CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Bài 01: Lý thuyết độ phức tạp thuật toán

Nguyễn Thị Tâm
nguyenthitam.hus@gmail.com

Ngày 7 tháng 2 năm 2025

Nội dung

Nội dung

Bài toán

Bài toán trong tin học là một vấn đề cần giải quyết thông qua việc sử dụng các phương pháp và kỹ thuật tin học, chẳng hạn như lập trình, thuật toán, cấu trúc dữ liệu, và các lý thuyết tính toán.

- Bài toán tìm ước chung lớn nhất của hai số nguyên dương
  - Cho: Hai số nguyên dương a, b
  - Tìm: UCLN u (là số lớn nhất mà cả a, b đều chia hết)
- Bài toán giải phương trình bậc hai một ẩn
  - Cho: Ba số a, b, c
  - Tìm: Phương trình có nghiệm hay không? Nếu có thì có bao nhiêu nghiệm, và giá trị của các nghiệm đó là bao nhiêu?
- Bài toán sắp xếp một dãy số theo thứ tự tăng dần
  - Cho: Dãy số $a = (a_1, a_2, ..., a_n)$
  - Tìm: Dãy số a với các phần tử được sắp xếp theo thứ tự tăng dần
- ...

Bài toán

- Bài toán được cấu tạo bởi hai thành phần cơ bản:
  - Đầu vào (*input*): cung cấp các dữ liệu đã có
  - Đầu ra (*output*): những yếu tố cần tìm
- Cho bài toán: cho *input* và *output*
- Giải bài toán: từ *input*, dùng các thao tác thích hợp để tìm được *output*

Khái niệm thuật toán (*Algorithm*)

Định nghĩa

Thuật toán (thuật giải/giải thuật) để giải một bài toán là một dãy hữu hạn các thao tác được sắp xếp theo một trình tự nhất định sao cho sau khi thực hiện dãy các thao tác đó, từ đầu vào của bài toán ta nhận được đầu ra.

Ví dụ:

- Thuật toán tương ứng với một công thức nấu ăn
- Thuật toán giải phương trình bậc hai $ax^2 + bx + c = 0$

Các đặc trưng của thuật toán

- Tính *xác định thứ tự*: các bước của thuật toán phải xác định rõ bước nào thực hiện trước, bước nào thực hiện sau
  - Mặc dù vậy, không nhất thiết yêu cầu phải thực hiện các dãy bước tuần tự bước 1, bước 2,. . . (ví dụ, các thuật toán song song)

Các đặc trưng của thuật toán

- Tính *xác định thứ tự*: các bước của thuật toán phải xác định rõ bước nào thực hiện trước, bước nào thực hiện sau
  - Mặc dù vậy, không nhất thiết yêu cầu phải thực hiện các dãy bước tuần tự bước 1, bước 2,. . . (ví dụ, các thuật toán song song)
- Tính *hiệu quả*: các bước của thuật toán phải *thực hiện được*
  - Ví dụ, thuật toán không thể chứa bước nào đó có dạng "*Tạo một dãy gồm tất cả các số nguyên dương*", vì dãy số nguyên dương là vô hạn, bước này không thực hiện được.

Các đặc trưng của thuật toán

- Tính *xác định thứ tự*: các bước của thuật toán phải xác định rõ bước nào thực hiện trước, bước nào thực hiện sau
  - Mặc dù vậy, không nhất thiết yêu cầu phải thực hiện các dãy bước tuần tự bước 1, bước 2,. . . (ví dụ, các thuật toán song song)
- Tính *hiệu quả*: các bước của thuật toán phải *thực hiện được*
  - Ví dụ, thuật toán không thể chứa bước nào đó có dạng "*Tạo một dãy gồm tất cả các số nguyên dương*", vì dãy số nguyên dương là vô hạn, bước này không thực hiện được.
- Tính *không nhập nhằng*
  - Thông tin ở mỗi trạng thái trong quá trình thực hiện thuật toán phải rõ ràng và đầy đủ để xác định chính xác các hành động cần thực hiện ở mỗi bước của thuật toán..

