh điều kiện, giả sử P(X) là số thao tác cơ bản của cấu trúc lệnh X, thì:

    $P(\underline{\text{if}}\ C\ \underline{\text{then}}\ I_1\ \underline{\text{else}}\ I_2) \leq P(C) + \max(P(I_1), P(I_2))$
  - Đối với các lệnh lặp, số các thao tác cơ bản trong vòng lặp là $\sum P(i)$, trong đó i biến điều khiển lặp, P(i) số thao tác cơ bản ở lần lặp thứ i

Tính độ phức tạp

- Đối với lời gọi hàm:
  - Nếu không có hàm đệ quy, chúng ta luôn có thể tổ chức sao cho mỗi một hàm gọi hàm khác mà số thao tác cơ bản của hàm đó đã xác định
  - Nếu là hàm đệ quy, tính số các thao tác cơ bản thường dẫn đến **hệ thức truy hồi**
    - **Ví dụ**

```
function factorial(n)
Begin
    If (n = 1) then
        factorial = 1
    Else
        factorial = n * factorial (n-1)
    EndIf
End
```

Thao tác cơ bản: số phép nhân

C(0) = 0
C(n) = C(n-1) + 1

Vậy: C(n) = n

Tính độ phức tạp: ba trường hợp

- Độ phức tạp phụ thuộc vào dữ liệu sử dụng bởi thuật toán

- Trường hợp tốt nhất
  - Dữ liệu được tổ chức sao cho thuật toán hoạt động hiệu quả nhất
  - Giá trị nhỏ nhất của độ phức tạp thực tế
- Trường hợp xấu nhất
  - Dữ liệu được tổ chức sao cho thuật toán hoạt động kém hiệu quả nhất
  - Giá trị lớn nhất của độ phức tạp thực tế
- Trường hợp trung bình
  - Dữ liệu tương ứng với sự tổ chức được xem là trung bình
  - Không đơn giản để xác định
  - Thường có ý nghĩa nhất
  - Không là trung bình của độ phức tạp tốt nhất và dộ phức xấu nhất

Tính độ phức tạp: ba trường hợp

- D_n : tập dữ liệu kích thước n
- $C_A(d)$: độ phức tạp của thuật toán A đối với dữ liệu d của tập D_n
- p(d) : xác xuất xuất hiện của d

- Độ phức tạp trong trường hợp tốt nhất: $Cmin_A(n)$
  - $Cmin_A(n) = \min\{C_A(d), d \in D_n\}$
- Độ phức tạp trong trường hợp xấu nhất: $Cmax_A(n)$
  - $Cmax_A(n) = \max\{C_A(d), d \in D_n\}$
- Độ phức tạp trung bình: $Cmoy_A(n)$
  - $Cmoy_A(n) = \sum p(d).C_A(d)$

Tính độ phức tạp: ba trường hợp

- Nếu xác xuất xuất hiện các dữ liệu là như nhau, thì
  - $\text{card}(D_n) = |D_n|$
  - $\text{Cmoy}_A(n) = (1/|D_n|) \sum C_A(d)$
- Độ phức tạp trung bình nói chung là không thể xác định chính xác
- Điều chắc chắn
  - $\text{Cmin}_A(n) \leq \text{Cmoy}_A(n) \leq \text{Cmax}_A(n)$

Tính độ phức tạp: ba trường hợp

- Ví dụ 1: nhân hai ma trận

```
matrix_product (A, B, C, n)
Begin
        For i from 1 to n
                For j from 1 to n
                        C(i,j) = 0
                        For k from 1 to n
                                C(i,j) = C(i,j) + A(i,k) * B(k,j)
                        EndFor
                EndFor
        EndFor
End
```

Quan hệ: $Cmin_A(n)$? $Cmoy_A(n)$? $Cmax_A(n)$

Tính độ phức tạp: ba trường hợp

- Ví dụ 2: tìm kiếm tuần tự trong một danh sách

```
function lookup (L, X, n)
Begin
        i = 1
        While ((i ≤ n) and (L(i) ≠ X))
                i = i + 1
        EndWhile
        If (i > n) then
                i = 0
        EndIf
        lookup = i
End
```

Thao tác cơ bản: số các phép so sánh

- Trường hợp tốt nhất: L(1) = X, $Cmin_A(n) = 1$
- Trường hợp xấu nhất: X không có trong L hoặc L(n) = X, $Cmax_A(n) = n$

