TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Toán rời rạc

Nguyễn Khánh Phương

Bộ môn Khoa học máy tính
E-mail: phuongnk@soict.hust.edu.vn

Phần thứ nhất

LÝ THUYẾT TỔ HỢP

Combinatorial Theory

Nguyễn Khánh Phương

Bộ môn Khoa học Máy tính,
Viện CNTT và Truyền thông,
Đại học Bách khoa Hà nội,
E-mail: phuongnk@soict.hust.edu.vn

Nội dung phần 1

Chương 0. Tập hợp

Chương 1. Bài toán đếm

Chương 2. Bài toán tồn tại

Chương 3. Bài toán liệt kê tổ hợp

Chương 4. Bài toán tối ưu tổ hợp

Nội dung chi tiết

1. **Giới thiệu bài toán**
2. Thuật toán và độ phức tạp
3. Phương pháp sinh
4. Thuật toán quay lui

Giới thiệu bài toán

- Bài toán đưa ra danh sách tất cả cấu hình tổ hợp thoả mãn một số tính chất cho trước được gọi là ***bài toán liệt kê tổ hợp***.
- Do số lượng cấu hình tổ hợp cần liệt kê thường là rất lớn ngay cả khi kích thước cấu hình chưa lớn:
  - Số hoán vị của n phần tử là $n!$
  - Số tập con m phần tử của n phần tử là $n!/(m!(n-m)!)$

Do đó cần có quan niệm thế nào là giải bài toán liệt kê tổ hợp

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Giới thiệu bài toán

- Bài toán liệt kê tổ hợp là giải được nếu như ta có thể *x*ác định một ***thuật toán*** để theo đó có thể lần lượt xây dựng được tất cả các cấu hình cần quan tâm.
- Một thuật toán liệt kê phải đảm bảo 2 yêu cầu cơ bản:
  - Không được lặp lại một cấu hình,
  - Không được bỏ sót một cấu hình.

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

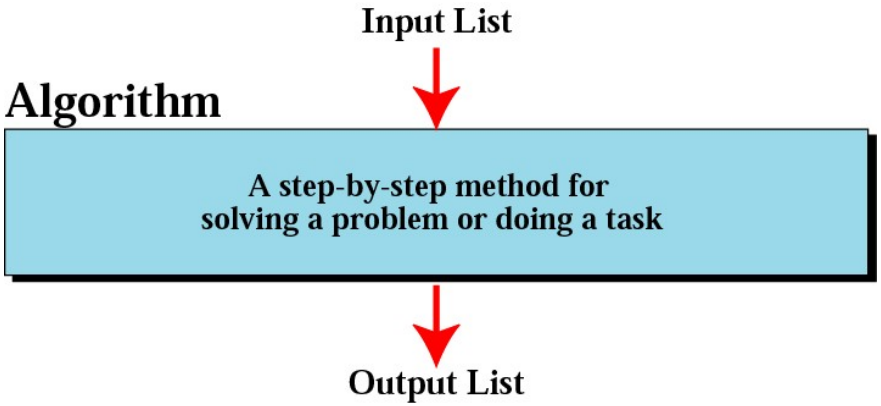
Nội dung chi tiết

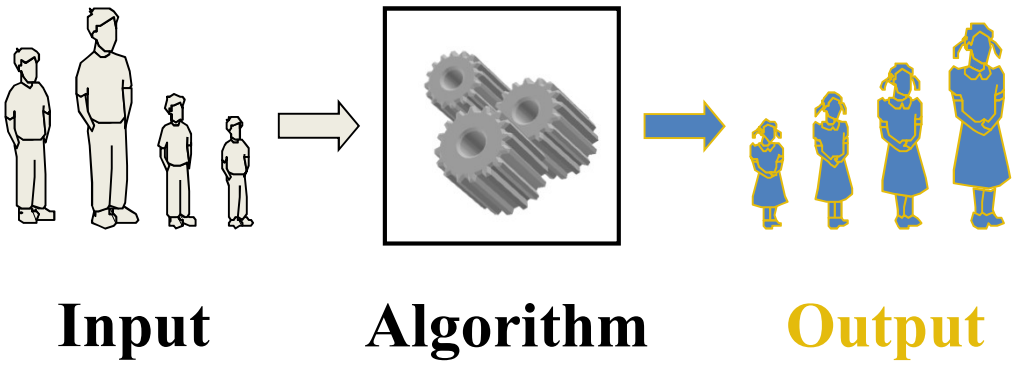
1. Giới thiệu bài toán
2. **Thuật toán và độ phức tạp**
3. Phương pháp sinh
4. Thuật toán quay lui

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Thuật toán (Algorithm)

- Thuật ngữ *algorithm* xuất phát từ tên của nhà toán học người Ba Tư: Abu Ja'far Mohammed ibn-i Musa al Khowarizmi.
- Trong ngành khoa học máy tính, thuật ngữ "thuật toán" được dùng để chỉ một phương pháp bao gồm **một chuỗi các câu lệnh** mà máy tính có thể sử dụng để giải quyết một bài toán.
- **Định nghĩa.** Ta hiểu thuật toán giải bài toán đặt ra là một thủ tục xác định bao gồm một dãy hữu hạn các bước cần thực hiện để **thu được đầu ra (output)** cho **một đầu vào cho trước (input)** của bài toán.





- Ví dụ: Bài toán tìm số nguyên lớn nhất trong một dãy các số nguyên dương cho trước
  - Đầu vào (Input) : dãy gồm n số nguyên dương $a_1, a_2, \ldots, a_N$
  - Đầu ra (Output): số có giá trị lớn nhất của dãy đã cho
  - Ví dụ: Input 12 8 13 9 11 ➔ Output: 12
  - Yêu cầu: thiết kế thuật toán giải bài toán trên

Thuật toán

- Thuật toán có các đặc trưng sau đây:
  - **Đầu vào (Input):** Thuật toán nhận dữ liệu vào từ một tập nào đó.
  - **Đầu ra (Output):** Với mỗi tập các dữ liệu đầu vào, thuật toán đưa ra các dữ liệu tương ứng với lời giải của bài toán.
  - **Chính xác (Precision):** Các bước của thuật toán được mô tả chính xác.
  - **Hữu hạn (Finiteness):** Thuật toán cần phải đưa được đầu ra sau một số hữu hạn (có thể rất lớn) bước với mọi đầu vào.
  - **Đơn trị (Uniqueness):** Các kết quả trung gian của từng bước thực hiện thuật toán được xác định một cách đơn trị và chỉ phụ thuộc vào đầu vào và các kết quả của các bước trước.
  - **Tổng quát (Generality):** Thuật toán có thể áp dụng để giải mọi bài toán có dạng đã cho.

Độ phức tạp của thuật toán

Cho 2 thuật toán cùng giải một bài toán cho trước, làm thế nào để xác định thuật toán nào tốt hơn?

- Khi nói đến hiệu quả của một thuật toán, ta quan tâm đến chi phí cần dùng để thực hiện nó:
  1) Dễ hiểu, dễ cài đặt, dễ sửa đổi?
  2) Thời gian sử dụng CPU? **THỜI GIAN**
  3) Tài nguyên bộ nhớ? **BỘ NHỚ**

Trong giáo trình này ta đặc biệt quan tâm đến đánh giá thời gian cần thiết để thực hiện thuật toán mà ta sẽ gọi là *thời gian tính* của thuật toán.

Làm thế nào để đo được thời gian tính

- Mọi thuật toán đều thực hiện chuyển đổi đầu vào thành đầu ra:

Rõ ràng: Thời gian tính của thuật toán phụ thuộc vào dữ liệu vào (kích thước tăng, thì thời gian tăng).

Ví dụ: Việc nhân hai số nguyên có 3 chữ số đòi hỏi thời gian khác hẳn so với việc nhân hai số nguyên có $3*10^9$ chữ số!

- **Định nghĩa.** ***Ta gọi kích thước dữ liệu đầu vào (hay độ dài dữ liệu vào) là số bít cần thiết để biểu diễn nó.***
- Vậy ta sẽ tìm cách đánh giá thời gian tính của thuật toán bởi một hàm của độ dài dữ liệu đầu vào.
- Tuy nhiên, trong một số trường hợp, kích thước dữ liệu đầu vào là như nhau, nhưng thời gian tính lại rất khác nhau
  - Ví dụ: Để tìm số nguyên tố đầu tiên có trong dãy: ta duyệt dãy từ trái sang phải

    Dãy 1: 3 9 8 12 15 20 (thuật toán dừng ngay khi xét phần tử đầu tiên)

    Dãy 2: 9 8 3 12 15 20 (thuật toán dừng khi xét phần tử thứ ba)

    Dãy 3: 9 8 12 15 20 3 (thuật toán dừng khi xét phần tử cuối cùng)

    ➔ 3 loại thời gian tính

Các loại thời gian tính

Thời gian tính tốt nhất (Best-case):

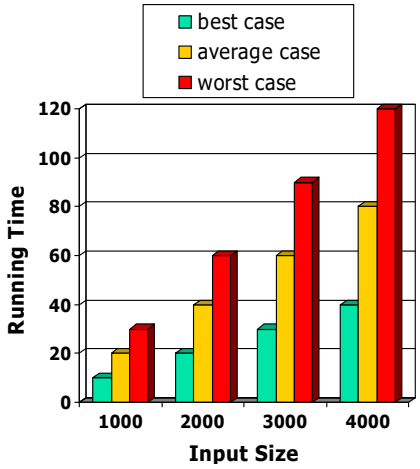
- T(n) : Thời gian tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước n.
- Dùng với các thuật toán chậm chạy nhanh với một vài đầu vào.

Thời gian tính trung bình (Average-case):

- T(n) : Thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích thước n.
- Rất hữu ích, nhưng khó xác định

Thời gian tính tồi nhất (Worst-case):

- T(n) : Thời gian nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích thước n. Thời gian như vậy sẽ được gọi là ***thời gian tính tồi nhất*** của thuật toán với đầu vào kích thước n.
- Dễ xác định



Có hai cách để đánh giá thời gian tính:

- Từ thời gian chạy thực nghiệm
- Lý thuyết: khái niệm xấp xỉ tiệm cận

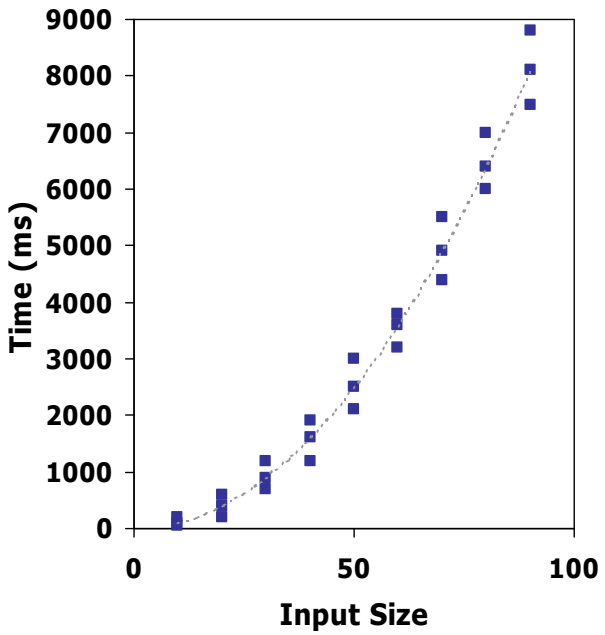
Phân tích thời gian tính của thuật toán bằng thời gian chạy thực nghiệm

Ta có thể đánh giá thuật toán bằng phương pháp thực nghiệm thông qua việc cài đặt thuật toán, rồi chọn các bộ dữ liệu đầu vào thử nghiệm:

- Viết chương trình thực hiện thuật toán
- Chạy chương trình với các dữ liệu đầu vào kích thước khác nhau
- Sử dụng hàm `clock( )` để đo thời gian chạy chương trình

```
clock_t startTime = clock();
doSomeOperation();
clock_t endTime = clock();
clock_t clockTicksTaken = endTime - startTime;
double timeInSeconds = clockTicksTaken / (double) CLOCKS_PER_SEC;
```

- Vẽ đồ thị kết quả



Nhược điểm của việc đánh giá thời gian tính của thuật toán dựa vào thời gian chạy thực nghiệm chương trình

- Bắc buộc phải cài đặt chương trình thì mới có thể đánh giá được thời gian tính của thuật toán. Do phải cài đặt bằng một ngôn ngữ lập trình cụ thể nên thuật toán sẽ chịu sự hạn chế của ngữ lập trình này. Đồng thời, hiệu quả của thuật toán sẽ bị ảnh hưởng bởi trình độ của người cài đặt.
- Kết quả thu được sẽ không bao gồm thời gian chạy của những dữ liệu đầu vào không được chạy thực nghiệm. Do vậy, cần chọn được các bộ dữ liệu thử đặc trưng cho tất cả tập các dữ liệu vào của thuật toán ➔ rất khó khăn và tốn nhiều chi phí.
- Để so sánh thời gian tính của hai thuật toán, cần phải chạy thực nghiệm hai thuật toán trên cùng một máy, và sử dụng cùng một phần mềm.

➔ Do đó người ta đã tìm kiếm những phương pháp đánh giá thuật toán hình thức hơn, ít phụ thuộc môi trường cũng như phần cứng hơn. Một phương pháp như vậy là phương pháp đánh giá thuật toán theo hướng xấp xỉ tiệm cận

Phân tích lý thuyết thời gian tính của thuật toán

- Sử dụng giả ngôn ngữ (pseudo-code) để mô tả thuật toán, thay vì tiến hành cài đặt thực sự
- Phân tích thời gian tính của thuật toán như là hàm của kích thước dữ liệu đầu vào, ***n***
- Phân tích tất cả các trường hợp có thể có của dữ liệu đầu vào
- Cho phép ta có thể đánh giá thời gian tính của thuật toán không bị phụ thuộc vào phần cứng/phần mềm chạy chương trình (Thay đổi phần cứng/phần mềm chỉ làm thay đổi thời gian chạy một lượng hằng số, chứ không làm thay đổi tốc độ tăng của thời gian tính)

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Kí hiệu tiệm cận (Asymptotic notation)

Θ, Ω, O, **o**, ω

» Những kí hiệu này:

- Được sử dụng để mô tả thời gian tính của thuật toán, mô tả tốc độ tăng của thời gian chạy phụ thuộc vào kích thước dữ liệu đầu vào.
- Được xác định đối với các hàm nhận giá trị nguyên không âm
- Dùng để so sánh tốc độ tăng của 2 hàm

  Ví dụ: $f(n) = \Theta(n^2)$: mô tả tốc độ tăng của hàm $f(n)$ và n^2.

» Thay vì cố gắng tìm ra một công thức phức tạp để tính chính xác thời gian tính của thuật toán, ta nói thời gian tính cỡ $\Theta(n^2)$ [đọc là theta n^2]: tức là, thời gian tính tỉ lệ thuận với n^2 cộng thêm các đa thức bậc thấp hơn

Θ – Kí hiệu Theta

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

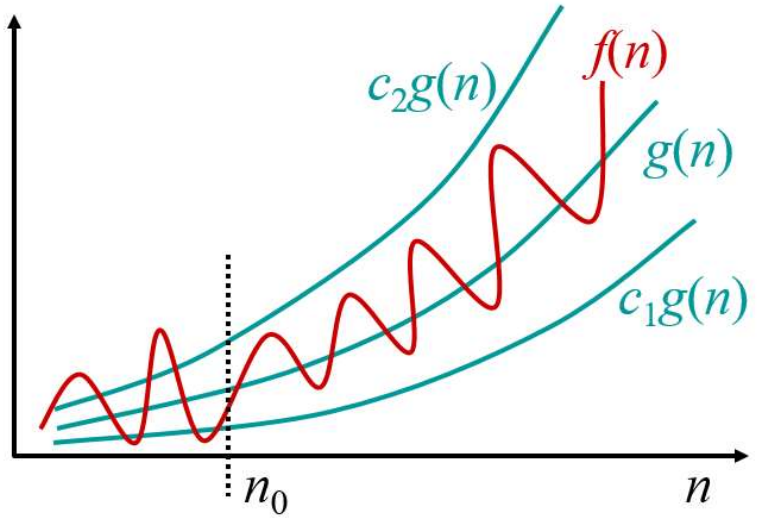
$\Theta(g(n)) = \{f(n)$: tồn tại các hằng số c_1, c_2 và n_0 sao cho

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$, với mọi $n \ge n_0 \}$

(tập tất cả các hàm có cùng tốc độ tăng với hàm $g(n)$).