Các đặc trưng của thuật toán

- Tính *xác định thứ tự*: các bước của thuật toán phải xác định rõ bước nào thực hiện trước, bước nào thực hiện sau
  - Mặc dù vậy, không nhất thiết yêu cầu phải thực hiện các dãy bước tuần tự bước 1, bước 2,. . . (ví dụ, các thuật toán song song)
- Tính *hiệu quả*: các bước của thuật toán phải *thực hiện được*
  - Ví dụ, thuật toán không thể chứa bước nào đó có dạng "*Tạo một dãy gồm tất cả các số nguyên dương*", vì dãy số nguyên dương là vô hạn, bước này không thực hiện được.
- Tính *không nhập nhằng*
  - Thông tin ở mỗi trạng thái trong quá trình thực hiện thuật toán phải rõ ràng và đầy đủ để xác định chính xác các hành động cần thực hiện ở mỗi bước của thuật toán..
- Tính *hữu hạn*: thuật toán phải xác định một quá trình có tính *kết thúc*, nghĩa là thuật toán phải dừng sau một số hữu hạn bước.

Thuật toán và biểu diễn của thuật toán

Cần nhấn mạnh sự khác nhau giữa một thuật toán và biểu diễn của nó. Tương tự như giữa một cốt truyện và một cuốn sách.

- Cốt truyện : trừu tượng, mang tính khái niệm;
- Cuốn sách : là biểu diễn vật lí của cốt truyện. Khi truyện được dịch sang thứ tiếng khác, hoặc xuất bản dưới dạng khác thì chỉ phần biểu diễn là thay đổi, cốt truyện vẫn không đổi.

Thuật toán và biểu diễn của thuật toán

Cần nhấn mạnh sự khác nhau giữa một thuật toán và biểu diễn của nó. Tương tự như giữa một cốt truyện và một cuốn sách.

- Cốt truyện : trừu tượng, mang tính khái niệm;
- Cuốn sách : là biểu diễn vật lí của cốt truyện. Khi truyện được dịch sang thứ tiếng khác, hoặc xuất bản dưới dạng khác thì chỉ phần biểu diễn là thay đổi, cốt truyện vẫn không đổi.

Thuật toán có thể được biểu diễn dưới nhiều dạng khác nhau. Ví dụ, thuật toán chuyển từ nhiệt độ C sang nhiệt độ F :

- Biểu diễn bằng công thức đại số : $\boxed{F = \frac{9}{5}C + 32}$;
- Biểu diễn bằng chỉ dẫn : *Nhân nhiệt độ C với* $\frac{9}{5}$ *và cộng với 32.*

Các cách biểu diễn thuật toán thường dùng

- Ngôn ngữ tự nhiên
- Sơ đồ khối
- Mã giả
- Ngôn ngữ lập trình

Dùng ngôn ngữ tự nhiên

Ví dụ:

- Thuật toán Euclid tìm ước chung lớn nhất của hai số nguyên dương x và y được diễn đạt như sau:

  Chừng nào cả x và y khác 0, chia giá trị lớn hơn cho giá trị nhỏ hơn và thay x, y tương ứng bởi số chia và phần dư. Giá trị cuối cùng của x là ước chung lớn nhất.

Dùng ngôn ngữ tự nhiên

Ví dụ: Bài toán tìm giá trị lớn nhất của một dãy số gồm N phần tử

1. Định nghĩa bài toán
   - Input: Số nguyên dương n và dãy gồm n số nguyên $a_1,...,a_n$
   - Output: Giá trị lớn nhất của dãy số max
2. Thuật toán
   - Bước 1: $max := a_1; i := 2;$
   - Bước 2: Kiểm tra nếu $i > n$ thì trả lại giá trị max rồi kết thúc
   - Bước 3:
     - 3.1 Nếu $a_i > max$ thì $max := a_i$
     - 3.2 Tăng $i := i + 1$ rồi quay lại Bước 2

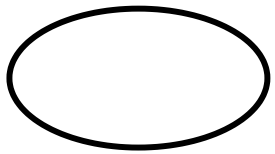
Dùng ngôn ngữ tự nhiên

Ví dụ: Kiểm tra số nguyên N có phải là số nguyên tố hay không?