Tính độ phức tạp: ba trường hợp

- Ví dụ 2: độ phức tạp trung bình
  - q: xác xuất X ở trong danh sách L
  - X trong L: khả năng xuất hiện tại các vị trí như nhau
  - $D_{n,i}$: tập dữ liệu với X ở vị trí i, $p(D_{n,i}) = q/n$
  - $D_{n,0}$: tập dữ liệu với X không thuộc L, $p(D_{n,0}) = 1-q$
  - $C_A(D_{n,i}) = i$ và $C_A(D_{n,0}) = n$

$$Cmoy_A(n) = \sum_{i=0}^{n} p(D_{n,i}) C_A(D_{n,i}) = (1-q)\, n + q/n \sum_{i=1}^{n} i$$

$$= (1-q)\, n + q(n+1)/2$$

Tính độ phức tạp: ba trường hợp

- Ví dụ 2: độ phức tạp trung bình
  - Nếu q = 1 (X thuộc L), thì $Cmoy_A(n) = (n+1)/2$
  - Nếu q = ½ (), thì $Cmoy_A(n) = (3n+1)/4$

  - Trường hợp trung bình
    - X nằm ở giữa danh sách L, tại vị trí n/2 hoặc (n+1)/2
    - độ phức tạp là n/2 hoặc (n+1)/2

Các hàm tiệm cận

- Trong trường hợp tổng quát, chúng ta thường không tính độ phức tạp chính xác, mà tính độ phức tạp tiệm cận (khi n rất lớn)
- Chúng ta xem xét các hàm (asymptotic notations) theo n và thay đổi của hàm khi mà n rất lớn

- Các hàm tiệm cận cho phép
  - đánh giá độ hiệu quả của thuật toán
  - so sánh hiệu quả của các thuật toán

Các hàm tiệm cận

- Kí hiệu o nhỏ
  - f = o(g) khi n $\to \infty$ nếu:

$$\forall c > 0, \exists n_0 > 0, \forall n \geq n_0, 0 \leq f(n) < cg(n)$$

  - g(n) là tiệm cận trên của f(n) với **mọi** hằng số c
  - f(n) tăng chậm hơn g(n):

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

  - Ví dụ
    - $2n = o(n^2)$
    - $n^2 = o(n^4)$

Các hàm tiệm cận

- Kí hiệu O lớn
  - f = O(g) khi n $\to \infty$ nếu:

$$\exists c > 0, \exists n_0 > 0, 0 \le f(n) \le cg(n), \forall n \ge n_0$$

  - g(n) là tiệm cận trên của f(n) với **một số** hằng số c
    - Nếu f = o(g) thì f = O(g)
  - f(n) không tăng nhanh hơn g(n)
  - Ví dụ
    - $2n = O(n^2)$
    - $2n^2 = O(n^2)$, nhưng $2n^2 \neq o(n^2)$

Các hàm tiệm cận

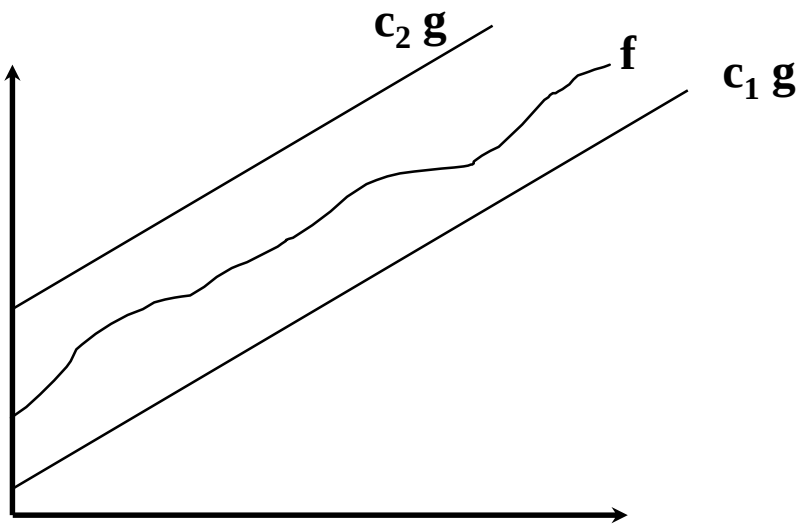
- Kí hiệu Θ
  - f = Θ(g) khi n $\to \infty$ nếu:

$$\exists c_1 > 0, \exists c_2 > 0, \exists n_0 > 0, 0 \le c_1 g(n) \le f(n) \le c_2 g(n), \forall n \ge n_0$$

