CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

CHƯƠNG 1: CÁC KHÁI NIỆM CƠ BẢN

NỘI DUNG

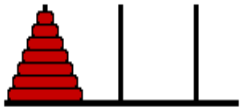
1.1. Ví dụ mở đầu

1.2. Thuật toán và độ phức tạp

1.3. Ký hiệu tiệm cận

1.4. Giả ngôn ngữ

Ví dụ mở đầu



- Bài toán tìm dãy con lớn nhất:

  Cho dãy số

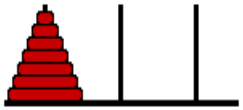
  $$a_1, a_2, \ldots, a_n$$

  Dãy số $a_i, a_{i+1}, \ldots, a_j$ với $1 \le i \le j \le n$ được gọi là **dãy con** của dãy đã cho và $\sum_{k=i}^{j} a_k$ được gọi là **trọng lượng** của dãy con này

  Bài toán đặt ra là: *Hãy tìm trọng lượng lớn nhất của các dãy con, tức là tìm cực đại giá trị* $\sum_{k=i}^{j} a_k$. Để đơn giản ta gọi dãy con có trọng lượng lớn nhất là ***dãy con lớn nhất***.

- **Ví dụ:** Nếu dãy đã cho là -2, **11, -4, 13**, -5, 2 thì cần đưa ra câu trả lời là 20 (là trọng lượng của dãy con 11, -4, 13)

Thuật toán trực tiếp



- Thuật toán đơn giản đầu tiên có thể nghĩ để giải bài toán đặt ra là: Duyệt tất cả các dãy con có thể

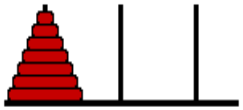
$$a_i, a_{i+1}, \ldots, a_j \quad \text{với} \quad 1 \le i \le j \le n$$

và tính tổng của mỗi dãy con để tìm ra trọng lượng lớn nhất.

- Trước hết nhận thấy rằng, tổng số các dãy con có thể của dãy đã cho là

$$C(n,2) + n = n^2/2 + n/2 .$$

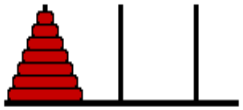
Thuật toán trực tiếp



- Thuật toán này có thể cài đặt trong đoạn chương trình sau:

```
int maxSum = 0;
for (int i=0; i<n; i++) {
   for (int j=i; j<n; j++) {
      int sum = 0;
      for (int k=i; k<=j; k++)
          sum += a[k];
      if sum > maxSum
          maxSum = sum;
   }
}
```

Thuật toán trực tiếp



- **Phân tích thuật toán:** Ta sẽ tính số lượng phép cộng mà thuật toán phải thực hiện, tức là đếm xem dòng lệnh

  `Sum += a[k]`

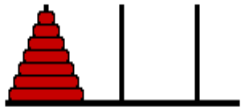
  phải thực hiện bao nhiêu lần. Số lượng phép cộng sẽ là

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}(j-i+1)=\sum_{i=0}^{n-1}(1+2+\ldots+(n-i))=\sum_{i=0}^{n-1}\frac{(n-i)(n-i+1)}{2}$$

$$=\frac{1}{2}\sum_{k=1}^{n}k(k+1)=\frac{1}{2}\left[\sum_{k=1}^{n}k^2+\sum_{k=1}^{n}k\right]=\frac{1}{2}\left[\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}\right]$$

$$=\frac{n^3}{6}+\frac{n^2}{2}+\frac{n}{3}$$

Thuật toán nhanh hơn

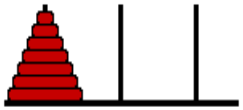


- Để ý rằng tổng các số hạng từ i đến j có thể thu được từ tổng của các số hạng từ i đến j-1 bởi 1 phép cộng, cụ thể là ta có:

$$\sum_{k=i}^{j} a[k] = a[j] + \sum_{k=i}^{j-1} a[k]$$

- Nhận xét này cho phép rút bớt vòng lặp for trong cùng.

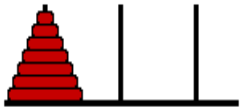
Thuật toán nhanh hơn



- Ta có thể cài đặt như sau

```
int maxSum = a[0];
for (int i=0; i<n; i++) {
   int sum = 0;
   for (int j=i; j<n; j++) {
      sum += a[j];
      if sum > maxSum
          maxSum = sum;
   }
}
```

Thuật toán nhanh hơn

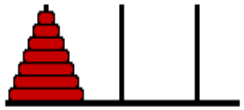


- **Phân tích thuật toán.** Ta lại tính số lần thực hiện phép cộng và thu được kết quả sau:

$$\sum_{i=0}^{n-1}(n-i) = n + (n-1) + \ldots + 1 = \frac{n^2}{2} + \frac{n}{2}$$

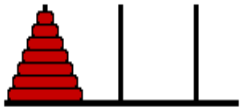
- Để ý rằng số này là đúng bằng số lượng dãy con. Dường như thuật toán thu được là rất tốt, vì ta phải xét mỗi dãy con đúng 1 lần.