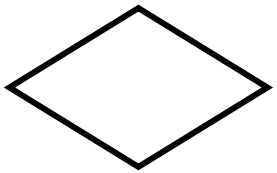
1. Định nghĩa bài toán
   - Input: Số nguyên n
   - Output: Giá trị Đúng (true) nếu n là số nguyên tố, giá trị Sai (false) nếu n không phải là số nguyên tố
2. Thuật toán
   - Bước 2: if n $<$ 2 then return false
   - Bước 3: for $i := 2 \rightarrow n - 1$
     - if $n\%i == 0$ then return false
   - Bước 4: return true

Dùng sơ đồ khối

- Vẽ sơ đồ biểu diễn thuật toán theo hệ thống kí hiệu trực quan được quy ước trước.



Bắt đầu, kết thúc



Điều kiện

Thao tác tính toán

Khối nhập xuất dữ liệu

Luồng xử lý

Dùng mã giả

- Mã giả (*pseudo-code*) là một hệ thống các kí hiệu, thuật ngữ dùng để biểu diễn thuật toán.
- Là sự trừu tượng hóa của ngôn ngữ lập trình, không phụ thuộc vào ngôn ngữ lập trình cụ thể nào.
- Được xây dựng dựa trên các cấu trúc ngữ nghĩa nhằm diễn tả quá trình phát triển của thuật toán.

Thuật toán và biểu diễn của thuật toán

Cần phân biệt 2 khái niệm *chương trình* và *quá trình*

- Mỗi chương trình (*program*) là biểu diễn của một thuật toán dùng cho máy tính.
- Mỗi quá trình (*process*) là hành động chạy một chương trình, cũng là chạy một thuật toán.

Câu hỏi/Bài tập

1. Giải thích vì sao các bước mô tả dưới đây không tạo thành một thuật toán:

   Bước 1. Lấy một đồng xu ra khỏi ví và đặt nó lên bàn.

   Bước 2. Trở lại bước 1.

Phát hiện thuật toán

Phát triển một thuật toán gồm 2 giai đoạn:

- Phát hiện thuật toán
- Biểu diễn thuật toán dưới dạng chương trình

Phương pháp phát hiện thuật toán? Vấn đề chính trong quá trình phát triển phần mềm.

Phát hiện thuật toán

Phát triển một thuật toán gồm 2 giai đoạn:

- Phát hiện thuật toán
- Biểu diễn thuật toán dưới dạng chương trình

Phương pháp phát hiện thuật toán? Vấn đề chính trong quá trình phát triển phần mềm.

Nhà toán học G. Polya, 1945, phương pháp giải quyết bài toán:

Bước 1. Hiểu bài toán.

Bước 2. Đặt ra kế hoạch giải quyết bài toán.

Bước 3. Thực hiện kế hoạch.

Bước 4. Đánh giá tính đúng của lời giải và khả năng dùng nó để giải các bài toán khác.

Phát hiện thuật toán

Trong ngữ cảnh phát triển phần mềm, các bước này trở thành

Bước 1. Hiểu bài toán.

Bước 2. Đưa ra ý tưởng về một thuật toán có khả năng giải quyết bài toán.

Bước 3. Trình bày rõ thuật toán và biểu diễn nó dưới dạng một chương trình.