  - f(n) và g(n) tăng cùng tốc độ (so sánh trong nghĩa độ phức tạp)
  - Ví dụ
    - $f(n) = n^2/2 - 3n = n^2(1/2 - 3/n)$, $g(n) = n^2$
    - xác định c_1 và c_2 sao cho: $c_1 <= 1/2 - 3/n <= c_2$
    - Với $c_2 = 1/2$, $n_0 = 12$, $c1 = 1/2 - 3/12 = 1/4$
    - Vậy $n^2/2 - 3n = \Theta(n_2)$

Các hàm tiệm cận

- Kí hiệu Θ
  - định nghĩa tiệm cận trên và tiện cận dưới của f(n)



  - Nếu f = Θ(g) thì:
    - f = O(g)
    - g = O(f)

Các hàm tiệm cận

- Kí hiệu Ω
  - f = Ω(g) khi n $\to \infty$ nếu:

$$\exists c > 0, \exists n_0 > 0, 0 \le cg(n) \le f(n), \forall n \ge n_0$$

  - g(n) là tiệm cận dưới của f(n) với **một số** hằng số c
  - f(n) tăng ít nhất cũng nhanh bằng g(n)
  - Ví dụ
    - $n^2 = \Omega(n)$

Các hàm tiệm cận

- Các tính chất
  - Nếu f = Θ(g) thì f = O(g) và f = Ω(g)
    - Suy ra từ định nghĩa
  - Nếu f = Θ(g) thì g = Θ(f)
    - Chứng minh

$$\exists(c_1, c_2, n_0), c_1 > 0, c_2 > 0, \forall n > n_0,\ 0 \le c_1 g(n) \le f(n) \le c_2\, g(n)$$

$$\exists(c_2, n_0),\ c_2 > 0,\ \forall n > n_0,\ \ 0 \le \frac{1}{c_2} f(n) \le g(n)$$

$$\exists(c_1, n_0),\ c_1 > 0,\ \forall n > n_0,\ \ 0 \le g(n) \le \frac{1}{c_1} f(n)$$

Các hàm tiệm cận

- Hàm tương đương
  - f ~ g khi n $\to \infty$ nếu:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$$

  - f(n) và g(n) tăng cùng tốc độ (một cách chính xác)
  - f(n) và g(n) có cùng các giới hạn
  - Ví dụ
    - f(n) = n^2 - n và g(n) = n^2
    - f(n) ~ g(n)

Các hàm tiệm cận

- Độ phức tạp lý thuyết khi $n \to \infty$

$f = o(g)$	độ phức tạp của f < độ phức tạp của g
$f = O(g)$	độ phức tạp của f $\leq$ độ phức tạp của g (theo các hằng số xác định)
$f = \Theta(g)$	độ phức tạp của f cùng cở độ phức tạp của g (theo các hằng số xác định)
$f = \Omega(g)$	độ phức tạp của f $\geq$ độ phức tạp của g (theo các hằng số xác định)
$f = \sim(g)$	độ phức tạp của f = độ phức tạp của g

Các hàm tiệm cận

- Ví dụ 1
  - Giả sử $Cmin_A(n) = 8n^2 + 2n + 1$ và $Cmin_B(n) = 1000n^2 + 10n + 2$ là các hàm biểu diễn thoài gian thực thi của hai phương pháp cài đặt cùng một thuật toán trong trường hợp xấu nhất. Xác định O của hai hàm trên.
  - $Cmin_A(n) = 8n^2 + 2n + 1$

    $\leq 8n^2 + 2n^2 + n^2$ với $n \geq 1$

    $= 11n^2$

    chọn $c = 11$, $n_0 = 1$, thì $Cmin_A(n) = O(n^2)$
  - $Cmin_B(n) = 1000n^2 + 10n + 2$