Thuật toán đệ qui



- Ta còn có thể xây dựng thuật toán tốt hơn nữa! Ta sẽ sử dụng kỹ thuật chia để trị. Kỹ thuật này bao gồm các bước sau:
  - Chia bài toán cần giải ra thành các bài toán con cùng dạng
  - Giải mỗi bài toán con một cách đệ qui
  - Tổ hợp lời giải của các bài toán con để thu được lời giải của bài toán xuất phát.
- Áp dụng kỹ thuật này đối với bài toán tìm trọng lượng lớn nhất của các dãy con: Ta chia dãy đã cho ra thành 2 dãy sử dụng phần tử ở chính giữa và thu được 2 dãy số (gọi tắt là dãy bên trái và dãy bên phải) với độ dài giảm đi một nửa.

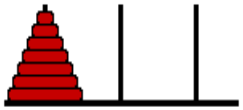
Thuật toán đệ qui



- Để tổ hợp lời giải, nhận thấy rằng chỉ có thể xảy ra một trong 3 trường hợp:
  - Dãy con lớn nhất nằm ở dãy con bên trái (nửa trái)
  - Dãy con lớn nhất nằm ở dãy con bên phải (nửa phải)
  - Dãy con lớn nhất bắt đầu ở nửa trái và kết thúc ở nửa phải (giữa).
- Do đó, nếu ký hiệu trọng lượng của dãy con lớn nhất ở nửa trái là w_L, ở nửa phải là w_R và ở giữa là w_M thì **trọng lượng cần tìm sẽ là**

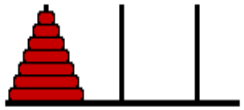
$$\mathbf{max(w_L, w_R, w_M).}$$

Thuật toán đệ qui

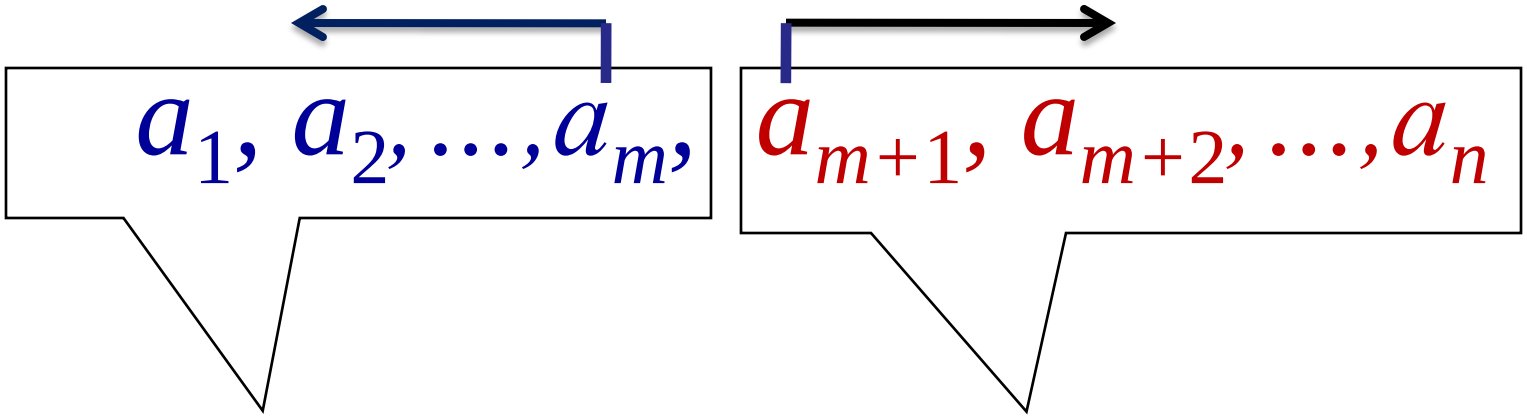


- Việc tìm trọng lượng của dãy con lớn nhất ở nửa trái (w_L) và nửa phải (w_R) có thể thực hiện một cách đệ qui
- Để tìm trọng lượng $\mathbf{w_M}$ của dãy con lớn nhất bắt đầu ở nửa trái và kết thúc ở nửa phải ta thực hiện như sau:
  - Tính trọng lượng của dãy con lớn nhất trong nửa trái kết thúc ở điểm chia (w_{ML}) và
  - Tính trọng lượng của dãy con lớn nhất trong nửa phải bắt đầu ở điểm chia (w_{MR}).
  - Khi đó $w_M = w_{ML} + w_{MR}$.