Ta nói $g(n)$ là đánh giá tiệm cận đúng cho $f(n)$ và viết $f(n) = \boldsymbol{\Theta}(g(n))$

- Khi ta nói một hàm là theta của hàm khác, nghĩa là không có hàm nào đạt tới giá trị vô cùng nhanh hơn



$f(n) = \Theta(g(n)) \iff \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, \; c_1 g(n) \leq f(n) \leq c_2 g(n)$

Ví dụ 1: Chứng minh rằng $10n^2 - 3n = \mathbf{\Theta}(n^2)$

- Ta cần chỉ ra với những giá trị nào n_0, c_1, c_2 thì bất đẳng thức trong định nghĩa của kí hiệu theta là đúng:

$$c_1 n^2 \leq f(n) = 10n^2 - 3n \leq c_2 n^2 \; \forall n \geq n_0$$

- Gợi ý: lấy c_1 nhỏ hơn hệ số của số hạng với số mũ cao nhất (=10), và c_2 lấy lớn hơn hệ số này.

➔ Chẳng hạn, chọn: $c_1 = 1$, $c_2 = 11$, $n_0 = 1$ thì ta có

$$n^2 \leq 10n^2 - 3n \leq 11n^2, \; với \; n \geq 1.$$

➔ $\forall n \geq 1$: $10n^2 - 3n = \mathbf{\Theta}(n^2)$

Tổng quát: Với các hàm đa thức: để so sánh tốc độ tăng, ta cần nhìn vào số hạng có số mũ cao nhất

$f(n) = \Theta(g(n))$ $\Longleftrightarrow$ $\exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, \; c_1 g(n) \leq f(n) \leq c_2 g(n)$

Ví dụ 2: Chứng minh rằng $f(n) = \frac{1}{2}n^2 - 3n = \boldsymbol{\Theta}(n^2)$

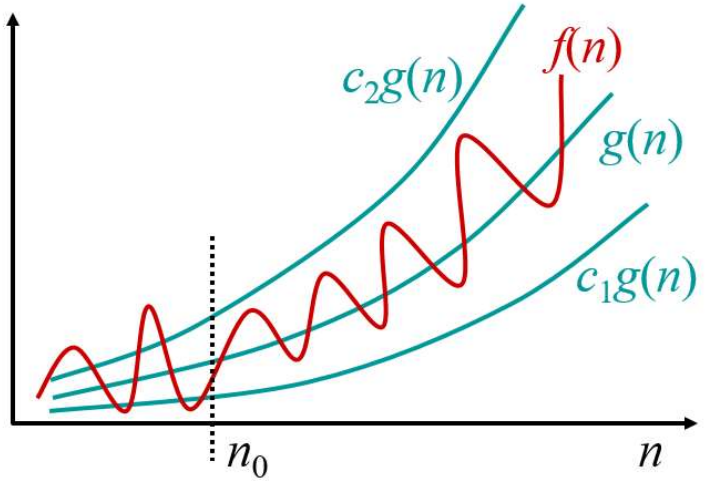
Ta cần tìm n_0, c_1 và c_2 sao cho

$$c_1 n^2 \leq f(n) = \frac{1}{2}n^2 - 3n \leq c_2 n^2 \; \forall n \geq n_0$$

Chọn $c_1 = ¼$, $c_2 = 1$, và $n_0 = 7$ ta có:

$$\frac{1}{4}n^2 \leq f(n) = \frac{1}{2}n^2 - 3n \leq n^2 \quad \forall\, n \geq 7$$

➔$\forall\, n \geq 7$: $\frac{1}{2}n^2 - 3n = \boldsymbol{\Theta}(n^2)$

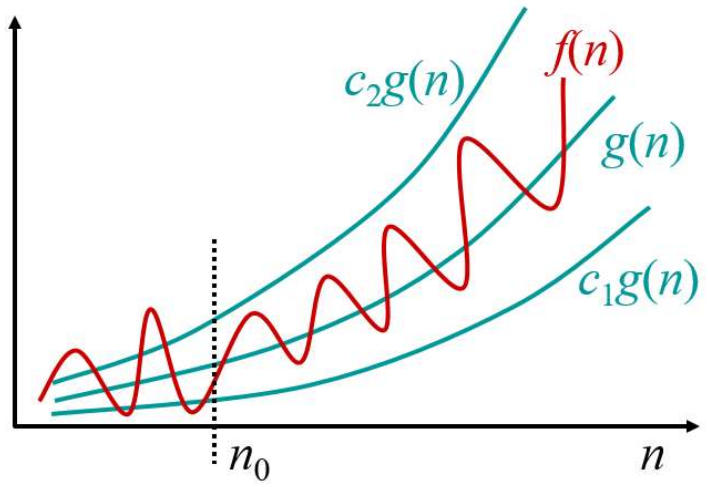


$f(n) = \Theta(g(n))$ $\Longleftrightarrow$ $\exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, \; c_1 g(n) \leq f(n) \leq c_2 g(n)$

Ví dụ 3: Chứng minh rằng $f(n) = 23n^3 - 10\, n^2 \log_2 n + 7n + 6 = \boldsymbol{\Theta}(n^3)$

Ta phải tìm n_0, c_1 và c_2 sao cho

$$c_1 n^3 \leq f(n) = 23n^3 - 10\, n^2 \log_2 n + 7n + 6 \leq c_2 n^3 \text{ với } \forall n \geq n_0$$



$f(n) = \Theta(g(n))$ $\Longleftrightarrow$ $\exists c_1, c_2, n_0 > 0 \ : \ \forall n \geq n_0, \ c_1 g(n) \leq f(n) \leq c_2 g(n)$

Ví dụ 3: Chứng minh rằng $f(n) = 23n^3 - 10\, n^2 \log_2 n + 7n + 6 = \Theta(n^3)$

$f(n) = 23n^3 - 10\, n^2 \log_2 n + 7n + 6$

➔ $f(n) = [23 - (10 \log_2 n)/n + 7/n^2 + 6/n^3]n^3$

Khi đó:

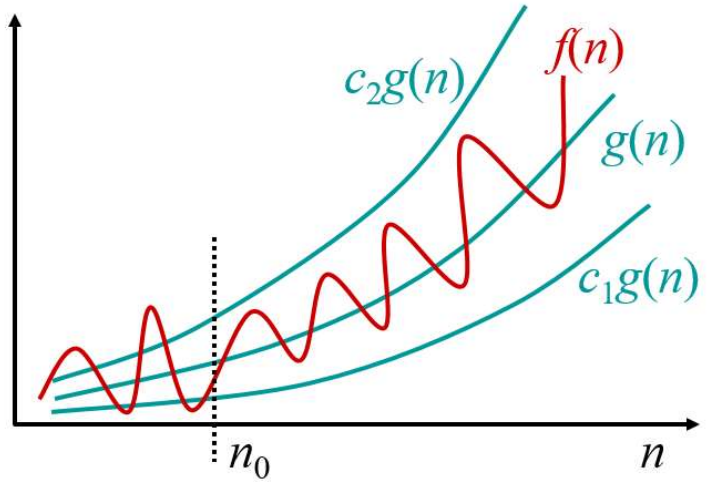
- $\forall n \geq 10 : \ f(n) \leq (23 + 0 + 7/100 + 6/1000)n^3$
  $= (23 + 0 + 0.07 + 0.006)n^3$
  $= 23.076\, n^3 < 24\, n^3$
- $\forall n \geq 10 : \ f(n) \geq (23 - \log_2 10 + 0 + 0)n^3 > (23 - \log_2 16)n^3 = 19n^3$

➔ Ta có:

$\forall n \geq 10: \ 19\, n^3 \leq f(n) \leq 24\, n^3$

($n_0 = 10, \ c_1 = 19, \ c_2 = 24, \ g(n) = n^3$)

Do đó: $f(n) = \Theta(n^3)$



O – Kí hiệu O lớn (big Oh)

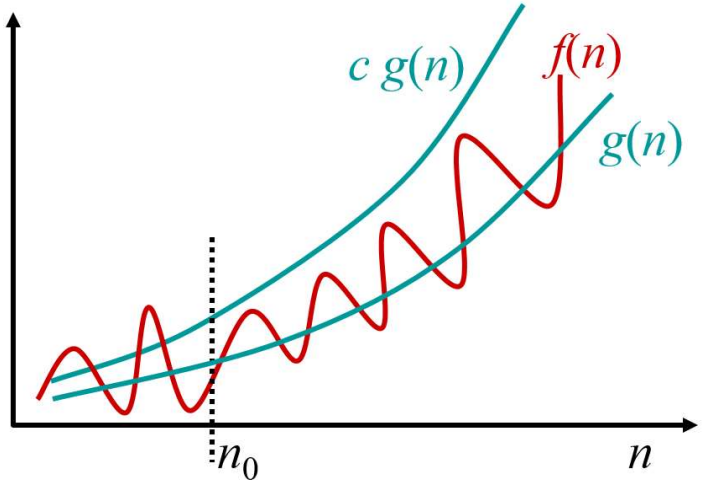
Đối với hàm $g(n)$ cho trước, ta ký hiệu $O(g(n))$ là tập các hàm

$O(g(n)) = \{f(n)$: tồn tại các hằng số dương c và n_0 sao cho:

$$f(n) \le cg(n) \text{ với mọi } n \ge n_0 \}$$

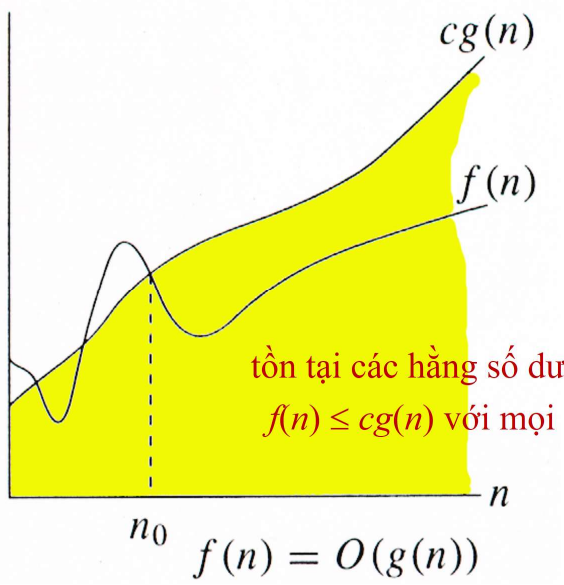
(tập tất cả các hàm có **tốc độ tăng nhỏ hơn hoặc bằng** tốc độ tăng của $g(n)$).

- Ta nói: $g(n)$ là cận trên tiệm cận của $f(n)$, và viết $f(n) = O(g(n))$.
- $f(n) = O(g(n))$ tức là tồn tại hằng số c sao cho $f(n)$ luôn $\le cg(n)$ với mọi giá trị n đủ lớn.
- $O(g(n))$ là tập các hàm đạt tới giá trị vô cùng không nhanh hơn $g(n)$.



Ví dụ 1

Chứng minh: $f(n) = n^2 + 2n + 1$ là $O(n^2)$



tồn tại các hằng số dương c và n_0 sao cho:
$f(n) \le cg(n)$ với mọi $n \ge n_0$ }

Cần chỉ ra với giá trị nào của các hằng số n_0 và c thì bất đẳng thức sau đây là đúng với $n \ge n_0$:

$$n^2 + 2n + 1 \le cn^2$$

Ta có:

$$2n \le 2n^2 \text{ khi } n \ge 1$$

và $1 \le n^2$ khi $n \ge 1$

Vì vậy: $n^2 + 2n + 1 \le 4n^2$ với mọi $n \ge 1$

Như vậy hằng số $c = 4$ và $n_0 = 1$

Rõ ràng: Nếu $f(n)$ là $O(n^2)$ thì nó cũng là $O(n^k)$ với $k > 2$.

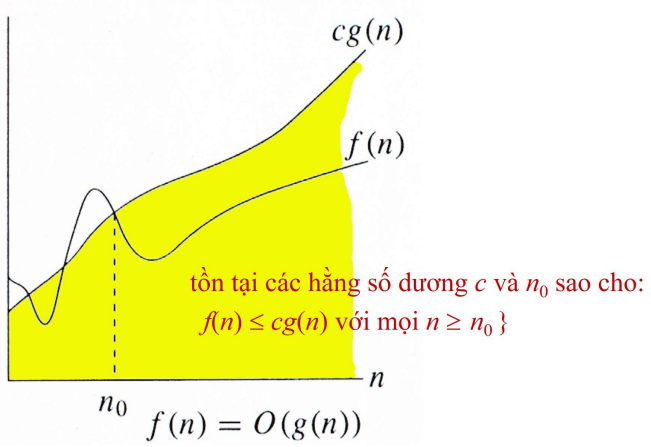
NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Chú ý

- Các giá trị của các hằng số dương n_0 và c **không phải là duy nhất** trong chứng minh công thức tiệm cận
- Ví dụ: Chứng minh rằng $100n + 5 = O(n^2)$
  - $100n + 5 \leq 100n + n = 101n \leq 101n^2 \quad \forall\, n \geq 5$

    $n_0 = 5$ and c = 101 là các hằng số cần tìm
  - $100n + 5 \leq 100n + 5n = 105n \leq 105n^2 \quad \forall\, n \geq 1$

    $n_0 = 1$ and $c = 105$ cũng là các hằng số cần tìm



Chỉ cần tìm các hằng số dương c và n_0 **nào đó** thỏa mãn bất đẳng thức trong định nghĩa công thức tiệm cận

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Ví dụ 2: Ký hiệu O lớn

Chứng minh: $f(n) = n^2 + 2n + 1 \notin O(n)$.

Phản chứng. Giả sử trái lại, khi đó phải tìm được hằng số c và số n_0 để cho:

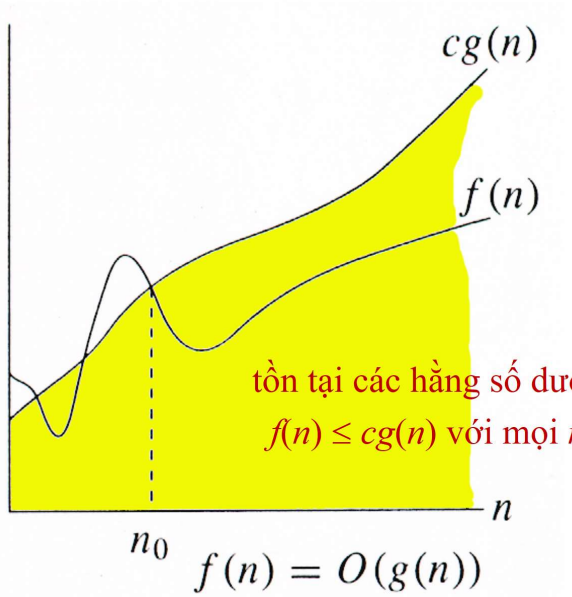
$$n^2 + 2n + 1 \leq c*n \text{ khi } n \geq n_0$$

Suy ra

$$n^2 < n^2 + 2n + 1 \leq c*n \text{ với mọi } n \geq n_0$$

- Từ đó ta thu được:

$$n < c \text{ với mọi } n \geq n_0 \quad ?!$$



tồn tại các hằng số dương c và n_0 sao cho:
$f(n) \leq cg(n)$ với mọi $n \geq n_0$ }

O lớn và tốc độ tăng

- Kí hiệu O lớn cho cận trên tốc độ tăng của một hàm
- Khi ta nói "$f(n)$ là $O(g(n))$" nghĩa là tốc độ tăng của hàm $f(n)$ không lớn hơn tốc độ tăng của hàm $g(n)$
- Ta có thể dùng kí hiệu O lớn để sắp xếp các hàm theo tốc độ tăng của chúng

	$f(n)$ là $O(g(n))$	$g(n)$ là $O(f(n))$
$g(n)$ có tốc độ tăng lớn hơn $f(n)$	Yes	No
$g(n)$ có tốc độ tăng nhỏ hơn $f(n)$	No	Yes
$g(n)$ và $f(n)$ cùng tốc độ tăng	Yes	Yes

Các biểu thức sai

$$f(n) \not\leq O(g(n))$$

$$f(n) \not\geq O(g(n))$$

Ví dụ O lớn (Big-Oh)

- $f(n) = 50n^3 + 20n + 4$ là $O(n^3)$
  - Cũng đúng khi nói $f(n)$ là $O(n^3+n)$
    - Không hữu ích, vì n^3 có tốc độ tăng lớn hơn rất nhiều so với n, khi n lớn
  - Cũng đúng khi nói $f(n)$ là $O(n^5)$
    - OK, nhưng $g(n)$ nên có tốc độ tăng càng gần với tốc độ tăng của $f(n)$ càng tốt, thì đánh giá thời gian tính mới có giá trị
- $3\log(n) + \log(\log(n)) = O(\ ?\ )$