    $\leq 1000n^2 + 10n^2 + 2n^2$ với $n \geq 1$

    $= 1012n^2$

    chọn $c = 1012$, $n_0 = 1$, thì $Cmin_B(n) = O(n^2)$
  - Thuật toán: $O(n^2)$

Các hàm tiệm cận

- Ví dụ 2
  - Chứng minh rằng $\forall k \in N$: $\sum_{i=1}^{n} i^k = \theta(n^{k+1})$
  - Ta có: $\sum_{i=1}^{n} i^k \le \sum_{i=1}^{n} n^k = n.n^k = n^{k+1}$
  - Vậy: $\sum_{i=1}^{n} i^k = O(n^{k+1})$ với c = 1
  - Ta có: $\sum_{i=1}^{n} i^k \ge \sum_{i=\lceil \frac{n}{2} \rceil}^{n} i^k \ge \sum_{i=\lceil \frac{n}{2} \rceil}^{n} (\frac{n}{2})^k \ge \frac{n}{2}(\frac{n}{2})^k = \frac{n^{k+1}}{2^{k+1}}$
  - Vậy: $\sum_{i=1}^{n} i^k = \Omega(n^{k+1})$ với c = $1/2^{k+1}$

Các hàm tiệm cận

- Bài tập
  - Giả sử hàm f(n) = log (2n + α) được định nghĩa, chứng minh rằng: f(n) = Θ(log(n))

Hệ thức truy hồi

- Như đã trình bày, tính độ phức tạp hàm đệ quy thường dẫn đến hệ thức truy hồi
- Các họ hệ thức truy hồi thường gặp
  - Truy hồi bậc nhất
    - tuyến tính: $a_n = n\, a_{n-1} + g(n)$
    - không tuyến tính: $a_n = 1/(n + a_{n-1}) + g(n)$

      hạng đầu tiên phải có giá trị
  - Truy hồi bậc hai
    - tuyến tính: $a_n = b\, a_{n-1} + c\, a_{n-2} + g(n)$
    - không tuyến tính: $a_n = 1/(a_{n-1} + a_{n-2}) + g(n)$

      hai hạng đầu tiên phải có giá trị
  - Truy hồi bậc k
    - tuyến tính: $a_n = b_1\, a_{n-1} + b_2\, a_{n-2} + \ldots + b_k\, a_{n-k} + g(n)$
    - tuyến tính thuần nhất hệ số hằng số:

      $a_n = b_1\, a_{n-1} + b_2\, a_{n-2} + \ldots + b_k\, a_{n-k}$
    - không tuyến tính: …

      k hạng đầu tiên phải có giá trị

Hệ thức truy hồi

- Các họ hệ thức truy hồi thường gặp (tiếp)
  - Truy hồi hoàn toàn
    - tuyến tính:

      $a_n = b_1 a_{n-1} + b_2 a_{n-2} + \ldots + b_k a_{n-k} + \ldots + + b_{n-1} a_1 + g(n)$
    - không tuyến tính: …

      n-1 hạng đầu tiên phải có giá trị
  - Truy hồi phân hoạch

    $a_n = b\, a_{\lfloor n/2 \rfloor} + c\, a_{\lceil n/2 \rceil} + g(n)$

    hoặc dưới dạng: $a_n = c\, a_{n/b} + g(n)$, với điều kiện n/b là số nguyên
  - Nhiều trường hợp, rất phức tạp để giải hệ thức truy hồi
  - Đề cập đến giải hệ thức
    - truy hồi tuyến tính bậc nhất
    - Truy hồi tuyến tính thuần nhất bậc 2 hệ số hằng số
    - truy hồi phân hoạch

Hệ thức truy hồi

- Truy hồi tuyến tính bậc nhất (1)
  - Dạng hệ số hằng số: $a_n = b\, a_{n-1} + g_n$, biết trước a_0

$a_n = b\, a_{n-1} + g_n$

$a_{n-1} = b\, a_{n-2} + g_{n-1}$ * với hệ số = b

$a_{n-2} = b\, a_{n-3} + g_{n-2}$ * với hệ số = b^2

…

$a_1 = b\, a_0 + g_1$ * với hệ số = b^{n-1}

$$a_n = b^n a_0 + \sum_{i=1}^{n} g_i\, b^{n-i}$$

Hệ thức truy hồi

- Truy hồi tuyến tính bậc nhất (2)
  - Dạng: $a_n = b_n a_{n-1} + g_n$, biết trước a_0