Thuật toán đệ qui



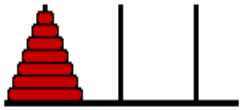
- m – điểm chia của dãy trái, m+1 là điểm chia của dãy phải



Tính W_{ML} của dãy con lớn nhất trong nửa trái kết thúc tại a_m

Tính W_{MR} của dãy con lớn nhất trong nửa phải bắt đầu từ a_{m+1}

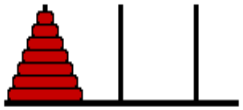
Thuật toán đệ qui



- **Để tính trọng lượng của dãy con lớn nhất ở nửa trái (từ a[i] đến a[j]) kết thúc ở a[j] ta dùng thuật toán sau:**

```
MaxLeft(a, i, j);
{
  maxSum = -∞; sum = 0;
  for (int k=j; k>=i; k--) {
       sum = sum+a[k];
       maxSum = max(sum, maxSum);
   }
    return maxSum;
}
```

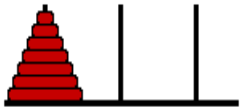
Thuật toán đệ qui



- **Để tính trọng lượng của dãy con lớn nhất ở nửa phải (từ a[i] đến a[j]) bắt đầu từ a[i] ta dùng thuật toán sau:**

```
MaxRight(a, i, j);
{
     maxSum = -∞; sum = 0;
     for (int k=i; k<=j; k++){
       sum = sum+a[k];
       maxSum = max(sum, maxSum);
   }
   return maxSum;
}
```

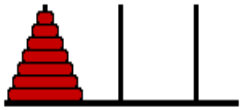
Thuật toán đệ qui



Sơ đồ của thuật toán đệ qui có thể mô tả như sau:

```
MaxSub(a, i, j);
{
  if (i = j) return a[i]
  else
  {   m = (i+j)/2;
      wL = MaxSub(a, i, m);
      wR = MaxSub(a, m+1, j);
      wM = MaxLeft(a, i, m)+
           MaxRight(a, m+1, j);
      return max(wL, wR, wM);
  }
}
```

Thuật toán đệ qui



- **Phân tích thuật toán:**

  Ta cần tính xem lệnh gọi MaxSub(a,1,n) để thực hiện thuật toán đòi hỏi bao nhiêu phép cộng?

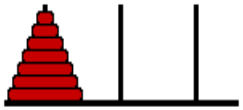
- Trước hết nhận thấy MaxLeft và MaxRight đòi hỏi

  $n/2 + n/2 = n$ phép cộng

- Vì vậy, nếu gọi $T(n)$ là số phép cộng cần tìm, ta có công thức đệ qui sau:

$$T(n) = \begin{cases} 0 & n = 1 \\ T(\frac{n}{2}) + T(\frac{n}{2}) + n = 2T(\frac{n}{2}) + n & n > 1 \end{cases}$$

Thuật toán đệ qui



- Ta khẳng định rằng $T(2^k) = k.2^k$. Ta chứng minh bằng qui nạp
- **Cơ sở qui nạp:** Nếu k=0 thì $T(2^0) = T(1) = 0 = 0.2^0$.
- **Chuyển qui nạp:** Nếu k>0, giả sử rằng $T(2^{k-1}) = (k-1)2^{k-1}$ là đúng. Khi đó

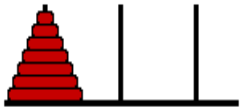
$$T(2^k) = 2T(2^{k-1})+2^k = 2(k-1).2^{k-1} + 2^k = k.2^k.$$

- Quay lại với ký hiệu n, ta có

$$T(n) = n \log n .$$

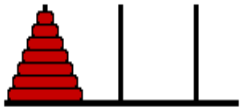
- **Kết quả thu được là tốt hơn thuật toán thứ hai !**

So sánh các thuật toán



- Cùng một bài toán ta đã đề xuất 3 thuật toán đòi hỏi số lượng phép toán khác nhau và vì thế sẽ đòi hỏi thời gian tính khác nhau.