• Quy tắc đơn giản: Bỏ qua các số hạng có số mũ thấp hơn và các hằng số

Một số quy tắc O lớn

- Nếu $f(n)$ là đa thức bậc d: $f(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_d n^d$

thì $f(n) = O(n^d)$, tức là,

1. Bỏ qua các số hạng có số mũ thấp hơn d
2. Bỏ qua các hằng số

Ví dụ: $3n^3 + 20n^2 + 5 = O(n^3)$

- Nếu $f(n) = O(n^k)$ thì $f(n) = O(n^p)$ với $\forall\ p > k$

Ví dụ: $2n^2 = O(n^2)$ thì $2n^2 = O(n^3)$

Khi đánh giá tiệm cận $f(n) = O(g(n))$, ta nên tìm hàm $g(n)$ có tốc độ tăng càng chậm càng tốt

- Sử dụng lớp các hàm có tốc độ tăng nhỏ nhất có thể

Ví dụ: Nói "$2n = O(n)$" tốt hơn là nói "$2n = O(n^2)$"

- Sử dụng lớp các hàm đơn giản nhất có thể

Ví dụ: Nói "$3n + 5 = O(n)$" thay vì nói "$3n + 5 = O(3n)$"

Ví dụ O lớn

- Tất cả các hàm sau đều là O(n):
  - n, $3n$, $61n + 5$, $22n – 5$, …
- Tất cả các hàm sau đều là O(n^2):
  - n^2, $9\ n^2$, $18\ n^2 + 4n – 53$, …
- Tất cả các hàm sau đều là O($n \log n$):
  - $n(\log n)$, $5n(\log 99n)$, $18 + (4n – 2)(\log (5n + 3))$, …

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Ω- kí hiệu Omega

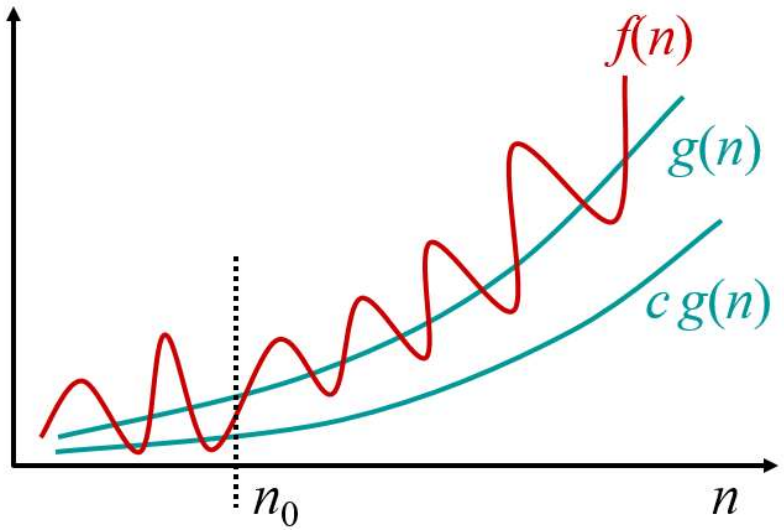
- Đối với hàm $g(n)$ cho trước, ta kí hiệu $\Omega(g(n))$ là tập các hàm

$$\Omega(g(n)) = \{f(n): \text{tồn tại các hằng số dương } c \text{ và } n_0 \text{ sao cho: } cg(n) \leq f(n) \text{ với mọi } n \geq n_0\}$$

(tập tất cả các hàm có **tốc độ tăng lớn hơn hoặc bằng** tốc độ tăng của $g(n)$).

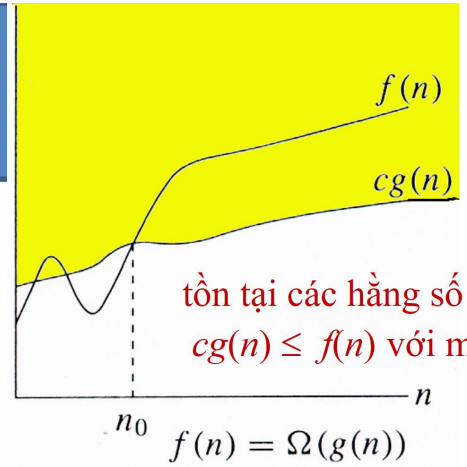
- Ta nói: $g(n)$ là cận dưới tiệm cận của hàm, và viết $f(n) = \Omega(g(n))$.
- $f(n) = \Omega(g(n))$ nghĩa là tồn tại hằng số c sao cho $f(n)$ luôn $\geq cg(n)$ với giá trị n đủ lớn.
- $\Omega(g(n))$ là tập các hàm đạt tới giá trị vô cùng không chậm hơn $g(n)$

$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$
$\Theta(g(n)) \subset \Omega(g(n))$.



Ví dụ 1: Ký hiệu Ω

Chứng minh: $f(n) = n^2 - 2000n$ là $\Omega(n^2)$



tồn tại các hằng số dương c và n_0 sao cho:
$cg(n) \le f(n)$ với mọi $n \ge n_0$

- Cần chỉ ra với giá trị nào của các hằng số n_0 và c thì bất đẳng thức sau đây là đúng với $n \ge n_0$:

$$n^2 - 2000n \ge c*n^2$$

Ta có:

$$n^2 - 2000n \ge 0.5*n^2 \text{ với mọi } n \ge 10000$$

(vì $n^2 - 2000n - 0.5*n^2 = 0.5*n^2 - 2000n$

$= n(0.5*n - 2000) \ge 0$

khi $n \ge 10000$)

Như vậy hằng số $c = 1$, và $n_0 = 10000$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Ví dụ 2: Kí hiệu Omega

$\Omega(g(n)) = \{f(n) : \exists$ hằng số dương c và n_0, sao cho $\forall n \geq n_0$, ta có $0 \leq cg(n) \leq f(n)\}$

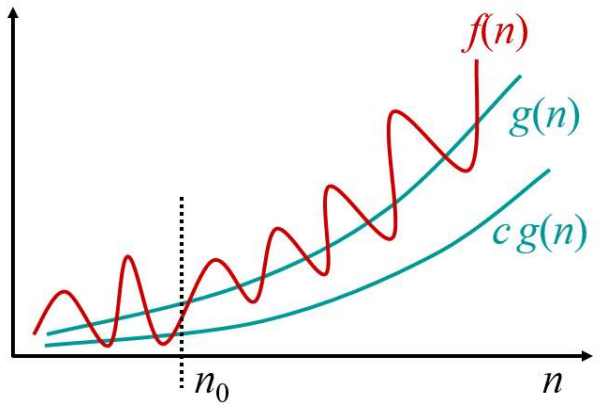
Chứng minh rằng $5n^2 = \Omega(n)$

Giải: Cần tìm c và n_0 sao cho $\boldsymbol{cn} \leq 5n^2$ với $n \geq n_0$

Bất đẳng thức đúng với $c = 1$ và $n_0 = 1$

Nhận xét:

- Nếu $f(n) = \Omega(n^k)$ thì $f(n) = \Omega(n^p)$ với $\forall\ p < k$.
- Khi đánh giá tiệm cận $f(n) = \Omega(g(n))$, ta cần tìm hàm $g(n)$ có tốc độ tăng càng nhanh càng tốt



Ví dụ 3: Ký hiệu Ω

- Rõ ràng: Nếu $f(n)$ là $\Omega(n^k)$ thì nó cũng là $\Omega(n^p)$ với $p < k$

Chứng minh: $f(n) = n^2 - 2000n \notin \Omega(n^3)$.

- Phản chứng. Giả sử trái lại, khi đó phải tìm được hằng số c và số n_0 để cho:

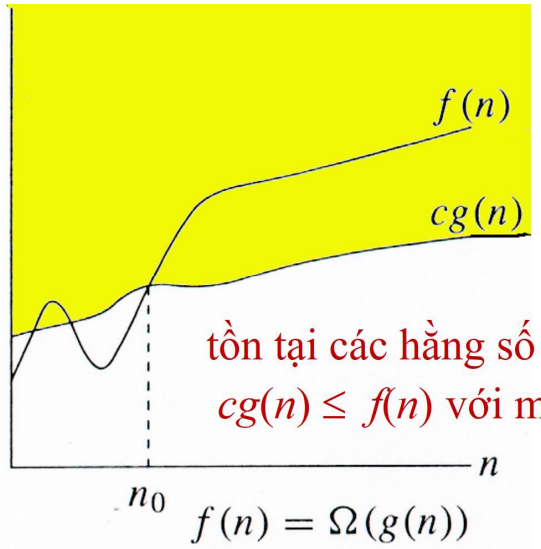
  $n^2 - 2000n \geq c*n^3$ khi $n \geq n_0$

- Suy ra:

  $n^2 \geq n^2 - 2000n \geq c*n^3$ khi $n \geq n_0$

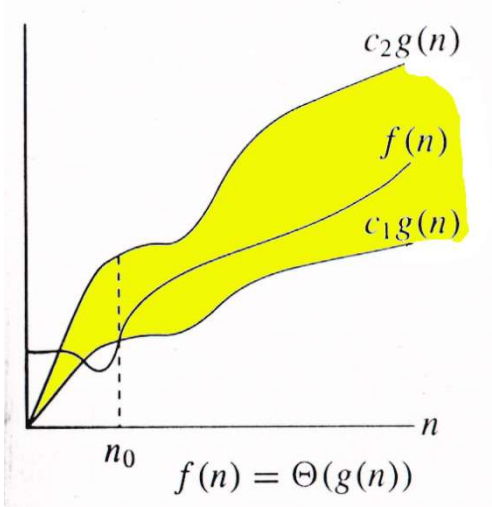
- Từ đó ta thu được:

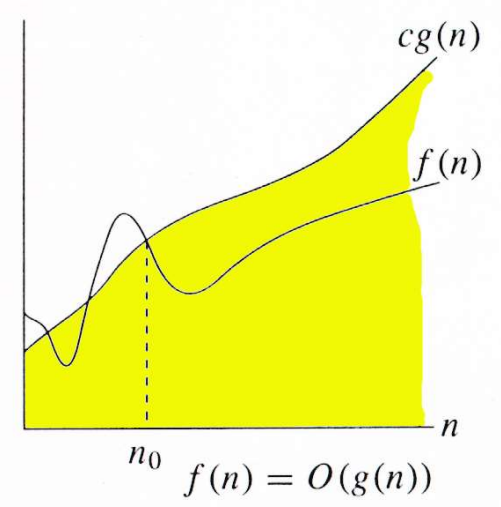
  $1/c \geq n$ với mọi $n \geq n_0$?!

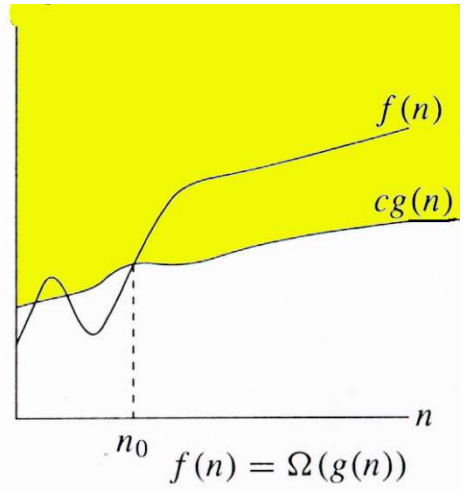


tồn tại các hằng số dương c và n_0 sao cho:
$cg(n) \leq f(n)$ với mọi $n \geq n_0$

Các kí hiệu tiệm cận







Đồ thị minh họa cho các kí hiệu tiệm cận **Θ**, O, và **Ω**

Định lý: Đối với hai hàm bất kỳ $f(n)$ và $g(n)$, ta có $f(n) = \Theta(g(n))$ khi và chỉ khi

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

tức là

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

Định lý: Đối với hai hàm bất kỳ $f(n)$ và $g(n)$, ta có $f(n) = \Theta(g(n))$ khi và chỉ khi $f(n) = O(g(n))$ và $f(n) = \Omega(g(n))$

Ví dụ 1: Chứng minh rằng $f(n) = 5n^2 = \Theta(n^2)$

Vì:

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

  $f(n)$ là $O(g(n))$ nếu tồn tại hằng số $c > 0$ và hằng số nguyên $n_0 \geq 1$ sao cho $f(n) \leq cg(n)$ với $n \geq n_0$

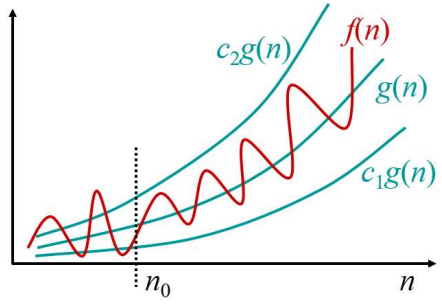
  Chọn $c = 5$ và $n_0 = 1$

- $5n^2 = \Omega(n^2)$

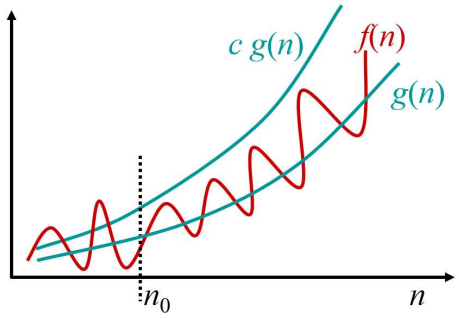
  $f(n)$ là $\Omega(g(n))$ nếu tồn tại hằng số $c > 0$ và hằng số nguyên $n_0 \geq 1$ sao cho $f(n) \geq cg(n)$ với $n \geq n_0$

  Chọn $c = 5$ và $n_0 = 1$

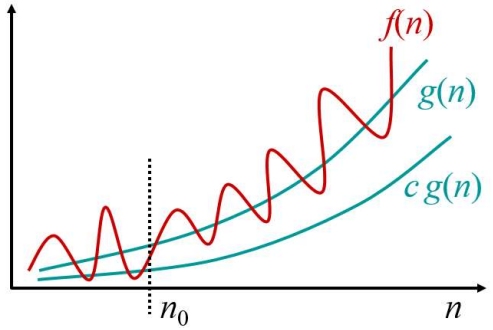
Do đó: $f(n) = \Theta(n^2)$



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



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



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

Định lý: Đối với hai hàm bất kỳ $f(n)$ và $g(n)$, ta có $f(n) = \Theta(g(n))$ khi và chỉ khi $f(n) = O(g(n))$ và $f(n) = \Omega(g(n))$

Ví dụ 2: Chứng minh rằng $f(n) = 3n^2 - 2n + 5 = \Theta(n^2)$

Vì:

$3n^2 - 2n + 5 = O(n^2)$

$f(n)$ là $O(g(n))$ nếu tồn tại hằng số $c > 0$ và hằng số nguyên $n_0 \geq 1$ sao cho $f(n) \leq cg(n)$ với $n \geq n_0$

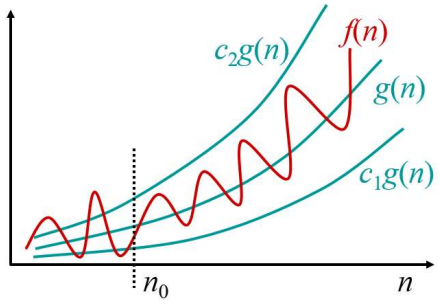
➔ chọn $c = ?$ và $n_0 = ?$

$3n^2 - 2n + 5 = \Omega(n^2)$

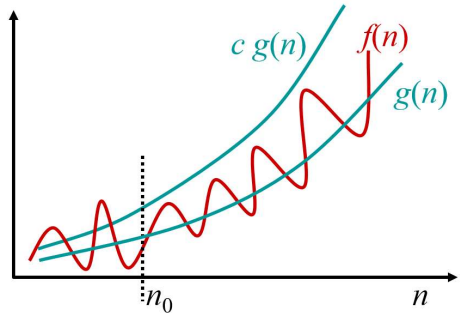
$f(n)$ là $\Omega(g(n))$ nếu tồn tại hằng số $c > 0$ và hằng số nguyên $n_0 \geq 1$ sao cho $f(n) \geq cg(n)$ với $n \geq n_0$