Đặt: $a_n = b_1 b_2 \dots b_n y_n$, $a_0 = y_0$

Vậy: $b_1 b_2 \dots b_n y_n = b_1 b_2 \dots b_n y_{n-1} + g_n$

Hay: $y_n = y_{n-1} + g_n/(b_1 b_2 \dots b_n)$

Dẫn đến: $$y_n = y_0 + \sum_{i=1}^{n} \frac{g_i}{b_1 b_2 \dots b_i}$$

Cuối cùng: $$a_n = \prod_{i=1}^{n} b_i \ (a_0 + \sum_{i=1}^{n} \frac{g_i}{b_1 b_2 \dots b_i})$$

Hệ thức truy hồi

- Truy hồi tuyến tính bậc nhất (3)
  - Ví dụ tính:
    - $a_n = 4\ a_{n-1} - 1, \quad a_0 = 1/3$
    - $a_n = 4\ a_{n-1} - 1, \quad a_0 = 0.333$
    - $a_n = 3\ a_{n-1}^2$ et $a_0 = 1$
      có thể đặt $b_n = \log_2 a_n$

Hệ thức truy hồi

- Truy hồi tuyến tính thuần nhất bậc 2 hệ số hằng số (1)
  - Dạng: $a_n = ba_{n-1} + ca_{n-2}$ (1)
  - Nghiệm của hệ thức trên có dạng $a_n=r^n$, r là hằng số
  - Nghĩa là: $r^n = br^{n-1} + cr^{n-2}$
  - Chia hai vế của hệ thức cho r^{n-2}, ta có:

    $$r^2 = br + c$$
  - Khi đó $a_n=r^n$ là nghiệm của hệ thức truy hồi khi r là nghiệm của phương trình

    $$r^2 - br - c = 0 \quad (2)$$

    (*gọi là phương trình đặc trưng của hệ thức truy hồi*)
  - Trong trường hợp tổng quát, ta thu được hai nghiệm phân biệt r_1 và r_2
  - Khi đó, $a_n = \alpha r_1^n + \beta r_2^n$ cũng là nghiệm của của (1), với α và β là các hằng số
    - Chứng minh …

Hệ thức truy hồi

- Truy hồi tuyến tính thuần nhất bậc 2 hệ số hằng số (2)
  - $a_n = \alpha r_1^n + \beta r_2^n$
  - các hệ số α và β được xác định bởi các điều kiện đầu

    $\alpha + \beta = a_0$

    $\alpha r_1 + \beta r_2 = a_1$

  - Ví dụ
    - Tìm nghiệm của hệ thức truy hồi $a_n = a_{n-1} + 2a_{n-2}$ với các điều kiện đầu $a_0=2$ và $a_1=7$
    - Phương trình đặc trưng của hệ thức truy hồi: $r^2-r-2=0$
    - Nghiệm của phương trình đặc trưng: $r_1 = 2$ và $r_2 = -1$
    - Vậy $a_n = \alpha (2)^n + \beta (-1)^n$
    - Thay các điều kiện đầu, có

      $a_0 = \alpha + \beta = 2$

      $a_1 = 2\alpha - \beta = 7$
    - Vậy: $\alpha = 3$, $\beta = -1$
    - Nghiệm của hệ thức truy hồi: $a_n = 3.2^n -(-1)^n$

Hệ thức truy hồi

- Truy hồi tuyến tính thuần nhất bậc 2 hệ số hằng số (3)
  - Nếu phương trình đặc trưng có nghiệm kép thì nghiệm của (1) là:

$$a_n = (\alpha n + \beta) r^n$$

  - Ví dụ
    - Tìm nghiệm của hệ thức truy hồi $a_n = 4a_{n-1} - 4a_{n-2}$ với các điều kiện đầu $a_0=3$ và $a_1=8$
    - Phương trình đặc trưng của hệ thức truy hồi: $r^2-4r+4=0$
    - Phương trình đặc trưng có nghiệm kép: $r = 2$
    - Vậy $a_n = (\alpha n + \beta)(2)^n$
    - Thay các điều kiện đầu, có

$$a_0 = 3 = \beta$$

$$a_1 = (\alpha + \beta)2 = 8$$

    - Vậy: $\alpha = 1, \ \beta = 3$
    - Nghiệm của hệ thức truy hồi: $a_n = 2^n + 3.2^n$