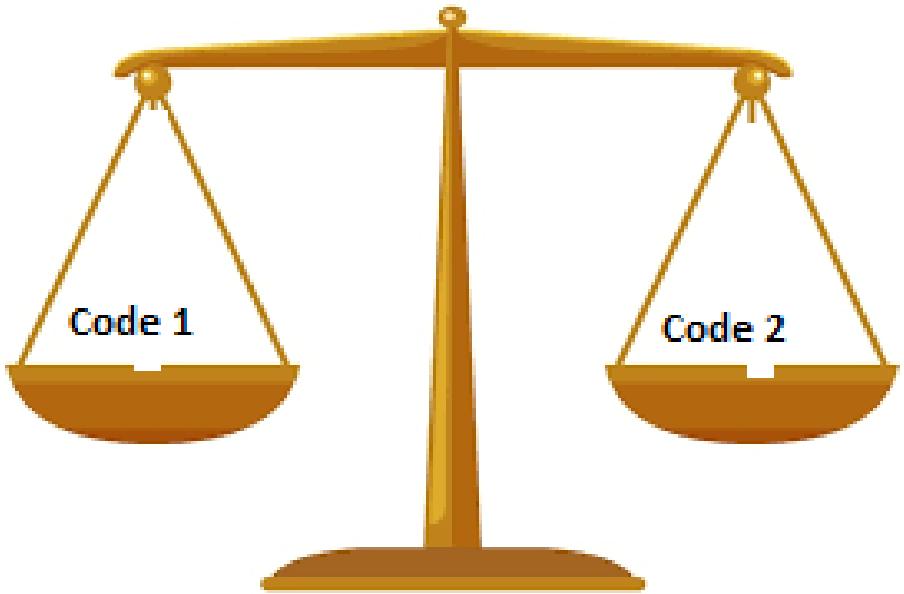
Bước 4. Đánh giá tính đúng của chương trình và khả năng dùng nó như là công cụ để giải các bài toán khác.

Chú ý:

- Cần hạn chế quá trình thử-và-sai trong quá trình xây dựng thuật toán
- Cần có đầy đủ dữ kiện (hoàn toàn hiểu bài toán) trước khi tìm được lời giải.

So sánh hai thuật toán

- Làm sao để so sánh hai thuật toán?



  - Thời gian thực thi
  - Bộ nhớ sử dụng
- Đo thời gian chạy thực tế
  - Phụ thuộc vào kích thước đầu vào cũng như hiệu năng của hệ thống chạy chương trình

Đánh giá độ phức tạp thuật toán

- Đánh giá độ phức tạp của thuật toán là đánh giá lượng tài nguyên các loại mà thuật toán cần sử dụng. Đặc biệt quan tâm đến tài nguyên *thời gian tính*
- Kích thước dữ liệu đầu vào (hay độ dài dữ liệu đầu vào)
  - Số lượng phần tử trong danh sách
  - Số đỉnh, số cạnh trong một đồ thị,...

Phép toán cơ bản

Là các phép toán có thể thực hiện được với thời gian bị chặn bởi một hằng số không phụ thuộc vào kích thước dữ liệu

- Gọi n là kích thước dữ liệu đưa vào thì thời gian thực hiện thuật toán có thể biểu diễn một cách tương đối như một hàm của n: $T(n)$.
- Độ phức tạp thuật toán $T(n)$ được tính bằng cách đếm số phép toán cơ bản thực hiện thuật toán đó.

Các loại thời gian tính

- Thời gian tính tốt nhất (*best-case complexity*): số lượng phép toán cơ bản tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào
- Thời gian tính tệ nhất (*worst-case complexity*): số lượng phép toán cơ bản nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào
- Thời gian tính trung bình (*average-case complexity*): số lượng phép toán cơ bản trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào.

Các loại thời gian tính

- Cần hiểu thế nào về đánh giá thuật toán (về thời gian)
  - Là đánh giá độ phức tạp về thời gian của thật toán trong trường hợp tồi nhất
  - Ước lượng lượng thời gian lớn nhất mà thuật toán cần để đưa ra kết quả
- Dùng đánh giá này thế nào cho đúng
  - VD. Có 2 thuật toán A và B cùng giải bài toán P với độ phức tạp về thời gian trong trường hợp tồi nhất lần lượt là $T_A(n) = 100n^2 + 20$ và $T_B(n) = n^3 + 2n^2 + n$.
  - Kết luận thuật toán A luôn nhanh hơn B?
  - Trong thực tế cần chọn thuật toán nào? A hay B?
- Trong trường hợp tổng quát, cần chọn thuật toán có độ phức tạp về thời gian nhỏ hơn