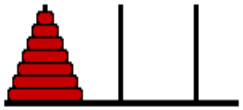
Thuật toán Quy hoạch động



Việc phát triển thuật toán dựa trên DP bao gồm 3 giai đoạn:

1. Phân rã: Chia bài toán cần giải thành những bài toán con nhỏ hơn có cùng dạng với bài toán ban đầu.
2. Ghi nhận lời giải: Lưu trữ lời giải của các bài toán con vào một bảng.
3. Tổng hợp lời giải: Lần lượt từ lời giải của các bài toán con kích thước nhỏ hơn tìm cách xây dựng lời giải của bài toán kích thước lớn hơn, cho đến khi thu được lời giải của bài toán xuất phát (là bài toán con có kích thước lớn nhất).

Thuật toán QHĐ



- **Phân rã.** Gọi s_i là trọng lượng của dãy con lớn nhất trong dãy $a_1, a_2, ..., a_i$, $i = 1, 2, ..., n$.

  Rõ ràng s_n là giá trị cần tìm.

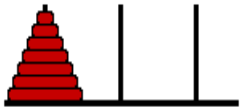
- **Tổng hợp lời giải.**
  - Trước hết, ta có

    $$s_1 = a_1.$$

  - Giả sử $i > 1$ và s_k là đã biết với $k = 1, 2, ..., i\text{-}1$. Ta cần tính s_i là trọng lượng của dãy con lớn nhất của dãy

    $$a_1, a_2, \ldots, a_{i\text{-}1}, a_i \ .$$

Thuật toán QHĐ

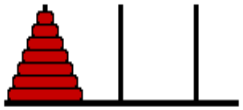


- Do dãy con lớn nhất của dãy này hoặc là có chứa phần tử a_i *hoặc là không chứa phần tử* a_i, *nên nó chỉ có thể là một trong hai dãy:*
  - Dãy con lớn nhất của dãy $a_1, a_2, ..., a_{i-1}$
  - Dãy con lớn nhất của dãy $a_1, a_2, ..., a_i$ *kết thúc tại* a_i.
- Từ đó suy ra

  $s_i = max \{s_{i-1}, e_i\}, i = 2, ..., n.$

  trong đó e_i *là trọng lượng của dãy con lớn nhất của dãy* $a_1, a_2, ..., a_i$ *kết thúc tại* a_i.
- Để tính e_i, *ta cũng có thể sử dụng công thức đệ qui sau:*
  - $e_1 = a_1$;
  - $e_i = max \{a_i, e_{i-1} + a_i\}, i = 2, ..., n.$

Thuật toán QHĐ

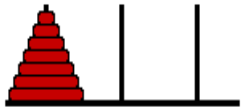


```
MaxSub(a);
{
    smax = a[1];            (* smax – trọng lượng của dãy con lớn nhất  *)
    maxendhere = a[1];      (* maxendhere – trọng lượng của dãy con lớn nhất kết thúc tại a[i] *)
    imax = 1;               (* imax - vị trí kết thúc của dãy con lớn nhất  *)
    for i = 2 to n {
            u = maxendhere + a[i];
            v = a[i];
            if (u > v) maxendhere = u
             else maxendhere = v;
            if (maxendhere > smax)then {
                smax := maxendhere;
                imax := i;
            }
    }
}
```

Phân tích thuật toán:

Dễ thấy số phép toán cộng phải thực hiện trong thuật toán (số lần thực hiện cấu lệnh **u = maxendhere + a[i];**) là *n*.

NỘI DUNG



1.1. Ví dụ mở đầu

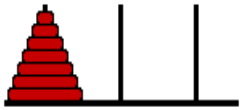
1.2. Thuật toán và độ phức tạp

1.3. Ký hiệu tiệm cận

1.4. Giả ngôn ngữ

Tham khảo tài liệu của PGS. TS. Nguyễn Đức Nghĩa

Khái niệm bài toán tính toán

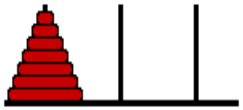


- **Định nghĩa.** *Bài toán tính toán F* là ánh xạ từ tập các xâu nhị phân độ dài hữu hạn vào tập các xâu nhị phân độ dài hữu hạn:

$$F : \{0, 1\}^* \rightarrow \{0, 1\}^*.$$

- **Ví dụ:**
  - Mỗi số nguyên x đều có thể biểu diễn d- ới dạng xâu nhị phân là cách viết trong hệ đếm nhị phân của nó.
  - Hệ ph- ơng trình tuyến tính $Ax = b$ có thể biểu diễn d- ới dạng xâu là ghép nối của các xâu biểu diễn nhị phân của các thành phần của ma trận A và vectơ b.
  - Đa thức một biến $P(x) = a_0 + a_1 x + ... + a_n x^n$ hoàn toàn xác định bởi dãy số $n, a_0, a_1, ..., a_n,$ mà để biểu diễn dãy số này chúng ta có thể sử dụng xâu nhị phân.