Hệ thức truy hồi

- Truy hồi phân hoạch (1)
  - Truy hồi dạng: $a_n = b\, a_{\lfloor n/2 \rfloor} + c\, a_{\lceil n/2 \rceil} + g(n)$
  - Phân tích bài toán thành hai bài toán con
    - kích thước $\lfloor n/2 \rfloor$ và $\lceil n/2 \rceil$
    - vậy, thông thường b = c (= 1)
    - g(n): chi phí phân tích và trộn các giải pháp con
  - Khi n lớn, có thể xem
    $\lfloor n/2 \rfloor = \lceil n/2 \rceil = n/2$
  - Vậy, hệ thức truy hồi trở thành
    $$a_n = 2\, a_{n/2} + g(n)$$

Hệ thức truy hồi

- Truy hồi phân hoạch (2)
  - Thường được xét dưới dạng:

    $a_n = c\ a_{n/b} + g(n)$, với $n=b^k$

  - Hay $a_{b^k} = c\ a_{b^{k-1}} + g(b^k)$

  - Đặt $t_k = a_{b^k}$
  - Vậy ta có

$$t_k = c\ t_{k-1} + g(b^k)$$
$$t_0 = 1\ (= a_1 = 1)$$

$$t_k = c\ t_{k-1} + g(b^k)$$
$$t_{k-1} = c\ t_{k-2} + g(b^{k-1})$$
$$\ldots$$
$$t_1 = c\ t_0 + g(b^1)$$

Hệ thức truy hồi

- Truy hồi phân hoạch (3)

$t_k = c\,t_{k-1} + g(b^k)$

$t_{k-1} = c\,t_{k-2} + g(b^{k-1})$ *c

$t_{k-2} = c\,t_{k-3} + g(b^{k-2})$ *c^2

…

$t_1 = c\,t_0 + g(b^1)$ *c^{k-1} $t_0=1$

$$t_k = c^k + \sum_{i=0}^{k-1} c^i g(b^{k-i})$$

với $n = b^k \Leftrightarrow k = \log_b n$

$b^{k-i} = n / b^i$

$$c^{\log_b n} = n^{\log_b c}$$

$$a_n = n^{\log_b c} + \sum_{i=0}^{k-1} c^i g(\frac{n}{b^i})$$

Hệ thức truy hồi

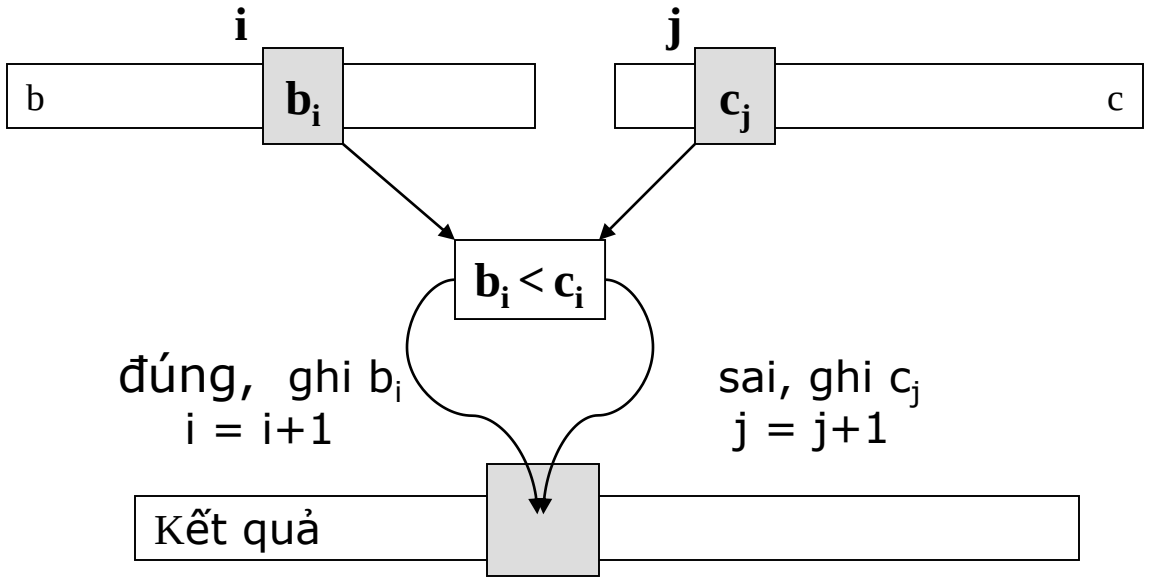
- Ví dụ (1)
  - Tính độ phức tạp của thuật toán sắp xếp trộn (merge sort)

```
merge-sort (A, p, q)
Begin
    If (p < q) then
        r = (p+q)/2
        merge-sort (A, p, r)
        merge-sort (A,r + 1, q)
        merge (A, p, r, q)
    EndIf
End
```