Nội dung

Kí hiệu Tiệm cận Big O, Big Theta - Θ, Big Ω

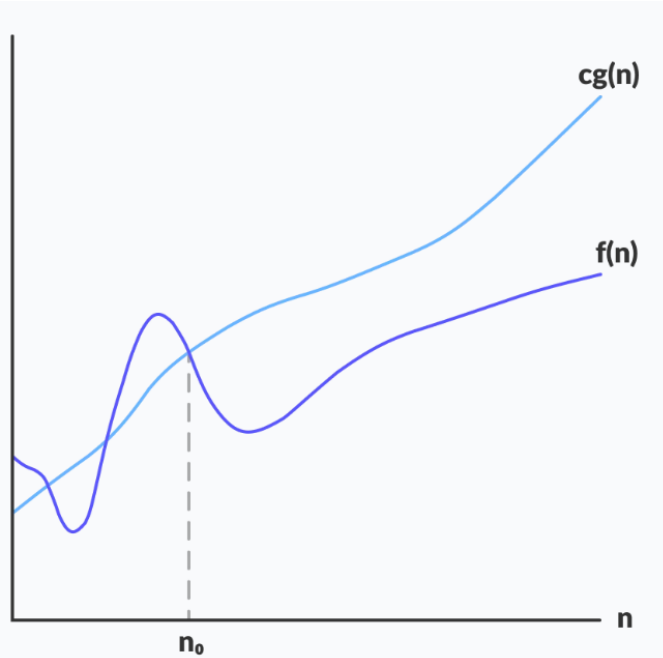
- O, Ω, Θ dùng để ký hiệu đại diện cho một tập các hàm số
- Dùng để đánh giá cận trên, cận dưới, cận chặt của thuật toán theo mức độ tăng của hàm theo kích thước đầu vào
- Mục đích là dùng để đánh giá độ phức tạp thuật toán của thuật toán thông qua các hàm đơn giản, quen thuộc.
  - Ví dụ hàm đa thức theo biến n, $nlog(n)$, n^2,...
  - Dễ dàng ước lượng, so sánh độ phức tạp của các thuật toán

Kí hiệu O lớn (Big-O) - O

- Đối với hàm $f(n)$ cho trước, ta kí hiệu $O(g(n))$ là tập các hàm

$$O(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \text{ sao cho } 0 \leq f(n) \leq c \cdot g(n)\ \forall n \geq n_0\}$$

- Ta nói rằng $g(n)$ là tiệm cận trên của $f(n)$ và viết $f(n) \in O(g(n))$ hoặc $f(n) = O(g(n))$



- **Mục tiêu là tìm hàm $g(n)$ nhỏ nhất thoả mãn $f(n) \leq c \cdot g(n)$**

Kí hiệu O lớn (Big-O) - O

Ví dụ

- $f(n) = n^2 + 1$

Kí hiệu O lớn (Big-O) - O

Ví dụ

- $f(n) = n^2 + 1$
  Ta có $n^2 + 1 \leq 2n^2, \forall n \geq 1$.
  Do đó, $n^2 + 1 = O(n^2)$ *với* $c = 2$ *và* $n_0 = 1$
- $f(n) = n^4 + 100n^2 + 50$

Kí hiệu O lớn (Big-O) - O

Ví dụ

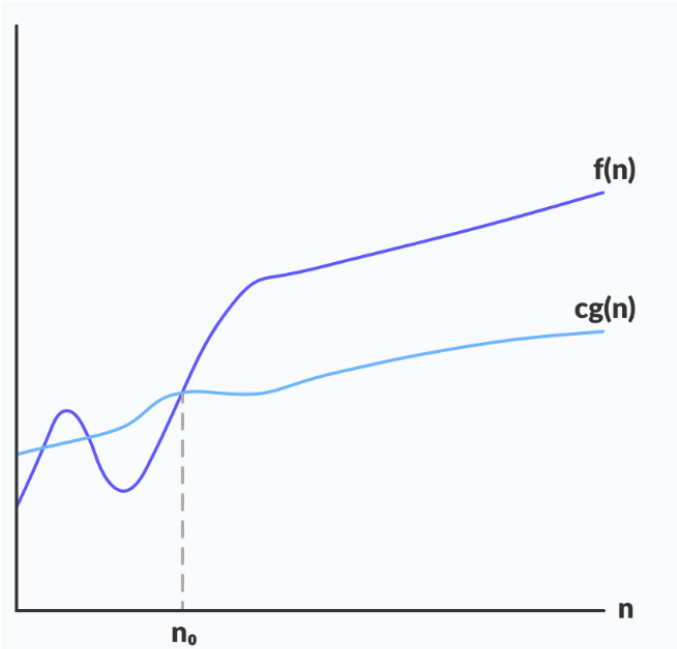
- $f(n) = n^2 + 1$
  Ta có $n^2 + 1 \leq 2n^2, \forall n \geq 1$.
  Do đó, $n^2 + 1 = O(n^2)$ *với* $c = 2$ *và* $n_0 = 1$
- $f(n) = n^4 + 100n^2 + 50$
  Ta có $n^4 + 100n^2 + 50 \leq 2n^4, \forall n \geq 11$.
  Do đó, $n^4 + 100n^2 + 50 = O(n^4)$ *với* $c = 2$ *và* $n_0 = 11$