➔ chọn $c = ?$ và $n_0 = ?$

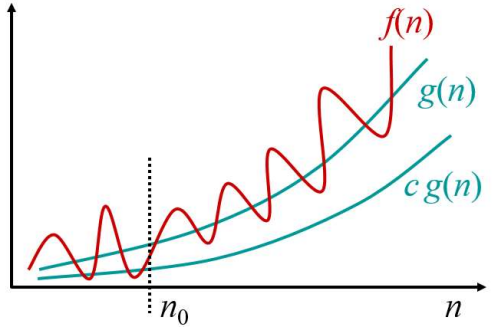
Do đó: $f(n) = \Theta(n^2)$



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

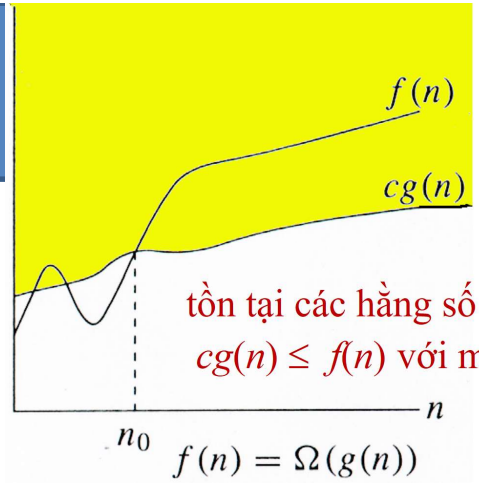


$f(n) = O(g(n))$



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

Bài tập 1



tồn tại các hằng số dương c và n_0 sao cho:
$cg(n) \le f(n)$ với mọi $n \ge n_0$

Chứng minh: $100n + 5 \neq \Omega(n^2)$

Giải: Chứng minh bằng phản chứng

- Giả sử: $100n + 5 = \Omega(n^2)$

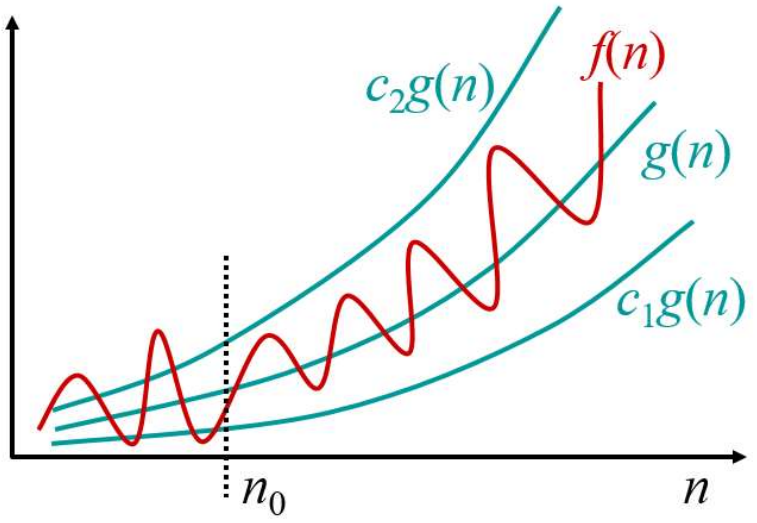
➔ $\exists\ c, n_0$ sao cho: $0 \le cn^2 \le 100n + 5$

- Ta có: $100n + 5 \le 100n + 5n = 105n \quad \forall\ n \ge 1$
- Do đó: $cn^2 \le 105n \Rightarrow n(cn - 105) \le 0$
- Vì $n > 0 \Rightarrow cn - 105 \le 0 \Rightarrow n \le 105/c$

bất đẳng thức trên không đúng vì ***c*** phải là hằng số

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Bài tập 2



Chứng minh: $n \neq \Theta(n^2)$

Giải: Chứng minh bằng phản chứng

– Giả sử: $n = \Theta(n^2)$

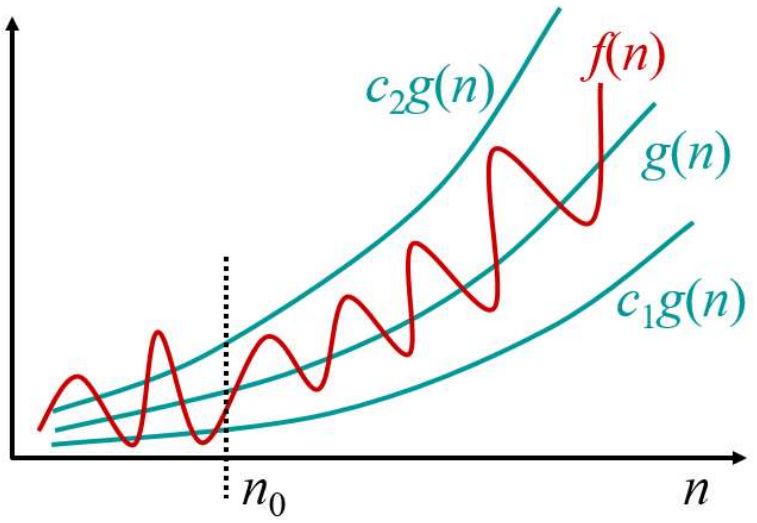
➔ $\exists\ c_1, c_2, n_0$ sao cho: $c_1\ n^2 \leq n \leq c_2\ n^2\ \forall\ n \geq n_0$

➔ $n \leq 1/c_1$

Bất đẳng thức trên không đúng vì c_1 phải là hằng số

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Bài tập 3:Chứng minh



a) $6n^3 \neq \Theta(n^2)$

Giải: Chứng minh bằng phản chứng

- Giả sử: $6n^3 = \Theta(n^2)$

➔ $\exists\ c_1, c_2, n_0$ sao cho: $c_1\ n^2 \leq 6n^3 \leq c_2\ n^2\ \forall\ n \geq n_0$

➔ $n \leq c_2/6\ \forall\ n \geq n_0$

Bất đẳng thức trên không đúng vì c_2 phải là hằng số

b) $n \neq \Theta(\log_2 n)$

Giải: Chứng minh bằng phản chứng

- Giả sử: $n = \Theta(\log_2 n)$

➔ $\exists\ c_1, c_2, n_0$ sao cho: $c_1 \log_2 n \leq n \leq c_2 \log_2 n\ \forall\ n \geq n_0$

➔ $n/\log_2 n \leq c_2\ \ \forall\ n \geq n_0$

Bất đẳng thức trên không đúng vì c_2 phải là hằng số

Cách nói về thời gian tính

- Nói "Thời gian tính là $O(f(n))$", hiểu là **đánh giá trong tình huống tồi nhất (worst case) là $O(f(n))$** (nghĩa là, không tồi hơn $c*f(n)$ với n lớn, vì kí hiệu O lớn cho ta cận trên). [Thường nói: "Đánh giá thời gian tính trong tình huống tồi nhất là $O(f(n))$"]
  - Nghĩa là thời gian tính trong tình huống tồi nhất được xác định bởi một hàm nào đó $g(n) \in O(f(n))$
- Nói "Thời gian tính là $\Omega(f(n))$", hiểu là **đánh giá trong tình huống tốt nhất (best case) là $\Omega(f(n))$** (nghĩa là, không tốt hơn $c*f(n)$ với n lớn, vì kí hiệu Omega cho ta cận dưới). [Thường nói: "Đánh giá thời gian tính 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 nào đó $g(n) \in \Omega(f(n))$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Thời gian tính trong tình huống tồi nhất

- Khi phân tích một thuật toán
  - Chỉ quan tâm đến các trường hợp thuật toán chạy
    - Tốn thời gian chạy nhất

    (tính cận trên cho thời gian tính của thuật toán)
  - Nếu thuật toán có thể giải trong thời gian cỡ hàm *f(n)*
    - Thì trường hợp tồi nhất, thời gian chạy không thể lớn hơn *c*f(n)*
- Hữu ích khi thuật toán áp dụng cho trường hợp
    - Cần biết cận trên cho thuật toán
- Ví dụ:
  - Các thuật toán áp dụng chạy trong nhà máy điện hạt nhân.

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Thời gian tính trong tình huống tốt nhất

- Khi ta phân tích một thuật toán
  - Chỉ quan tâm đến các trường hợp thuật toán chạy
    - Ít thời gian nhất

    (tính cận dưới của thời gian chạy)

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

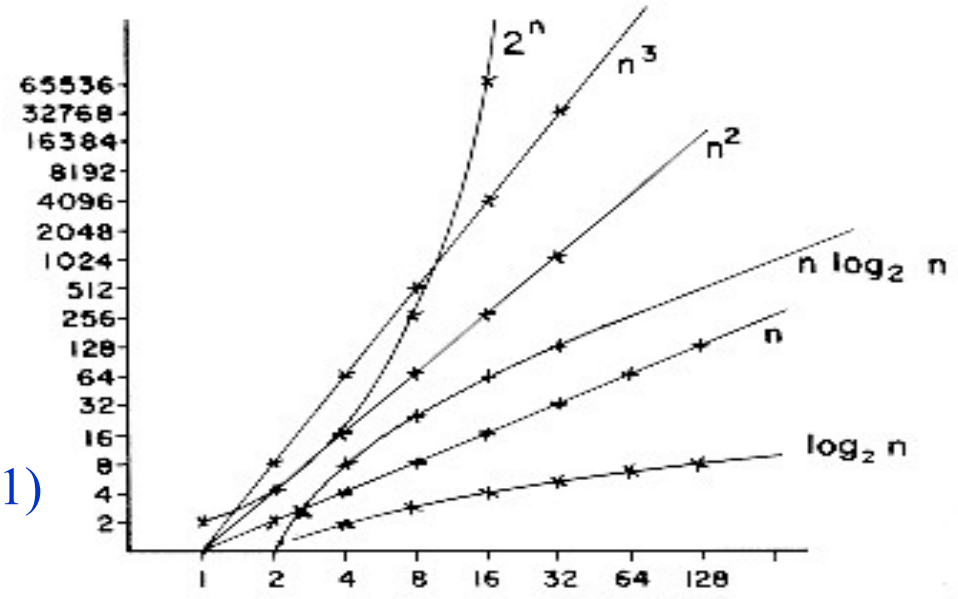
Thời gian tính trong tình huống trung bình

- Nếu thuật toán phải sử dụng nhiều lần
  - hữu ích khi tính đượng thời gian chạy trung bình của thuật toán với kích thước dữ liệu đầu vào là n
- Đánh giá thời gian tính trung bình khó hơn với việc đánh giá thời gian tính tồi/tốt nhất
  - Cần có thông tin về phân phối của dữ liệu dữ liệu

Ví dụ: Thuật toán sắp xếp chèn (Insertion sorting) có thời gian trung bình cỡ n^2

Một số lớp thuật toán cơ bản

- Hằng số ≈ O(**1**)
- Logarithmic ≈ O($\log_2 n$)
- Tuyến tính ≈ O(n)
- N-Log-N ≈ O($n \log_2 n$)
- Bình phương (Quadratic) ≈ O(n^2)
- Bậc 3 (Cubic) ≈ O(n^3)
- Hàm mũ (exponential) ≈ O($\mathbf{a}^n$) ($a > 1$)
- Đa thức (polynomial): O(n^k) ($k \geq 1$)



- Thuật toán có đánh giá thời gian tính là $O(n^k)$ được gọi là ***thuật toán thời gian tính đa thức*** (hay vắn tắt: ***thuật toán đa thức***)
- Thuật toán đa thức được coi là thuật toán hiệu quả.
- Các thuật toán với thời gian tính hàm mũ là không hiệu quả.

Phân loại thời gian tính của thuật toán

- Thời gian để giải một bộ dữ liệu đầu vào của
  - Thuật toán có thời gian tính tuyến tính (Linear Algorithm):
    - Không bao giờ lớn hơn $c*n$
  - Thuật toán có thời gian tính là hàm đa thức bậc 2 (Quadratic Algorithm):
    - Không bao giờ lớn hơn $c*n^2$
  - Thuật toán có thời gian tính là hàm đa thức bậc ba (Cubic Algorithm):
    - Không bao giờ lớn hơn $c*n^3$
  - Thuật toán có thời gian tính đa thức (Polynomial Algorithm)
    - Không bao giờ lớn hơn n^k
  - Thuật toán có thời gian tính hàm mũ (Exponential Algorithm)
    - Không bao giờ lớn hơn c^n

với c và k là các hằng số tương ứng

Sự tương tự giữa so sánh các hàm số và so sánh số

Chú ý rằng mối quan hệ giữa các hàm số cũng giống như mỗi quan hệ "<, >, =" giữa các số

$$f \leftrightarrow g \;\approx\; a \leftrightarrow b$$

$$f(n) = O(g(n)) \;\approx\; a \leq b$$
$$f(n) = \Omega(g(n)) \;\approx\; a \geq b$$
$$f(n) = \Theta(g(n)) \;\approx\; a = b$$
$$f(n) = o(g(n)) \;\approx\; a < b$$
$$f(n) = \omega(g(n)) \;\approx\; a > b$$

Ví dụ phân tích thuật toán

Ví dụ. Xét thuật toán tìm kiếm tuần tự để giải bài toán:

- **Đầu vào:** *key*, n và dãy số $s_1, s_2, \ldots, s_n$.
- **Đầu ra:** Vị trí phần tử có giá trị *key* hoặc là n+1 nếu không tìm thấy.

```
int  Linear_Search(s,n,key)
{
    i=0;
    do
        i=i+1;
    while  (i<=n) or (key != s_i);
    return i;
}
```

Ví dụ phân tích thuật toán

```
int  Linear_Search(s,n,key)
{
     i=0;
     do
          i=i+1;
     while  (i<=n) or (key != s_i);
     return i;
}
```

Cần đánh giá **thời gian tính tốt nhất, tồi nhất, trung bình** của thuật toán với độ dài đầu vào là n.

Rõ ràng thời gian tính của thuật toán có thể đánh giá bởi số lần thực hiện câu lệnh $i=i+1$ trong vòng lặp **do-while**.

Thời gian tính tốt nhất: Nếu $s_1 = key$ thì câu lệnh $i=i+1$ trong thân vòng lặp do-while thực hiện 1 lần. Do đó thời gian tính tốt nhất của thuật toán là $\Theta(1)$.

Thời gian tính tồi nhất: Nếu *key* không có mặt trong dãy đã cho, thì câu lệnh $i= i+1$ thực hiện n lần. Vì thế thời gian tính tồi nhất của thuật toán là $O(n)$.

Ví dụ phân tích thuật toán

```
int  Linear_Search(s,n,key)
{
      i=0;
      do
            i=i+1;
      while  (i<=n) or (key != s_i);
      return i;
}
```

- **Thời gian tính trung bình của thuật toán:**
  - Nếu *key* tìm thấy ở vị trí thứ *i* của dãy ($key = s_i$) thì câu lệnh $i = i+1$ phải thực hiện *i* lần ($i = 1, 2, ..., n$),
  - Nếu *key* không có mặt trong dãy đã cho thì câu lệnh $i = i+1$ phải thực hiện *n* lần.

→ Từ đó suy ra số lần trung bình phải thực hiện câu lệnh $i = i+1$ là

$$[(1 + 2 + \ldots + n) + n] /(n+1) = [n(n+1)/2 + n]/(n+1).$$

Ta có:

$$n/4 \leq [n(n+1)/2 + n]/(n+1) \leq n \text{ với mọi } n \geq 1.$$

Vậy thời gian tính trung bình của thuật toán là $\Theta(n)$.

Ví dụ

Đưa ra đánh giá kí hiệu tiệm cận big-Oh cho thời gian tính $T(n)$ của các đoạn code sau:

b)
```
int x = 0;
for (int i = 1; i <= n; i *= 2)
    x=x+1;
```

- Trả lời:

Câu lệnh loop **for** được thực hiện $\log_2 n$ lần, do đó $T(n) = O(\log_2 n)$.

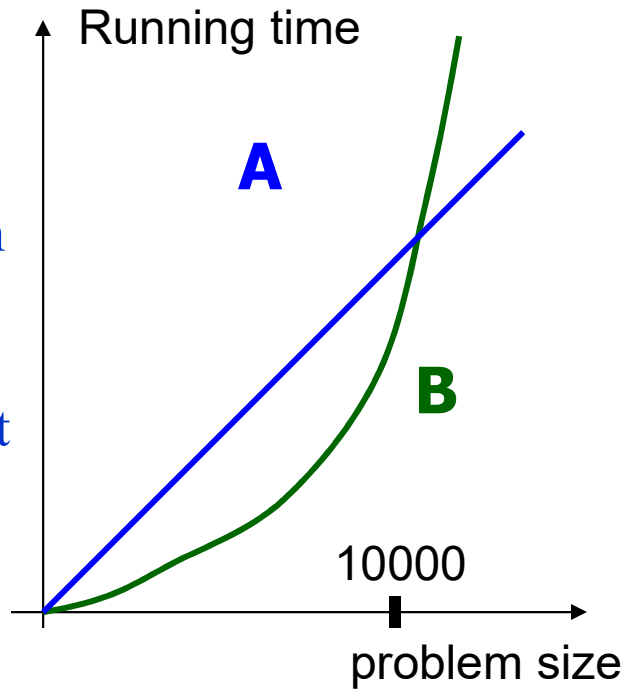
c)
```
int x = 0;
for (int i = n; i > 0; i /= 2)
    x=x+1;
```

- Trả lời:

Câu lệnh loop **for** được thực hiện …..