  - Phân tích thành hai bài toán con có kích thước $\lfloor n/2 \rfloor$ và $\lceil n/2 \rceil$

Hệ thức truy hồi

- Ví dụ (2)
  - Trộn hai danh sách b và c kích thước n/2 đã được sắp xếp: merge()



  - Nhiều nhất sử dụng n-1 phép so sánh

Hệ thức truy hồi

- Ví dụ (3)
  - Vậy ta có hệ thức truy hồi ($a_n = C(n)$)

$$a_n = 2\,a_{n/2} + n - 1$$

$$a_n = n^{\log_b c} + \sum_{i=0}^{k-1} c^i g(\frac{n}{b^i}) \Rightarrow a_n = n^{\log_2 2} + \sum_{i=0}^{k-1} 2^i (\frac{n}{2^i} - 1)$$

sử dụng công thức: $\sum_{i=0}^{n} q^i = \frac{1-q^{n+1}}{1-q} \quad (q \neq 1)$

$$a_n = n + k\,n - \sum_{i=0}^{k-1} 2^i = n + k\,n - (2^k - 1)$$ với $k = \log_2 n$

$$a_n = n + n\log_2 n - (2^{\log_2 n} - 1)$$ sử dụng $c^{\log_b^n} = n^{\log_b^c}$

$$a_n = n\log_2 n + 1$$

Vậy: $C(n) = \Theta(n\log(n))$

Độ phức tạp thực tế

- Đánh giá một cách chi tiết thuật toán
- Tất cả các câu lệnh đều được xem xét
- Đánh giá riêng rẽ các loại thao tác khác nhau
  - phép cộng/trừ số nguyên
  - phép nhân số nguyên
  - phép so sánh
  - thao tác đọc/ghi
  - truy cập bộ nhớ
  - …
- Mang lại các kết quả rất chi tiết
  - nhưng thường phức tạp
  - thường khó để so sánh (do đánh giá riêng rẽ)

Độ phức tạp thực tế

- Ví dụ (1)
  - Đánh giá độ phức tạp của thuật toán sắp xếp chèn (insertion sort)
  - Mỗi lệnh được gán chi phí (cost) về thời gian thực thi và số lần thực thi (times). Với mỗi j = 2,3,…n, kí hiệu t_j là số lần thực thi lệnh lặp While

insertion-sort(A)	cost	times
Begin		
For j from 2 to n	c_1	n
key = A[j]	c_2	n-1
i = j – 1	c_3	n-1
While (i > 0 and A[i] > key) do	c_4	$\sum_{j=2}^{n} t_j$
A[i+1] = A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
i = i – 1	c_6	$\sum_{j=2}^{n} (t_j - 1)$
EndWhile		
A[i+1] = key	c_7	n-1
EndFor		
End		

Độ phức tạp thực tế

- Ví dụ (2)
  - Tổng thời gian thực thi là

$$C(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

  - Độ phức tạp trong trường hợp tốt nhất: danh sách đã được sắp xếp đúng thứ tự, khi đó $t_j = 1$ với mọi j. Vậy:

$$C(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$
$$= n(c_1 + c_2 + c_3 + c_4 + c_7) - (c_2 + c_3 + c_4 + c_7)$$

Có thể viết C(n) = an + b, với a và b là các hằng số

Độ phức tạp trong trường hợp tốt nhất là hàm tuyến tính n

Độ phức tạp thực tế

- Ví dụ (2)
  - Độ phức tạp trong trường hợp xấu nhất: danh sách đã được sắp xếp theo thứ tự ngược lại, khi đó $t_j = j$ với mọi j.
  - Ta có:
  - Vậy: $\sum_{j=1}^{n} j = \frac{n(n+1)}{2}, \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1, \sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$

$$C(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - \left(c_2 + c_3 + c_4 + c_7\right)$$