Kí hiệu Omega lớn (Big-Ω) - Ω

- Đối với hàm $f(n)$ cho trước, ta kí hiệu $\Omega(g(n))$ là tập các hàm

$$\Omega(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \text{ sao cho } 0 \leq c \cdot g(n) \leq f(n)\ \forall n \geq n_0\}$$

- Ta nói rằng $g(n)$ là tiệm cận dưới của $f(n)$ và viết $f(n) \in \Omega(g(n))$ hoặc $f(n) = \Omega(g(n))$



- **Mục tiêu là tìm hàm $g(n)$ lớn nhất thoả mãn $c \cdot g(n) \leq f(n)$**

Kí hiệu Omega lớn (Big-Ω) - Ω

Ví dụ

- $f(n) = 5n^2$

Kí hiệu Omega lớn (Big-Ω) - Ω

Ví dụ

- $f(n) = 5n^2$
  Ta cần tìm c, n_0 *sao cho* $0 \leq cn^2 \leq 5n^2 \rightarrow c = 5, n_0 = 1$
  Do đó, $5n^2 = \Omega(n^2)$ *với* $c = 5$ *và* $n_0 = 1$
- $f(n) = 100n + 5 \neq \Omega(n^2)$

Kí hiệu Omega lớn (Big-Ω) - Ω

Ví dụ

- $f(n) = 5n^2$
  Ta cần tìm c, n_0 *sao cho* $0 \leq cn^2 \leq 5n^2 \rightarrow c = 5, n_0 = 1$
  Do đó, $5n^2 = \Omega(n^2)$ *với* $c = 5$ *và* $n_0 = 1$
- $f(n) = 100n + 5 \neq \Omega(n^2)$
  Giả sử, tồn tại c, n_0 *sao cho* $0 \leq cn^2 \leq 100n + 5$
  $100n + 5 \leq 100n + 5n(\forall n \geq 1) = 105n$
  $cn^2 \leq 105n \rightarrow n(cn - 105) \leq 0$
  Do n dương nên $cn - 105n \leq 0 \rightarrow n \leq \frac{105}{c}$
  Mâu thuẫn, n không thể nhỏ hơn một hằng số

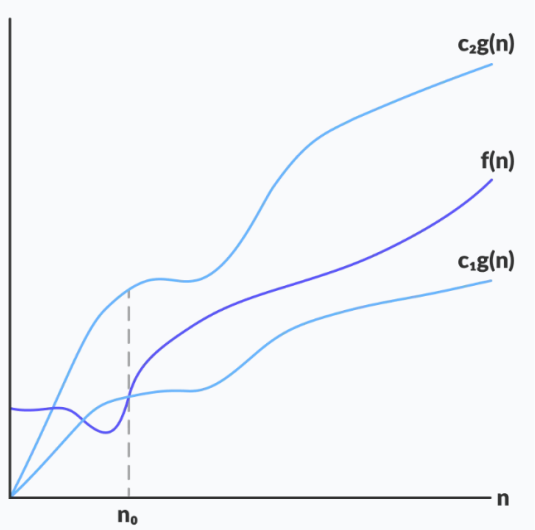
Kí hiệu Big Theta - Θ

Định nghĩa

Đối với hàm $g(n)$ cho trước, ta kí hiệu $\Theta(g(n))$ là tập các hàm

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 \text{ sao cho } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall n \geq n_0\}$$