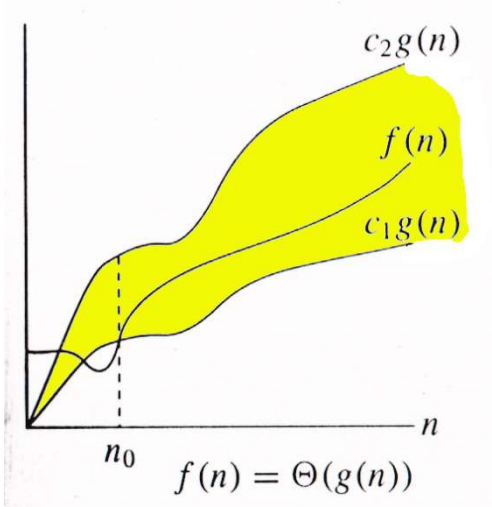
Ghi chú

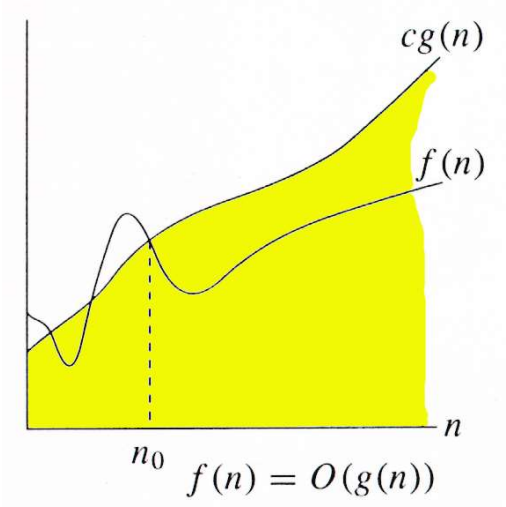
- Giả sử có 2 thuật toán:
  - Thuật toán A có thời gian chạy $30000n$
  - Thuật toán B có thời gian chạy $3n^2$
- Theo kí hiệu tiệm cận, thuật toán A tốt hơn thuật toán B
- Tuy nhiên, nếu kích thước bài toán cần giải quyết luôn nhỏ hơn 10000, thì thuật toán B nhanh hơn thuật toán A

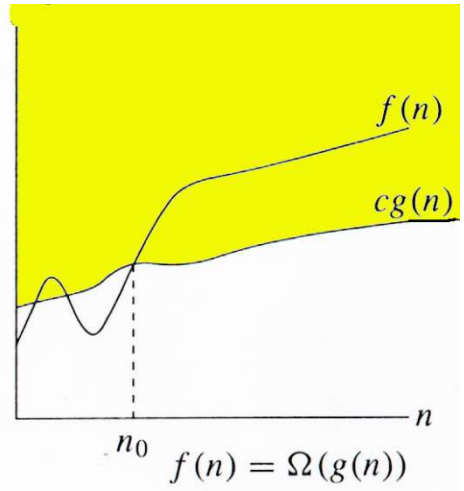


NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Các kí hiệu tiệm cận







Đồ thị minh họa cho các kí hiệu tiệm cận **Θ**, *O*, và **Ω**

- $f(n)$ và $g(n)$ có cùng tốc độ tăng $f(n) = \Theta(g(n))$
- $f(n)$ có tốc độ tăng không lớn hơn tốc độ tăng của $g(n)$ $f(n) = O(g(n))$
- $f(n)$ có tốc độ tăng không nhỏ hơn tốc độ tăng của $g(n)$ $f(n) = \Omega(g(n))$

1) $3n^2 - 100n + 6 \;?\; O(n)$
2) $3n^2 - 100n + 6 \;?\; O(n^2)$
3) $3n^2 - 100n + 6 \;?\; O(n^3)$
4) $3n^2 - 100n + 6 \;?\; \Omega(n)$
5) $3n^2 - 100n + 6 \;?\; \Omega(n^2)$
6) $3n^2 - 100n + 6 \;?\; \Omega(n^3)$
7) $3n^2 - 100n + 6 \;?\; \Theta(n)$
8) $3n^2 - 100n + 6 \;?\; \Theta(n^2)$
9) $3n^2 - 100n + 6 \;?\; \Theta(n^3)$

Giải công thức đệ quy

- Trên thực tế, khi phân tích độ phức tạp của một thuật toán nào đó nhờ sử dụng công thức đệ quy tuyến tính, thì công thức đệ quy ít khi có bậc lớn hơn 2.
- Do đó, ta cũng có thể sử dụng hai phương pháp sau đây để giải công thức đệ quy
  - Thay thế quay lui: xuất phát từ công thức đã cho, ta thế lần lượt lùi về các số hạng phía trước của công thức đệ quy
  - Cây đệ quy: biểu diễn công thức đệ quy bởi một cây đệ quy. Xuất phát từ công thức đệ quy đã cho, ta biểu diễn các số hạng đệ quy có kích thước dữ liệu đầu vào lớn ở mức A nào đó trên cây bởi các dữ liệu đầu vào nhỏ hơn ở mức A+1 của vây, và tính các số hạng không đệ quy. Cuối cùng, tính tổng tất cả các số hạng không đệ quy ở tất cả các mức của cây.
  - **Định lý thợ (Master problem)**

Định lý Thợ (Master Theorem)

- Cung cấp công cụ để giải công thức đệ qui dạng

$$T(n) = a\,T(n/b) + f(n)$$

Các giả thiết:

$a \geq 1$ và $b \geq 2$ là các hằng số

$f(n)$ hàm đa thức

$T(n)$ được xác định với đối số nguyên không âm

Ta dùng n/b thay cho cả $\lfloor n/b \rfloor$ lẫn $\lceil n/b \rceil$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Định lý Thợ (Master Theorem)

Xét công thức đệ qui $T(n) = a\ T(n/b) + f(n)$ thoả mãn các điều kiện đã nêu, khi đó có thể đánh giá tiệm cận $T(n)$ như sau:

1. Nếu $f(n)=O(n^{\log_b a-\varepsilon})$ đối với hằng số $\varepsilon > 0$ nào đó, thì $T(n)= \Theta(n^{\log_b a})$
2. Nếu $f(n) = \Theta(n^{\log_b a})$, thì $T(n) = \Theta(n^{\log_b a} \log n)$
3. Nếu $f(n) = \Omega(n^{\log_b a+\varepsilon})$ với hằng số $\varepsilon > 0$ nào đó, và nếu $a\ f(n/b) \le c\ f(n)$ với hằng số $c < 1$ và với mọi n đủ lớn, thì $T(n)= \Theta(f(n))$.

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \ge 1$, $b \ge 2$, $c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui

$T(n) = a\ T(n/b) + f(n)$

xác định với $n \ge 0$; và $f(n) = c*n^k$, hay nói cách khác $f(n) = \Theta\ (n^k)$ với $k \ge 0$

1. Nếu $a > b^k$, thì $T(n) = \Theta\ (n^{log_b a})$
2. Nếu $a = b^k$, thì $T(n) = \Theta\ (n^k log n)$
3. Nếu $a < b^k$, thì $T(n) = \Theta\ (n^k)$

Master Theorem: Pitfalls

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1$, $b \geq 2$, $c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui

$$T(n) = a\,T(n/b) + f(n)$$

xác định với $n \geq 0$. và $f(n) = c*n^k$ hay nói cách khác $f(n) = \Theta(n^k)$

1. Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$.
2. Nếu $a = b^k$, thì $T(n) = \Theta(n^k \log n)$.
3. Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$.

- Ta không thể sử dụng định lý thợ nếu:
  - $T(n)$ không phải là hàm đơn điệu, ví dụ $T(n) = \sin(n)$
  - $f(n)$ không phải là hàm đa thức, ví dụ $T(n) = 2T(n/2) + 2^n$
  - b không được biểu diễn bởi hằng số, ví dụ $T(n) = T(\sqrt{n})$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Master Theorem: Ví dụ 1

- Cho $T(n) = T(n/2) + ½\ n^2 + n$. Xác định giá trị các tham số?

  a = 1

  b = 2

  k = 2

  Do đó, điều kiện nào thảo mãn?

  $1 < 2^2$, → điều kiện 3

- Vậy $T(n) \in \Theta(n^k) = \Theta(n^2)$

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1$, $b \geq 2$, $c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui

$T(n) = a\ T(n/b) + f(n)$

xác định với $n \geq 0$; và $f(n) = c*n^k$, hay nói cách khác $f(n) = \Theta(n^k)$ với $k \geq 0$

1. Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$
2. Nếu $a = b^k$, thì $T(n) = \Theta(n^k \log n)$
3. Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Master Theorem: Ví dụ 2

- Cho T(n)= 2 T(n/4) + $\sqrt{n}$ + 42. Xác định giá trị các tham số?

  a = 2

  b = 4

  k = 1/2

  Do đó, điều kiện nào thỏa mãn?

  $2 = 4^{1/2}$ ➔ điều kiện 2 thỏa mãn

- Vậy $$T(n) \in \Theta(n^d \log n) = \Theta(\log n\sqrt{n})$$

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1$, $b \geq 2$, $c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui

$T(n) = a\ T(n/b) + f(n)$

xác định với $n \geq 0$; và $f(n) = c*n^k$, hay nói cách khác $f(n) = \Theta(n^k)$ với $k \geq 0$

1. Nếu $a > b^k$, thì $T(n) = \Theta(n^{log_b a})$
2. Nếu $a = b^k$, thì $T(n) = \Theta(n^k log n)$
3. Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$

Master Theorem: Ví dụ 3

- Cho T(n)= 3 T(n/2) + (3/4)n + 1. Xác định giá trị các tham số?

  a = 3

  b = 2

  k = 1

  Do đó, điều kiện áp dụng là?

  $3 > 2^1$, ➔ điều kiện 2 thỏa mãn

  - Vậy $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$

- Chú ý $\log_2 3 \approx 1.584...$, liệu ta có thể nói $T(n) \in \Theta(n^{1.584})$????

  NO, vì $\log_2 3 \approx 1.5849...$ và $n^{1.584} \notin \Theta(n^{1.5849})$

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1$, $b \geq 2$, $c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui

$T(n) = a\ T(n/b) + f(n)$

xác định với $n \geq 0$; và $f(n) = c*n^k$, hay nói cách khác $f(n) = \Theta(n^k)$ với $k \geq 0$

1. Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$
2. Nếu $a = b^k$, thì $T(n) = \Theta(n^k \log n)$
3. Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$

Ví dụ 3: Tìm kiếm nhị phân dưới dạng đệ quy

Đầu vào: Mảng *S* gồm *n* phần tử: S[0],…,S[*n-1*] đã sắp xếp theo thứ tự tăng dần; Giá trị ***key***.

Đầu ra: chỉ số của phần tử có giá trị ***key*** nếu có; -1 nếu ***key*** không xuất hiện trong mảng *S*

```
int binsearch(int low, int high, int S[], int key)
{
    if (low <= high)
    {
        mid = (low + high) / 2;
        if (S[mid]==key) return mid;
        else if (key < S[mid])
            return binsearch(low, mid-1, S, key);
        else
            return binsearch(mid+1, high, S, key);
     }
     else return -1;
}
```

Ví dụ 3: Tìm kiếm nhị phân dưới dạng đệ quy

```
int binsearch(int low, int high, int S[], int key)
{
    if (low <= high)
    {
        mid = (low + high) / 2;
        if (S[mid]==key) return mid;
        else if (key < S[mid])
            return binsearch(low, mid-1, S, key);
        else
            return binsearch(mid+1, high, S, key);
     }
     else return -1;
}
```

➔ T(0) = ?

➔ binsearch(0, -1, *S*, *key*);

➔ T(1) = ?

➔ binsearch(0, 0, *S*, *key*);

➔ binsearch(0, -1, *S*, *key*);

➔ binsearch(1, 0, *S*, *key*);

➔ binsearch(0, *n*-1, *S*, *key*);

Gọi T(*n*): số lần hàm binsearch được gọi trong trường hợp tồi nhất khi mảng *S* có *n* phần tử

- $T(0) = 1$
- $T(1) = 2$
- $T(2) = T(1) + 1 = 3$
- $T(4) = T(2) + 1 = 4$
- $T(8) = T(4) + 1 = 4 + 1 = 5$
- $T(n) = T(n/2) + 1$

Công thức đệ quy:

$T(n) = T(n/2) + 1$

$T(0) = 1$

$T(1) = 2$

Bài tập: Hãy giải công thức đệ quy này

Ví dụ 3: Tìm kiếm nhị phân dưới dạng đệ quy

- Cho T(n) = T(n/2) + 1. Xác định giá trị các tham số?

  a = 1

  b = 2

  k = 0

  Do đó, điều kiện nào thảo mãn?

  $1 = 2^0$, → điều kiện 2

- Vậy $T(n) \in \Theta(n^0 \log n) = \Theta(\log n)$

Định lý Thợ rút gọn (Simplified Master Theorem):

Giả sử $a \geq 1$, $b \geq 2$, $c > 0$ là các hằng số. Xét $T(n)$ là công thức đệ qui

$T(n) = a\,T(n/b) + f(n)$

xác định với $n \geq 0$; và $f(n) = c*n^k$, hay nói cách khác $f(n) = \Theta(n^k)$ với $k \geq 0$

1. Nếu $a > b^k$, thì $T(n) = \Theta(n^{\log_b a})$
2. Nếu $a = b^k$, thì $T(n) = \Theta(n^k \log n)$
3. Nếu $a < b^k$, thì $T(n) = \Theta(n^k)$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Nội dung chi tiết

1. Giới thiệu bài toán
2. Thuật toán và độ phức tạp
3. **Phương pháp sinh**
4. Thuật toán quay lui

3. PHƯƠNG PHÁP SINH

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- Sinh xâu nhị phân độ dài *n*
- Sinh tập con *m* phần tử của tập *n* phần tử
- Sinh hoán vị của *n* phần tử

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

3.1. SƠ ĐỒ THUẬT TOÁN

Phương pháp sinh có thể áp dụng để giải bài toán liệt kê tổ hợp đặt ra nếu như hai điều kiện sau được thực hiện:

1) *Có thể xác định được một thứ tự trên tập các cấu hình tổ hợp cần liệt kê. Từ đó có thể xác định được cấu hình đầu tiên và cấu hình cuối cùng trong thứ tự đã xác định.*

2) *Xây dựng được thuật toán từ cấu hình chưa phải là cuối cùng đang có, đưa ra cấu hình kế tiếp nó.*

Thuật toán nói đến trong điều kiện 2) được gọi là Thuật toán *Sinh kế tiếp*

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Thuật toán sinh

```
void Generate ( )
{
   <Xây dựng cấu hình đầu tiên>;
   Stop=false;
   while (!stop)
   {
       <Đưa ra cấu hình đang có>;
       if (cấu hình đang có chưa là cuối cùng)
          <Sinh_kế_tiếp>;
     else Stop = true;
   }
}
```

- <Sinh_kế_tiếp> là thủ tục thực hiện thuật toán sinh kế tiếp đã xây dựng trong điều kiện 2). Thủ tục này sẽ xây dựng cấu hình kế tiếp của cấu hình đang có trong thứ tự đã xác định.
- **Chú ý:** Do tập các cấu hình tổ hợp cần liệt kê là hữu hạn nên luôn có thể xác định được thứ tự trên nó. Tuy nhiên, thứ tự cần xác định sao cho có thể xây dựng được thuật toán Sinh kế tiếp.

3. PHƯƠNG PHÁP SINH

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- Sinh xâu nhị phân độ dài *n*
- Sinh tập con *m* phần tử của tập *n* phần tử
- Sinh hoán vị của *n* phần tử

Sinh các xâu nhị phân độ dài *n*

Bài toán: Liệt kê tất cả các xâu nhị phân độ dài n:

$$b_1\ b_2\ ...\ b_n, \text{ trong đó } b_i \in \{0, 1\}.$$

Giải:

- Thứ tự từ điển:

Xem mỗi xâu nhị phân $b = b_1\ b_2\ ...\ b_n$ là biểu diễn nhị phân của một số nguyên $p(b)$

Ta nói xâu nhị phân $b = b_1\ b_2\ ...\ b_n$ ***đi trước*** *xâu* nhị phân $b' = b'_1\ b'_2\ ...\ b'_n$ trong thứ tự tự nhiên và ký hiệu là $b \prec b'$ nếu $p(b) < p(b')$.