Có thể viết $C(n) = an^2 + bn + c$, với a, b và c là các hằng số

Độ phức tạp trong trường hợp xấu nhất là hàm bình phương n

Độ phức tạp thực tế

- Ví dụ (3)
  - Độ phức tạp trong trường hợp trung bình
    - giả sử áp dụng thuật toán cho danh sách n phân tử được chọn một cách ngẫu nhiên.
    - Giá trị của t_j ? Phần tử A[j] được chèn vào vị trí nào trong danh sách A[1..j-1] ?
    - Trường hợp trung bình, một nữa số phần tử của A[1..j-1] lớn hơn A[j] và một nữa số phần tử của A[1..j-1] nhỏ hơn A[j]
    - Vậy $t_j = j/2$
  - Tương tự trường hợp xấu nhất, C(n) sẽ có dạng

    $C(n) = an^2 + bn + c$, với a, b và c là các hằng số
  - Độ phức tạp trong trường hợp trung bình là hàm bình phương n

Độ phức tạp thực tế

- Ví dụ (4)
  - Tuy nhiên, thông thường cái chúng ta quan tâm thực sự là giá trị của các hàm tiệm cận (thường là O và Θ)
  - Chỉ có số hạng quan trọng của biểu thức độ phức tạp có ý nghĩa khi n rất lớn, bỏ qua các số hạng còn lại
  - Khi n lớn, chúng ta bỏ qua luôn hằng số của số hạng quan trọng
  - Vậy độ phức tạp của thuật toán sắp xếp chèn trong
    - trường hợp tốt nhất: $\Theta(n)$
    - trường hợp xấu nhất: $\Theta(n^2)$
    - trường hợp trung bình: $\Theta(n^2)$

Các lớp thuật toán theo độ phức tạp

- **Các thuật toán thường được so sánh bởi độ phức tạp trong trường hợp xấu nhất**, tức là giá trị của O(g(n))
- Các lớp thuật toán theo độ phức tạp
  - Hằng số: O(1), tất cả các lệnh thực thi đúng một lần không phụ thuộc kích thước dữ liệu
  - Dưới tuyến tính: O(log(n)), kích thước của bài toán được chia nhỏ bởi một hằng số ở mỗi bước lặp
  - Tuyến tính:
    - O(n), vòng lặp n lần, thân vòng lặp thực hiện công việc độc lập với n
    - O(nlog(n)), kích thước của bài toán được chia nhỏ bởi một hằng số ở mỗi bước lặp,và ở mỗi lần chia nhỏ một bước duyệt tuyến tính các dữ liệu được thực hiện
  - Đa thức: $O(n^k)$, các thuật toán được xem là chậm khi $k>3$
  - Hàm mũ: $O(k^n)$, $k \geq 2$, các thuật toán là không thể ứng dụng được
  - Các **thuật toán sử dụng được** thường có độ phức tạp $\leq O(n^3)$

Các lớp thuật toán theo độ phức tạp

- Minh hoạ độ phức tạp các lớp thuật toán

n	$\log_2(n)$	$n\log_2(n)$	n^2	n^3	2^n
1	0	0	1	1	2
10	3.32	33.2	100	10^3	1024
100	6.64	664	10^4	10^6	1.26^{30}
1000	9.965	9 965	10^6	10^9	∞
10^6	19.931	19 931 568	10^{12}	10^{18}	∞

Bài tập

- Bài 1
  - Giả sử có một danh sách n phần tử có giá trị khác nhau. Hãy
    1. Xây dựng thuật toán xác định phần tử nhỏ nhất và phần tử lớn nhất
    2. Đánh giá độ phức tạp của thuật toán bởi số phép so sánh
    3. Đề xuất thuật toán hiệu quả hơn
    4. Đánh giá độ phức tạp của thuật toán bởi số phép so sánh

Bài tập

- Bài 2
  - Đánh giá độ phức tạp của thuật toán sau

```
timkiemnhiphan (A, x, l, r)
// tìm x trong danh sách A[l,r]
begin
    if (l = r) then return (l)
    else
        m = (l + r)/2
        if (x ≤ A[m]) then return (timkiemnhiphan (A, x, l, m))
        else return (timkiemnhiphan (A, x, m+1, r))
        endif
    endif
end
```