- Ta nói rằng $g(n)$ là đánh giá tiệm cận đúng cho $f(n)$ và viết $f(n) \in \Theta(g(n))$ hoặc $f(n) = \Theta(g(n))$



Đối với hàm đa thức: để so sánh tốc độ tăng chỉ cần nhìn vào số hạng với số mũ cao nhất

Kí hiệu Big Theta - Θ

Ví dụ

- *Chứng minh rằng* $11n^2 - 4n = \Theta(n^2)$

Kí hiệu Big Theta - Θ

Ví dụ

- *Chứng minh rằng* $11n^2 - 4n = \Theta(n^2)$
  Ta có: $n^2 \leq 11n^2 - 4n \leq 12n^2$, *với mọi* $n \geq 1$, $c_1 = 1$ *và* $c_2 = 12$
- $f(n) = \frac{n^2}{2} - \frac{n}{2}$

Kí hiệu Big Theta - Θ

Ví dụ

- *Chứng minh rằng* $11n^2 - 4n = \Theta(n^2)$
  Ta có: $n^2 \leq 11n^2 - 4n \leq 12n^2$, *với mọi* $n \geq 1$, $c_1 = 1$ *và* $c_2 = 12$
- $f(n) = \frac{n^2}{2} - \frac{n}{2}$
  Ta có: $\frac{n^2}{5} \leq \frac{n^2}{2} - \frac{n}{2} \leq n^2, \forall n \geq 2$
  Tồn tại $c_1 = \frac{1}{5}, c_2 = 1$ *và* $n_0 = 2 \rightarrow \frac{n^2}{2} - \frac{n}{2} = \Theta(n^2)$

Sử dụng kí hiệu tiệm cận Θ, Ω và O

Đối với hai hàm $g(n)$ và $f(n)$ bất kỳ,

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

khi và chỉ khi

$$f(n) = O(g(n)) \text{và } f(n) = \Omega(g(n))$$

Cách nói về thời gian tính

- Thời gian tính là $O(g(n))$ hiểu là: Đánh giá tình huống tồi nhất là $O(g(n))$. Nghĩa là, thời gian tính trong tính huống tồi nhất được xác định bởi một hàm $f(n) \in O(g(n))$
- Thời gian tính là $\Omega((f(n))$ hiểu là: Đánh giá trong tình huống tốt nhất là $\Omega((f(n))$. Nghĩa là, thời gian tính trong tình huống tốt nhất được xác định bởi một hàm $f(n) \in \Omega(g(n))$

Nội dung

Quy tắc bỏ hằng số

- Nếu đoạn chương trình P có thời gian thực hiện $T(n) = O(c_1 g(n))$ với c_1 là một hằng số dương thì có thể coi đoạn chương trình đó có độ phức tạp tính toán là $O(g(n))$
- Quy tắc này cũng đúng với các kí pháp Ω và Θ

Quy tắc lấy max

- Nếu đoạn chương trình P có thời gian thực hiện $T(n) = O(g_1(n) + g_2(n))$ thì có thể coi đoạn chương trình có độ phức tạp tính toán bằng $O(max(g_1(n), g_2(n)))$
- Quy tắc này cũng đúng với các kí pháp Ω và Θ

Quy tắc cộng

- Nếu đoạn chương trình P_1 có thời gian thực hiện $T_1(n) = O(g_1(n))$, đoạn chương trình P_2 có thời gian thực hiện là $T_2(n) = O(g_2(n))$ thì thời gian thực hiện đoạn chương trình P_1 rồi đến P_2 sẽ là

$$T_1(n) + T_2(n) = O(g_1(n) + g_2(n))$$

- Quy tắc này cũng đúng với các kí pháp Ω và Θ

Quy tắc nhân

- Nếu đoạn chương trình P có thời gian thực hiện $T(n) = O(g_1(n))$. Khi đó, nếu thực hiện $k(n)$ lần đoạn chương trình P với $k(n) = O(g_2(n))$ thì độ phức tạp tính toán sẽ là $O(g_1(n)g_2(n))$
- Quy tắc này cũng đúng với các kí pháp Ω và Θ