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sinh các xâu nhị phân độ dài *n*

Ví dụ: Khi n=3, các xâu nhị phân độ dài 3 được liệt kê theo thứ tự tự nhiên trong bảng bên

Dễ thấy thứ tự này trùng với thứ tự từ điển

b	*p*(*b*)
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sinh các xâu nhị phân độ dài *n*

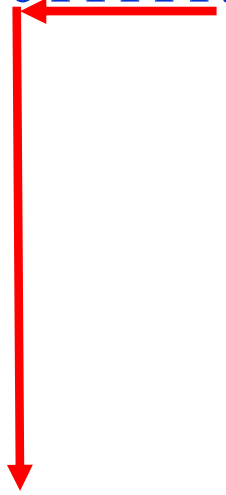
Thuật toán sinh kế tiếp:

- Xâu đầu tiên sẽ là 0 0 ... 0,
- Xâu cuối cùng là 1 1 ... 1.
- Giả sử $b_1\ b_2 \ldots b_n$ *là dãy đang có.*
  - Nếu xâu này gồm toàn số 1 → kết thúc,
  - Trái lại, xâu kế tiếp nhận được bằng cách cộng thêm 1 (theo modul 2, có nhớ) vào xâu hiện tại.
- Từ đó ta có qui tắc sinh xâu kế tiếp như sau:
  - Tìm i đầu tiên (theo thứ tự $i=n, n\text{-}1, \ldots, 1$) thoả mãn $b_i = 0$.
  - Gán lại $b_i = 1$ và $b_j = 0$ với tất cả $j > i$. Dãy mới thu được sẽ là xâu cần tìm.

Sinh các xâu nhị phân độ dài *n*

- Tìm i đầu tiên (theo thứ tự $i=n, n\text{-}1, \ldots, 1$) thoả mãn $b_i = 0$.
- Gán lại $b_i = 1$ và $b_j = 0$ với tất cả $j > i$. Dãy mới thu được sẽ là xâu cần tìm.

Ví dụ: xét xâu nhị phân độ dài 10: b = 1101011111.
Tìm xâu nhị phân kế tiếp.



- Ta có i = 5. Do đó, đặt:
  - b_5 = 1,
  - và b_i = 0, i = 6, 7, 8, 9, 10

→ ta thu được xâu nhị phân kế tiếp là 1101100000.

```
  1101011111
+            1
------------
  1101100000
```

Thuật toán sinh xâu kế tiếp

```
void  Next_Bit_String ( )
/*Sinh xâu nhị phân kế tiếp theo thứ tự từ điển của xâu
   đang có   b1 b2 ... bn ≠ 1 1 ... 1          */
{
   i=n;
   while  (bi == 1)
   {
        bi = 0;
        i=i-1;
   }
   bi = 1;
}
```

- Tìm i đầu tiên (theo thứ tự i=n, n-1, ..., 1) thoả mãn $b_i = 0$.
- Gán lại $b_i = 1$ và $b_j = 0$ với tất cả $j > i$. Dãy mới thu được sẽ là xâu cần tìm.

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

3. PHƯƠNG PHÁP SINH

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- Sinh xâu nhị phân độ dài *n*
- **Sinh tập con *m* phần tử của tập *n* phần tử**
- Sinh hoán vị của *n* phần tử

Sinh các tập con *m* phần tử của tập *n* phần tử

Bài toán đặt ra là: Cho $X = \{1, 2, \ldots, n\}$. Hãy liệt kê các tập con *m* phần tử của *X*.

Giải:

- Thứ tự từ điển:

Mỗi tập con *m* phần tử của *X* có thể biểu diễn bởi bộ có thứ tự gồm *m* thành phần

$$a = (a_1, a_2, \ldots, a_m)$$

thoả mãn

$$1 \le a_1 < a_2 < \ldots < a_m \le n.$$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sinh các tập con *m* phần tử của tập *n* phần tử

Ta nói tập con $a = (a_1, a_2, \ldots, a_m)$ đi trước tập con $a' = (a'_1, a'_2, \ldots, a'_m)$ trong thứ tự từ điển và kí hiệu là $a \prec a'$, nếu tìm được chỉ số k $(1 \le k \le m)$ sao cho:

$$a_1 = a'_1 , a_2 = a'_2, \ldots , a_{k-1} = a'_{k-1},$$

$$a_k < a'_k .$$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sinh các tập con *m* phần tử của tập *n* phần tử

Ví dụ: Các tập con 3 phần tử của $X = \{1, 2, 3, 4, 5\}$ được liệt kê theo thứ tự từ điển như sau

1	2	3
1	2	4
1	2	5
1	3	4
1	3	5
1	4	5
2	3	4
2	3	5
2	4	5
3	4	5

Sinh các tập con *m* phần tử của tập *n* phần tử

- **Thuật toán sinh kế tiếp:**
  - Tập con đầu tiên là (1, 2, ... , m)
  - Tập con cuối cùng là (n-m+1, n-m+2, ..., n).
  - Giả sử $a=(a_1, a_2, \dots, a_m)$ là tập con đang có chưa phải cuối cùng, khi đó tập con kế tiếp trong thứ tự từ điển có thể xây dựng bằng cách thực hiện các quy tắc biến đổi sau đối với tập đang có:
    - Tìm từ bên phải dãy $a_1, a_2, \dots, a_m$ phần tử $a_i \neq n-m+i$
    - Thay a_i bởi $a_i + 1$
    - Thay a_j bởi $a_i + j - i$, với $j = i+1, i+2, \dots, m$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sinh các tập con *m* phần tử của tập *n* phần tử

Ví dụ: $n = 6, m = 4$

Giả sử đang có tập con (1, 2, 5, 6), cần xây dựng tập con kế tiếp nó trong thứ tự từ điển.

- Tìm từ bên phải dãy $a_1, a_2, \ldots, a_m$ phần tử $a_i \neq n\text{-}m\text{+}i$
- Thay a_i bởi $a_i + 1$
- Thay a_j bởi $a_i + j - i$, với $j = i+1, i+2, \ldots, m$

- Ta có i=2:

Dãy: (1, **2,** 5, 6)

Giá trị $n\text{-}m\text{+}i$: (3, **4,** 5, 6)

thay $a_2 = a_2+1 = 3$

$a_3 = a_i + j - i = a_2 + 3 - 2 = 4$

$a_4 = a_i + j - i = a_2 + 4 - 2 = 5$

ta được tập con kế tiếp (1, 3, 4, 5).

Thuật toán sinh *m*-tập kế tiếp

```
void Next_Combination( )
/*Sinh tập con kế tiếp theo thứ tự từ điển của tập con
   (a1, a2,..., am)  ≠  (n-m+1,...,n) */
{
    i=m;
    while (ai == n-m+i)   i=i-1;
    ai= ai + 1;
    for  (j=i+1; j++; j <= m) aj = ai + j - i;
}
```

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

3. PHƯƠNG PHÁP SINH

3.1. Sơ đồ thuật toán

3.2. Sinh các cấu hình tổ hợp cơ bản

- Sinh xâu nhị phân độ dài *n*
- Sinh tập con *m* phần tử của tập *n* phần tử
- **Sinh hoán vị của *n* phần tử**

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sinh các hoán vị của tập *n* phần tử

Bài toán: Cho $X = \{1, 2, \ldots, n\}$, hãy liệt kê các hoán vị từ n phần tử của X.

Giải:

- Thứ tự từ điển:
  - Mỗi hoán vị từ n phần tử của X có thể biểu diễn bởi bộ có thứ tự gồm n thành phần

$$a = (a_1, a_2, \ldots, a_n)$$

thoả mãn

$$a_i \in X,\ i = 1, 2, \ldots, n,\ a_p \neq a_q, p \neq q.$$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sinh các hoán vị của tập *n* phần tử

Ta nói hoán vị $a = (a_1, a_2, ..., a_n)$ đi trước hoán vị $a' = (a'_1, a'_2, ..., a'_n)$ trong thứ tự từ điển và ký hiệu là $a \prec a'$, nếu tìm được chỉ số k ($1 \le k \le n$) sao cho:

$$a_1 = a'_1 , a_2 = a'_2, ... , a_{k-1} = a'_{k-1},$$

$$a_k < a'_k .$$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sinh các hoán vị của tập *n* phần tử

- Ví dụ: Các hoán vị từ 3 phần tử của X ={1, 2, 3} được liêt kê theo thứ tự từ điển như sau:

1	2	3
1	3	2
2	1	3
2	3	1
3	1	2
3	2	1

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sinh các hoán vị của tập *n* phần tử

- **Thuật toán sinh kế tiếp:**
  - Hoán vị đầu tiên: $(1, 2, \ldots, n)$
  - Hoán vị cuối cùng: $(n, n\text{-}1, \ldots, 1)$.
  - Giả sử $a = (a_1, a_2, \ldots, a_n)$ là hoán vị chưa phải cuối cùng, khi đó hoán vị kế tiếp nó có thể xây dựng nhờ thực hiện các biến đổi sau:
    - Tìm từ phải qua trái: chỉ số j đầu tiên thoả mãn $a_j < a_{j+1}$ (nói cách khác: j là chỉ số lớn nhất thoả mãn $a_j < a_{j+1}$);
    - Tìm a_k là số nhỏ nhất còn lớn hơn a_j trong các số ở bên phải a_j ;
    - Đổi chỗ a_j với a_k ;
    - Lật ngược đoạn từ a_{j+1} đến a_n .

Sinh các hoán vị của tập *n* phần tử

Ví dụ: Giả sử đang có hoán vị (3, 6, 2, 5, 4, 1), cần xây dựng hoán vị kế tiếp nó trong thứ tự từ điển.

- Tìm từ phải qua trái: chỉ số j đầu tiên thoả mãn $a_j < a_{j+1}$ (nói cách khác: j là chỉ số lớn nhất thoả mãn $a_j < a_{j+1}$);
- Tìm a_k là số nhỏ nhất còn lớn hơn a_j trong các số ở bên phải a_j
- Đổi chỗ a_j với a_k ;
- Lật ngược đoạn từ a_{j+1} đến a_n

- Ta có chỉ số $j = 3$ ($a_3 = 2 < a_4 = 5$).
- Số nhỏ nhất còn lớn hơn a_3 trong các số bên phải của a_3 là $a_5 = 4$. Đổi chỗ a_3 với a_5 trong hoán vị đã cho (3, 6, **2**, 5, **4**, 1)

  ta thu được (3, 6, **4**, 5, **2**, 1),
- Cuối cùng, lật ngược thứ tự đoạn $a_4\ a_5\ a_6$ ta thu được hoán vị kế tiếp (3, 6, 4, **1, 2, 5**).

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Thuật toán sinh hoán vị kế tiếp

```
void Next_Permutation ( )
{
    /*Sinh hoán vị kế tiếp (a1, a2, ..., an) ≠(n, n-1, ..., 1)   */
    // Tìm j là chỉ số lớn nhất thoả aj < aj+1 :
    j=n-1; while (aj > aj+1 ) j=j-1;
    // Tìm ak là số nhỏ nhất còn lớn hơn aj ở bên phải aj :
    k=n; while (aj > ak ) k=k-1;
    Swap(aj, ak); //đổi chỗ aj với ak
    // Lật ngược đoạn từ aj+1 đến an :
    r=n; s=j+1;
    while ( r>s )
    {
        Swap(ar, as); // đổi chỗ ar với as
        r=r-1; s= s+1;
    }
}
```

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Nội dung chi tiết

1. Giới thiệu bài toán
2. Thuật toán và độ phức tạp
3. Phương pháp sinh
4. **Thuật toán quay lui**

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Thuật toán quay lui (Backtracking)

4.1. Sơ đồ thuật toán

4.2. Liệt kê các cấu hình tổ hợp cơ bản

- Liệt kê xâu nhị phân độ dài *n*
- Liệt kê tập con *m* phần tử của tập *n* phần tử
- Liệt kê hoán vị

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sơ đồ thuật toán quay lui

- ***Bài toán liệt kê (Q):*** *Cho $A_1, A_2,..., A_n$ là các tập hữu hạn. Ký hiệu*

$$A = A_1 \times A_2 \times ... \times A_n = \{ (a_1, a_2, ..., a_n): \ a_i \in A_i , \ i=1, 2, ..., n\}.$$

Giả sử P là tính chất cho trên A. Vấn đề đặt ra là liệt kê tất cả các phần tử của A thoả mãn tính chất P:

$$D = \{ a = (a_1, a_2, ..., a_n) \in A: \ a \text{ thoả mãn tính chất } P \}.$$

- Các phần tử của tập *D* được gọi là các ***lời giải chấp nhận được.***

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sơ đồ thuật toán quay lui

Tất cả các bài toán liệt kê tổ hợp cơ bản đều có thể phát biểu dưới dạng bài toán liệt kê (Q).

Ví dụ:

- Bài toán liệt kê xâu nhị phân độ dài n dẫn về việc liệt kê các phần tử của tập

$$B^n = \{(a_1, ..., a_n): a_i \in \{0, 1\}, i=1, 2, ..., n\}.$$

- Bài toán liệt kê các tập con m phần tử của tập $N = \{1, 2, ..., n\}$ đòi hỏi liệt kê các phần tử của tập:

$$S(m,n) = \{(a_1,..., a_m) \in N^m: 1 \le a_1 < ... < a_m \le n \}.$$

- Tập các hoán vị của các số tự nhiên 1, 2, ..., n là tập

$$\Pi_n = \{(a_1,..., a_n) \in N^n: a_i \ne a_j ; i \ne j \}.$$

Lời giải bộ phận

Lời giải của bài toán là bộ có thứ tự gồm n thành phần $(a_1, a_2, ..., a_n)$, trong đó $a_i \in A_i$, $i = 1, 2, ..., n$.

Định nghĩa. Ta gọi lời giải bộ phận cấp k $(0 \le k \le n)$ là bộ có thứ tự gồm k thành phần

$$(a_1, a_2, ..., a_k),$$

trong đó $a_i \in A_i$, $i = 1, 2, ..., k$.

- Khi $k = 0$, lời giải bộ phận cấp 0 được ký hiệu là ( ) và còn được gọi là lời giải rỗng.
- Nếu $k = n$, ta có lời giải đầy đủ hay đơn giản là một lời giải của bài toán.

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sơ đồ thuật toán quay lui

Thuật toán quay lui được xây dựng dựa trên việc xây dựng dần từng thành phần của lời giải.

- Thuật toán bắt đầu từ lời giải rỗng ( ).
- Trên cơ sở tính chất P ta xác định được những phần tử nào của tập A_1 có thể chọn vào vị trí thứ nhất của lời giải. Những phần tử như vậy ta sẽ gọi là những ứng cử viên (viết tắt là UCV) vào vị trí thứ nhất của lời giải. Ký hiệu tập các UCV vào vị trí thứ nhất của lời giải là S_1. Lấy $a_1 \in S_1$, bổ sung nó vào lời giải rỗng đang có ta thu được lời giải bộ phận cấp 1: (a_1).

Bài toán liệt kê (Q): *Cho $A_1, A_2, ..., A_n$ là các tập hữu hạn. Ký hiệu*

$$A = A_1 \times A_2 \times ... \times A_n = \{ (a_1, a_2, ..., a_n): \ a_i \in A_i , \ i=1, 2, ..., n\}.$$

Giả sử P là tính chất cho trên A. Vấn đề đặt ra là liệt kê tất cả các phần tử của A thoả mãn tính chất P:

$$D = \{ a = (a_1, a_2, ..., a_n) \in A: \ a \text{ thoả mãn tính chất } P \}.$$

Sơ đồ thuật toán quay lui

- Bước tổng quát: Giả sử ta đang có lời giải bộ phận cấp k-1:

$$(a_1, a_2, ..., a_{k-1}),$$

cần xây dựng lời giải bộ phận cấp k:

$$(a_1, a_2, ..., a_{k-1}, \mathbf{a_k})$$

  - Trên cơ sở tính chất P, ta xác định được những phần tử nào của tập A_k có thể chọn vào vị trí thứ k của lời giải.
  - Những phần tử như vậy ta sẽ gọi là những ứng cử viên (viết tắt là UCV) vào vị trí thứ k của lời giải khi k-1 thành phần đầu của nó đã được chọn là $(a_1, a_2, ..., a_{k-1})$. Ký hiệu tập các ứng cử viên này là S_k.
  - Xét 2 tình huống:
    - $S_k \neq \varnothing$
    - $S_k = \varnothing$

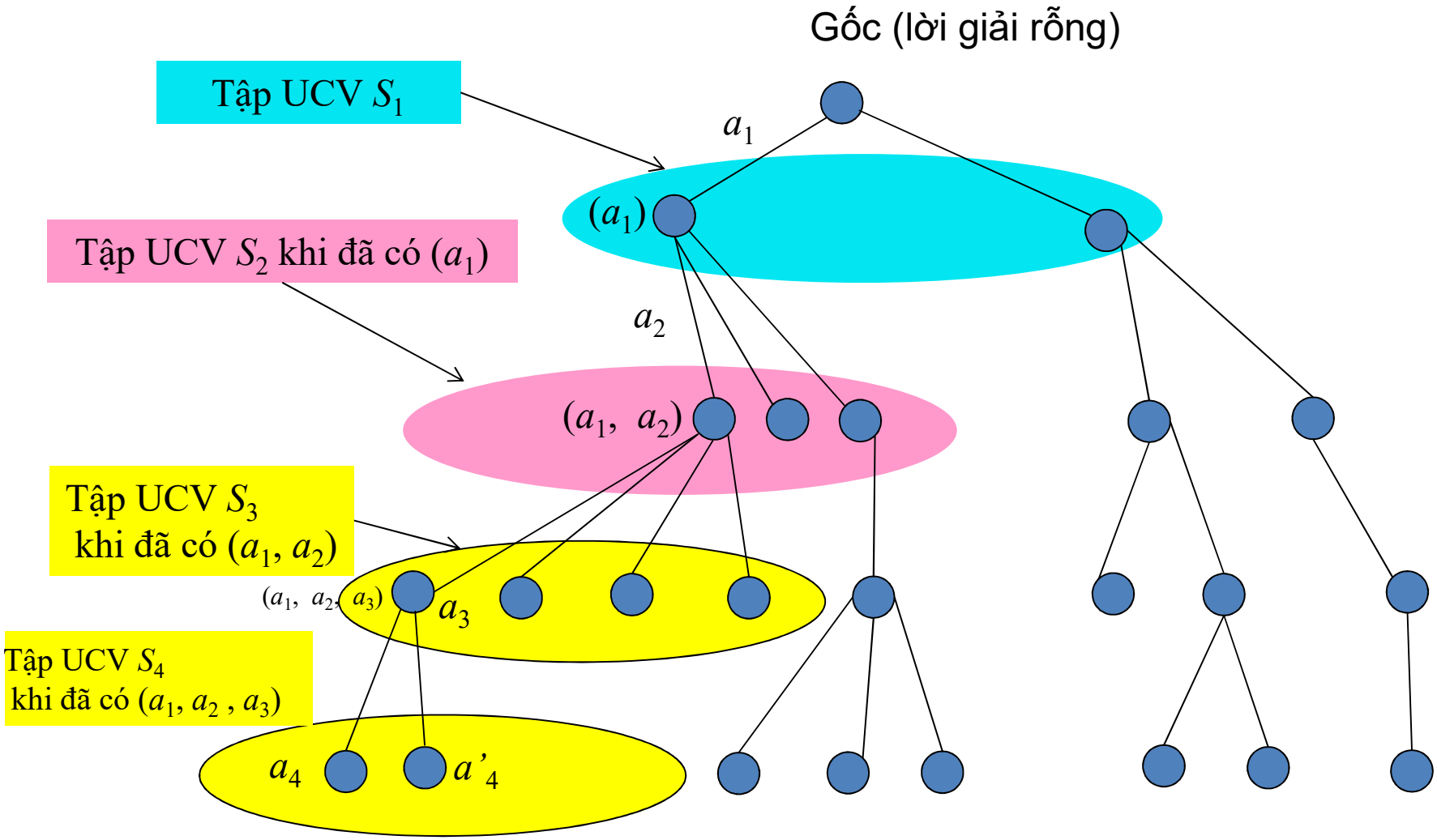
NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Sơ đồ thuật toán quay lui

- **$S_k \neq \varnothing$:** Lấy $a_k \in S_k$ bổ sung nó vào lời giải bộ phận cấp k-1 đang có (a_1, a_2, ..., a_{k-1}) ta thu được lời giải bộ phận cấp k (a_1, a_2, ..., a_{k-1}, a_k). Khi đó
  - Nếu $k = n$ thì ta thu được một lời giải,
  - Nếu $k < n$, ta tiếp tục đi xây dựng thành phần thứ k+1 của lời giải.
- **$S_k = \varnothing$:** Điều đó có nghĩa là lời giải bộ phận (a_1, a_2, ..., a_{k-1}) không thể tiếp tục phát triển thành lời giải đầy đủ. Trong tình huống này ta **quay trở lại** tìm ứng cử viên mới vào vị trí thứ k-1 của lời giải (chú ý: ứng cử viên mới này nằm trong tập S_{k-1})
  - Nếu tìm thấy UCV như vậy, thì bổ sung nó vào vị trí thứ k-1 rồi lại tiếp tục đi xây dựng thành phần thứ k.
  - Nếu không tìm được thì ta lại **quay trở lại** thêm một bước nữa tìm UCV mới vào vị trí thứ k-2, ... Nếu quay lại tận lời giải rỗng mà vẫn không tìm được UCV mới vào vị trí thứ 1, thì thuật toán kết thúc.

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Cây liệt kê lời giải theo thuật toán quay lui



Thuật toán quay lui (đệ qui)

```
void Try(int k)
{
   <Xây dựng Sk là tập chứa các ứng cử viên cho vị trí
     thứ k của lời giải>;
  for y ∈ Sk  //Với mỗi UCV y từ Sk
  {
     ak = y;
     if  (k == n) then<Ghi nhận lời giải (a1, a2, ..., ak) >;
     else Try(k+1);
     Trả các biến về trạng thái cũ;
  }
}
```

Lệnh gọi để thực hiện thuật toán quay lui là:

Try(1);

- Nếu chỉ cần tìm một lời giải thì cần tìm cách chấm dứt các thủ tục gọi đệ qui lồng nhau sinh bởi lệnh gọi Try(1) sau khi ghi nhận được lời giải đầu tiên.
- Nếu kết thúc thuật toán mà ta không thu được một lời giải nào thì điều đó có nghĩa là bài toán không có lời giải.

Thuật toán quay lui (không đệ qui)

```
void Backtracking ( )
{
 k=1;
 <Xây dựng Sk>;
 while (k > 0)  {
    while (Sk ≠ ∅ ) {
        ak ← Sk; // Lấy ak từ Sk
        if  <(k == n) >  then <Ghi nhận (a1,a2,...,ak) >;
        else {
              k = k+1;
              <Xây dựng Sk>;
        }
    }
     k = k - 1;   // Quay lui
  }
}
```

Lệnh gọi để thực hiện thuật toán quay lui là:

Backtracking ( );

Hai vấn đề mấu chốt

- Để cài đặt thuật toán quay lui giải các bài toán tổ hợp cụ thể ta cần giải quyết hai vấn đề cơ bản sau:
  - Tìm thuật toán xây dựng các tập UCV S_k
  - Tìm cách mô tả các tập này để có thể cài đặt thao tác liệt kê các phần tử của chúng (cài đặt vòng lặp qui ước **for $y \in S_k$).**
- Hiệu quả của thuật toán liệt kê phụ thuộc vào việc ta có xác định được chính xác các tập UCV này hay không.

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Chú ý

- Nếu độ dài của lời giải là không biết trước và các lời giải cũng không nhất thiết phải có cùng độ dài.

```
void Try(int k)
{
   <Xây dựng Sk là tập chứa các ứng cử viên cho vị trí
     thứ k của lời giải>;
  for y ∈ Sk  //Với mỗi UCV y từ Sk
  {
     ak = y;
     if  (k == n) then<Ghi nhận lời giải (a1, a2, ..., ak) >;
     else Try(k+1);
     Trả các biến về trạng thái cũ;
  }
}
```

- Khi đó chỉ cần sửa lại câu lệnh

  if ($k == n$) then <Ghi nhận lời giải (a_1, a_2, ..., a_k) >;

  else Try(k+1);

  thành

  if <(a_1, a_2, ..., a_k) là lời giải> then <Ghi nhận (a_1, a_2, ..., a_k) >;

  else Try(k+1);

→ Cần xây dựng hàm nhận biết (a_1, a_2, ..., a_k) đã là lời giải hay chưa.

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Thuật toán quay lui (Backtracking)

4.1. Sơ đồ thuật toán

4.2. Liệt kê các cấu hình tổ hợp cơ bản

- **Liệt kê xâu nhị phân độ dài *n***
- Liệt kê tập con *m* phần tử của tập *n* phần tử
- Liệt kê hoán vị

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Ví dụ 1: Liệt kê xâu nhị phân độ dài n

- Bài toán liệt kê xâu nhị phân độ dài n dẫn về việc liệt kê các phần tử của tập

$$B^n = \{(b_1, ..., b_n): b_i \in \{0, 1\}, i=1, 2, ..., n\}.$$

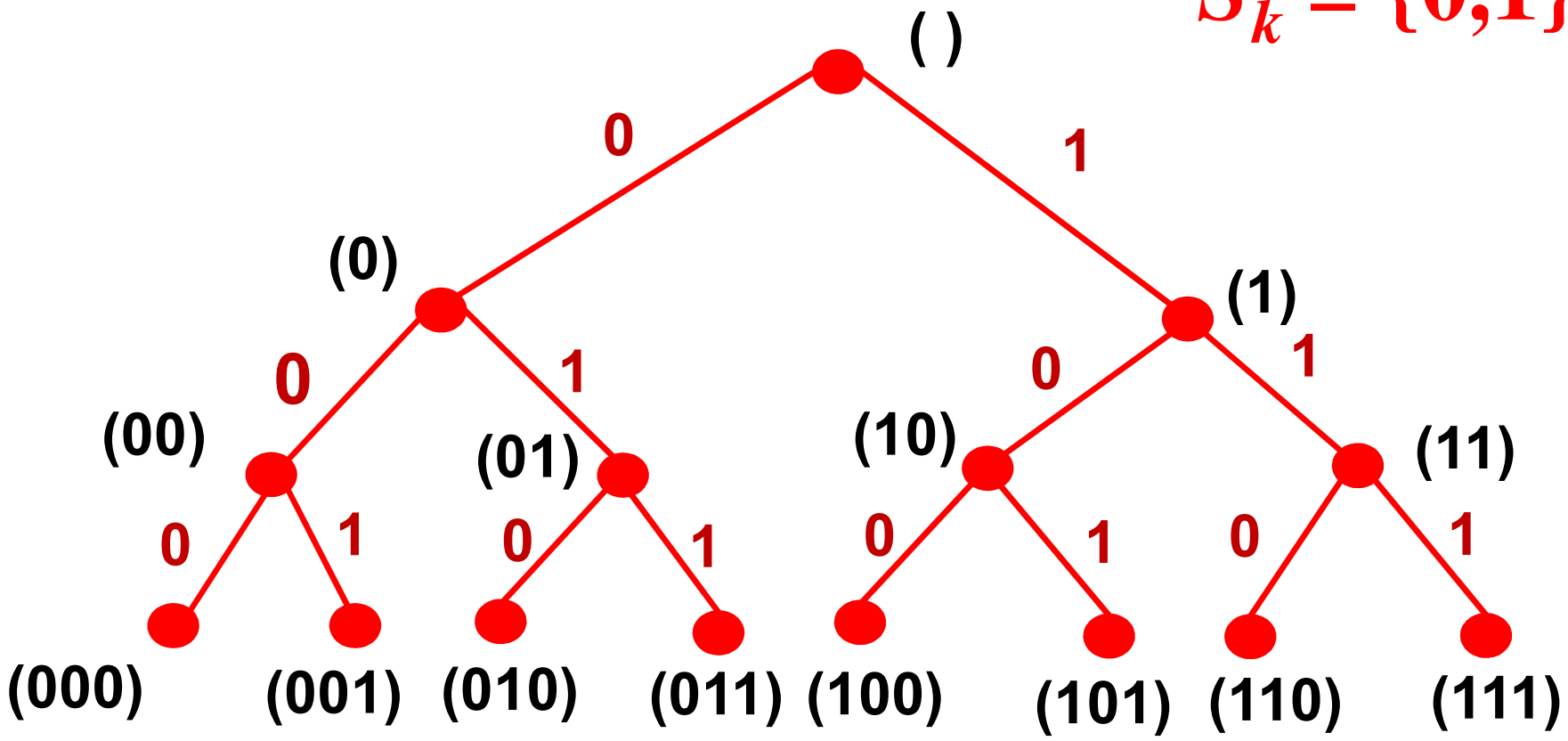
- Ta xét cách giải quyết hai vấn đề cơ bản để cài đặt thuật toán quay lui:
  - Xây dựng tập ứng cử viên S_k: Rõ ràng ta có $S_1 = \{0, 1\}$. Giả sử đã có xâu nhị phân cấp k-1 $(b_1, ..., b_{k-1})$, khi đó rõ ràng $S_k = \{0,1\}$. Như vậy, tập các UCV vào các vị trí của lời giải đã được xác định.
  - Cài đặt vòng lặp liệt kê các phần tử của S_k: ta có thể sử dụng vòng lặp for

```
for (y=0;y<=1;y++)
```

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Cây liệt kê dãy nhị phân độ dài 3

$S_k = \{0,1\}$



NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Chương trình trên C++ (Đệ qui)

```
#include <iostream>
using namespace std;
int  n, count;
int b[100];

void Ghinhan() {
   int i, j;
    count++;
    cout<<"Xau thu " << count<<": ";
    for  (i=1 ; i<= n ;i++) {
         j=b[i];
         cout<<j<<"   ";
    }
    cout<<endl;
}
```

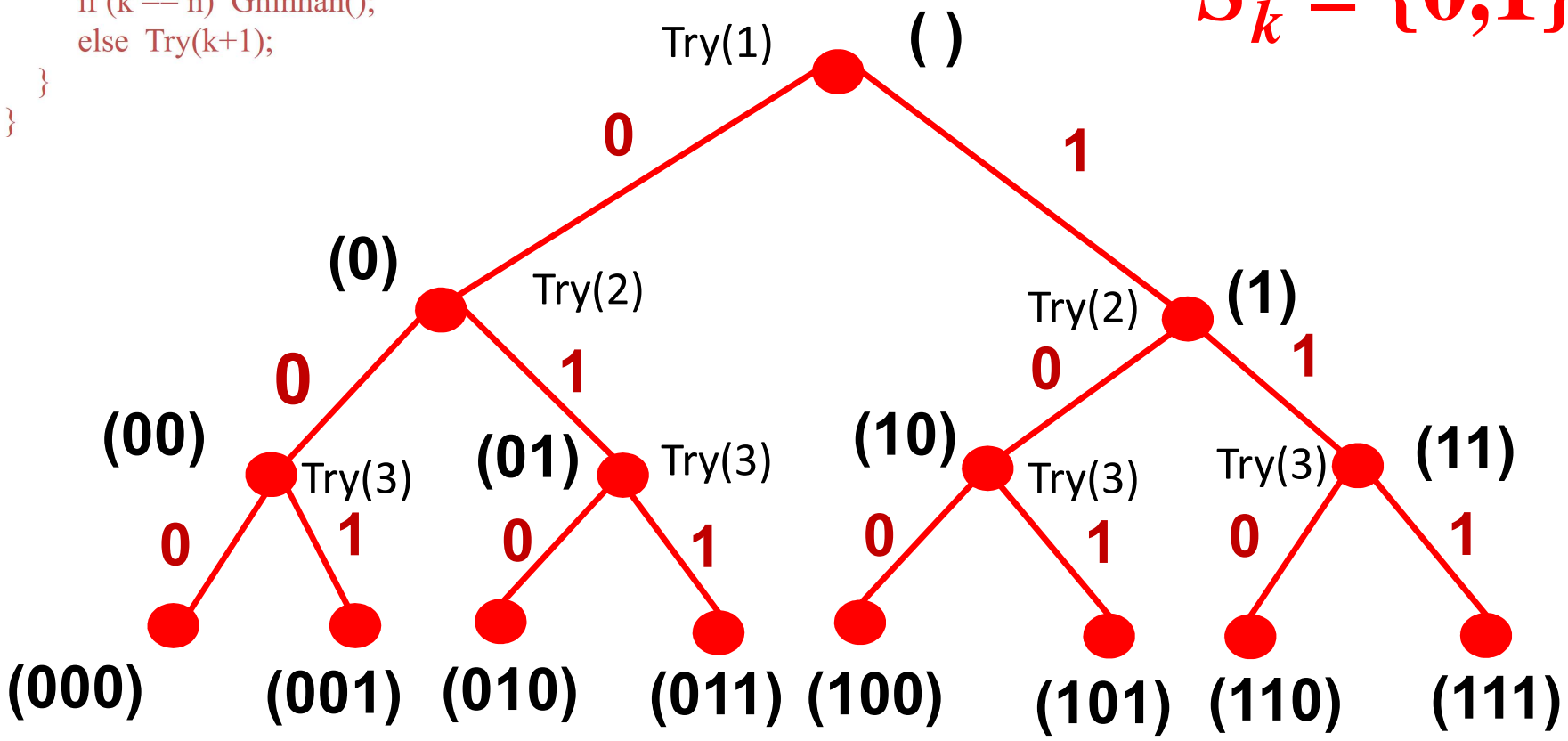
```
void  Try(int k){
   for (int j = 0; j<=1; j++) {
         b[k] = j;
         if (k == n)  Ghinhan();
         else  Try(k+1);
    }
}

int main() {
     cout<<"Nhap n = ";cin>>n;
     count = 0; Try(1);
     cout<<"So luong xau = "<<count<<endl;
}
```