Đánh giá độ phức tạp của chương trình

Một số công thức tổng chuỗi

- $S_n = 1 + 2 + 3 + ... + n = \frac{n \times (n+1)}{2}$
- $S_n = 1^2 + 2^2 + 3^2 + ... + n^2 = \frac{n \times (n+1)(2n+1)}{6}$
- $S_n = 1^3 + 2^3 + 3^3 + ... + n^3 = \frac{n^2 \times (n+1)^2}{4}$
- $S_n = 1^4 + 2^4 + 3^4 + ... + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$

Đánh giá độ phức tạp của chương trình

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        count++;
```

Đánh giá độ phức tạp của chương trình

```
// Lặp n lần
for (i = 0; i < n; i++)
    // Lặp n lần
    for (j = 0; j < n; j++)
        count++;
```

Đánh giá độ phức tạp của chương trình

```
for (i = n/2; i <=n; i++)
    for (j = 1; j + n/2 <=n; j++)
        for (k = 1; k <=n; k = k*2)
            count++;
```

Đánh giá độ phức tạp của chương trình

```
// Lặp n/2 lần
for (i = n/2; i <=n; i++)
    // Lặp n/2 lần
    for (j = 1; j + n/2 <=n; j++)
        // Lặp logn lần
        for (k = 1; k <=n; k = k*2)
            count++;
```

Đánh giá độ phức tạp của chương trình

```
x = x + 1
for (i = 1; i <= n; i++)
    m = m + 2

for (j = 1; j <=n; j++)
        for (k = 1; k<=n; k++)
            count++;
```

Đánh giá độ phức tạp của chương trình

```
x = x + 1 // Hằng số
// Lặp n lần
for (i = 1; i <= n; i++)
    m = m + 2; // Hằng số

// Lặp n lần
for (j = 1; j <=n; j++)
        for (k = 1; k<=n; k++) // Lặp n lần
            count++; // Hằng số
```

Đánh giá độ phức tạp của chương trình

```
for (i = n/2; i <=n; i++)
    for (j = 1; j <= n; j = j*2)
        for (k = 1; k <=n; k = k*2)
            count++;
```

Đánh giá độ phức tạp của chương trình

```
//Lặp n/2 lần
for (i = n/2; i <=n; i++)
    //Lặp logn lần
    for (j = 1; j <= n; j = j*2)
        // Lặp logn lần
        for (k = 1; k <=n; k = k*2)
            count++;
```

Một số tính chất của hàm mũ, loga và giai thừa

- Hàm mũ

$$a = b^{log_b a}, log_b a = 1/log_a b$$

- Trong kí hiệu tiệm cận cơ số của log là không quan trọng

$$O(lgn) = O(lnn) = O(logn)$$

- Công thức Stirling

$$n! = \sqrt{2\pi n}(\frac{n}{e})^n(1 + \Theta(\frac{1}{n}))$$

- Giai thừa và hàm mũ:

$$2^n < n! < n^n \text{ với } n > 5$$

$$logn! = \Theta(nlogn)$$

Tốc độ tăng của các hàm

Time Complexity	Name
1	Constant
$logn$	Logarithmic
n	Linear
$nlogn$	Linear Logarithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential
$n!$	Factorial

Định lý chính/Định lý thợ

Định lý chính/Định lý thợ (*master theorem*)

- Dùng để giải các công thức đệ quy dạng $T(n) = aT(\frac{n}{b}) + \Theta(n^k log^p n)$, với $a \geq 1, b > 1, k \geq 0$
- Bài toán ban đầu được chia thành a bài toán con có kích thước mỗi bài toán con là $\frac{n}{b}$ chi phí để tổng hợp các bài toán con là $\Theta(n^k log^p n)$
  - Nếu $a > b^k$, $T(n) = \Theta(n^{log_b a})$
  - Nếu $a = b^k$
    - Nếu $p > -1$, $T(n) = \Theta(n^{log_b a} log^{p+1} n)$
    - Nếu $p = -1$, $T(n) = \Theta(n^{log_b a} loglogn)$
    - Nếu $p < -1$, $T(n) = \Theta(n^{log_b a})$
  - Nếu $a < b^k$
    - Nếu $p \geq 0$, $T(n) = \Theta(n^k log^p n)$
    - Nếu $p < 0$, $T(n) = O(n^k)$