Cây liệt kê dãy nhị phân độ dài 3

```
void  Try(int k){
  for (int j = 0; j<=1; j++) {
      b[k] = j;
      if (k == n)  Ghinhan();
      else  Try(k+1);
   }
}
```

$S_k = \{0,1\}$



NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Chương trình trên C++ (không đệ qui)

```
#include <iostream>
using namespace std;
int  n, count,k;
int b[100], s[100];

void Ghinhan() {
   int i, j;
    count++;
    cout<<"Xau thu " << count<<":
     ";
    for  (i=1 ; i<= n ;i++) {
         j=b[i];
         cout<<j<<"   ";
    }
    cout<<endl;
}
```

```
void  Xau( ) {
   k=1;  s[k]=0;
   while (k > 0) {
        while (s[k] <= 1) {
             b[k]=s[k];
             s[k]=s[k]+1;
             if (k==n) Ghinhan();
             else {
                       k++;
                       s[k]=0;
             }
        }
        k--;       // Quay lui
   }
}
```

Chương trình trên C++ (không đệ qui)

```
int main() {
    cout<<"Nhap n = ";cin>>n;
    count = 0; Xau();
    cout<<"So luong xau = "<<count<<endl;

}
```

Thuật toán quay lui (Backtracking)

4.1. Sơ đồ thuật toán

4.2. Liệt kê các cấu hình tổ hợp cơ bản

- Liệt kê xâu nhị phân độ dài *n*
- **Liệt kê tập con *m* phần tử của tập *n* phần tử**
- Liệt kê hoán vị

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Ví dụ 2. Liệt kê các *m*-tập con của *n*-tập

Bài toán: Liệt kê các tập con m phần tử của tập $N = \{1, 2, ..., n\}$.

Ví dụ: Liệt kê các tập con 3 phần tử của tập N = {1, 2, 3, 4, 5}

Giải: (1, 2, 3), (1, 2, 4), (1, 2, 5), (1, 3, 4), (1, 3, 5), (1, 4, 5), (2, 3, 4), (2, 3, 5), (2, 4, 5), (3, 4, 5)

➔ Bài toán dẫn về: Liệt kê các phần tử của tập:

$$S(m,n)=\{(a_1,..., a_m)\in N^m: 1 \le a_1<...<a_m\le n\}$$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Ví dụ 2. Liệt kê các *m*-tập con của *n*-tập

Ta xét cách giải quyết 2 vấn đề cơ bản để cài đặt thuật toán quay lui:

- Xây dựng tập ứng cử viên S_k:
  - Từ điều kiện: $1 \le a_1 < a_2 < ... < a_m \le n$

    suy ra $S_1 = \{1, 2, ..., n-(m-1)\}$.
  - Giả sử có tập con $(a_1, ..., a_{k-1})$. Từ điều kiện $a_{k-1} < a_k < ... < a_m \le n$, ta suy ra:

    $S_k = \{a_{k-1}+1, a_{k-1}+2, ..., n-(m-k)\}$.
- Cài đặt vòng lặp liệt kê các phần tử của S_k:

```
for (y=a[k-1]+1;y<=n-m+k;y++) ...
```

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Chương trình trên C++ (Đệ qui)

```
#include <iostream>
using namespace std;

int  n, m, count;
int a[100];
void Ghinhan() {
    int i;
    count++;
    cout<<"Tap con thu " <<count<<": ";
    for  (i=1 ; i<= m ;i++)
          cout<<a[i]<<"  ";
    cout<<endl;
}
```

```
void  Try(int k){
   int j;
   for (j = a[k-1] +1; j<= n-m+k; j++) {
        a[k] = j;
        if (k==m)  Ghinhan();
        else Try(k+1);
    }
}
int main() {
     cout<<"Nhap n, m = "; cin>>n; cin>>m;
     a[0]=0; count = 0; Try(1);
     cout<<"So tap con "<<m<<" phan tu cua tap
     "<<n<<" phan tu = "<<count<<endl;
}
```

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

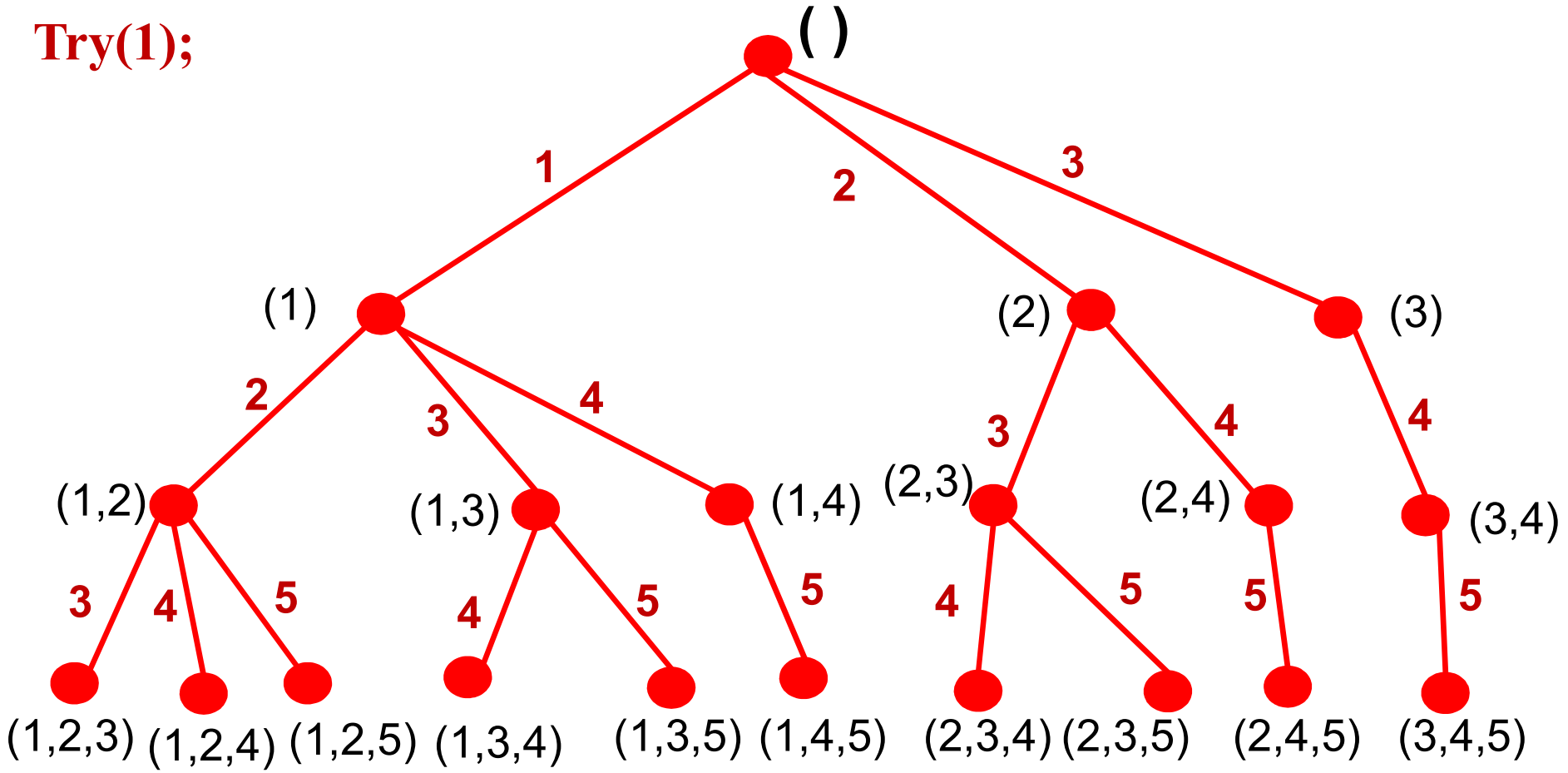
Chương trình trên C++ (không đệ qui)

```
#include <iostream>
using namespace std;

int  n, m, count,k;
int a[100], s[100];
void Ghinhan() {
    int i;
    count++;
    cout<<"Tap con thu " <<count<<":
     ";
    for  (i=1 ; i<= m ;i++)
          cout<<a[i]<<"  ";
    cout<<endl;
}
```

```
void  MSet(){
   k=1;  s[k]=1;
   while(k>0){
      while (s[k]<= n-m+k) {
         a[k]=s[k];  s[k]=s[k]+1;
         if (k==m) Ghinhan();
         else { k++; s[k]=a[k-1]+1;  }
      }  k--;
    }
}
int main() {
     cout<<"Nhap n, m = "; cin>>n; cin>>m;
     a[0]=0; count = 0; MSet();
     cout<<"So tap con "<<m<<" phan tu cua
     tap "<<n<<" phan tu = "<<count<<endl;
}
```

Cây liệt kê S(5,3)



$$S_k = \{a_{k-1}+1, a_{k-1}+2, ..., n-(m-k)\}$$

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Thuật toán quay lui (Backtracking)

4.1. Sơ đồ thuật toán

4.2. Liệt kê các cấu hình tổ hợp cơ bản

- Liệt kê xâu nhị phân độ dài *n*
- Liệt kê tập con *m* phần tử của tập *n* phần tử
- **Liệt kê hoán vị**

Ví dụ 3. Liệt kê hoán vị

Tập các hoán vị của các số tự nhiên 1, 2, ..., n là tập:

$$\Pi_n = \{(x_1,..., x_n) \in N^n: \ x_i \neq x_j \ , i \neq j \ \}.$$

Bài toán: Liệt kê tất cả các phần tử của Π_n

Ví dụ 3. Liệt kê hoán vị

- Xây dựng tập ứng cử viên S_k:
  - Rõ ràng $S_1 = N$. Giả sử ta đang có hoán vị bộ phận $(a_1, a_2, ..., a_{k-1})$, từ điều kiện $a_i \neq a_j$, với mọi $i \neq j$ ta suy ra

$$S_k = N \setminus \{ a_1, a_2, ..., a_{k-1} \}.$$

Mô tả S_k

Xây dựng hàm nhận biết UCV:

```
bool UCV(int j, int k)
{
   //UCV nhận giá trị true khi và chỉ khi j ∈ Sk
    int i;
    for (i=1;i++;i<=k-1)
        if (j == a[i]) return false;
    return true;
}
```

Liệt kê hoán vị

- Cài đặt vòng lặp liệt kê các phần tử của S_k:

```
for (y=1; y++; y <= n)
    if  (UCV(y, k) )
    {
            // y là UCV vào vị trí k
            . . .
     }
```

Chương trình trên C++

```
#include <iostream>
using namespace std;

int  n, m, count;
int a[100];

int Ghinhan() {
    int i, j;
    count++;
    cout<<"Hoan vi thu "<<count<<": ";
    for  (i=1 ; i<= n ;i++)
        cout<<a[i]<<"  ";
    cout<<endl;
}
```

```
bool UCV(int j, int k)
{
    int i;
    for (i=1; i<=k-1; i++)
       if (j == a[i]) return false;
    return true;
}
```

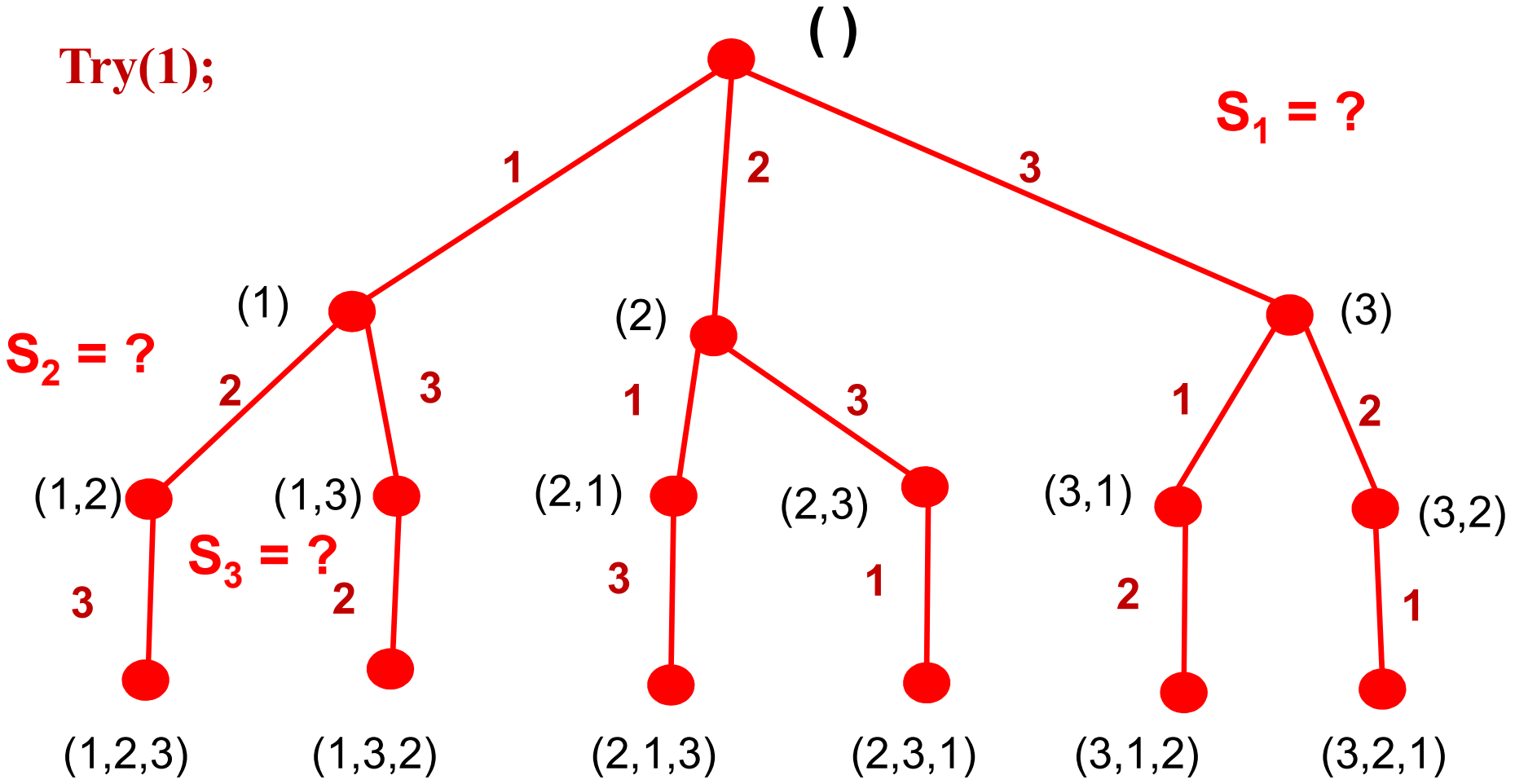
NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

```
void  Try(int k)
{
  int j;
  for (j = 1; j<=n; j++)
    if (UCV(j,k))
    {  a[k] = j;
       if (k==n)  Ghinhan( );
       else  Try(k+1);
    }
}
```

```
int main() {
    cout<<("Nhap n = "); cin>>n;
     count = 0; Try(1);
     cout<<"So hoan vi = " << count;
}
```

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

Cây liệt kê hoán vị của 1, 2, 3



$$S_k = N \setminus \{ a_1, a_2, ..., a_{k-1} \}$$