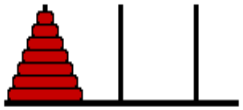
Khái niệm thuật toán



- **Định nghĩa.** *Ta hiểu thuật toán giải bài toán đặt ra là một thủ tục xác định bao gồm một dãy hữu hạn các b- ớc cần thực hiện để* ***thu đ ợc đầu ra*** *cho* ***một đầu vào cho tr ớc*** *của bài toán.*
- *Thuật toán có các đặc tr- ng 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 đ- a ra các dữ liệu t- ơng ứng với lời giải của bài toán.*
  - ***Chính xác (Precision):*** *Các b- ớc của thuật toán đ- ợc mô tả chính xác.*
  - ***Hữu hạn (Finiteness):*** *Thuật toán cần phải đ- a đ- ợc đầu ra sau một số hữu hạn (có thể rất lớn) b- ớc với mọi đầu vào.*
  - ***Đơn trị (Uniqueness):*** *Các kết quả trung gian của từng b- ớc thực hiện thuật toán đ- ợc xác định một cách đơn trị và chỉ phụ thuộc vào đầu vào và các kết quả của các b- ớc tr- ớc.*
  - ***Tổng quát (Generality):*** *Thuật toán có thể áp dụng để giải mọi bài toán có dạng đã cho.*

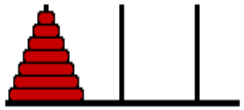
Giải bài toán là gì?

What is Problem Solving?



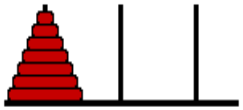
- Problem solving
  - Là quá trình đặt bài toán và phát triển chương trình máy tính để giải bài toán đặt ra
- Lời giải bài toán bao gồm:
  - Thuật toán (Algorithms)
    - *Algorithm: là dãy các bước cần thực hiện để từ dữ liệu vào (input) đưa ra kết quả đầu ra (output) của bài toán trong thời gian hữu hạn.*
  - Cấu trúc dữ liệu:
    - *Cách tổ chức lưu trữ dữ liệu vào - ra*

Độ phức tạp của thuật toán



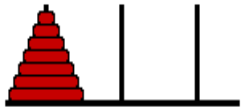
- *Đánh giá **độ phức tạp tính toán** của thuật toán là đánh giá **lượng tài nguyên các loại** mà thuật toán đòi hỏi sử dụng*. Có hai loại tài nguyên quan trọng đó là **thời gian** và **bộ nhớ**. Trong giáo trình này ta đặc biệt quan tâm đến đánh giá thời gian cần thiết để thực hiện thuật toán mà ta sẽ gọi là *thời gian tính* của thuật toán.
- Thời gian tính phụ thuộc vào dữ liệu vào.
- **Định nghĩa.** *Ta gọi kích th- ớc dữ liệu đầu vào* (*hay độ dài dữ liệu vào*) *là số bít cần thiết để biểu diễn nó*.
- Ta sẽ tìm cách đánh giá thời gian tính của thuật toán bởi một ***hàm của độ dài dữ liệu vào.***

Phép toán cơ bản



- Đo thời gian tính bằng đơn vị đo nào?
- **Định nghĩa.** *Ta gọi phép toán cơ bản là phép toán có thể thực hiện với thời gian bị chặn bởi một hằng số không phụ thuộc vào kích th- ớc dữ liệu.*
- Để tính toán thời gian tính của thuật toán ta sẽ đếm ***số phép toán cơ bản*** mà nó phải thực hiện.

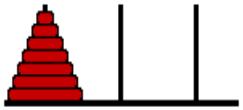
Các loại thời gian tính



Chúng ta sẽ quan tâm đến

- Thời gian tối thiểu cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích th- ớc n. Thời gian nh- vậy sẽ đ- ợc gọi là *thời gian tính tốt nhất* của thuật toán với đầu vào kích th- ớc n.
- Thời gian nhiều nhất cần thiết để thực hiện thuật toán với mọi bộ dữ liệu đầu vào kích th- ớc n. Thời gian nh- vậy sẽ đ- ợc gọi là *thời gian tính tồi nhất* của thuật toán với đầu vào kích th- ớc n.
- Thời gian trung bình cần thiết để thực hiện thuật toán trên tập hữu hạn các đầu vào kích th- ớc n. Thời gian nh- vậy sẽ đ- ợc gọi là *thời gian tính trung bình* của thuật toán.

NỘI DUNG



1.1. Ví dụ mở đầu

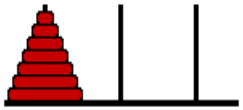
1.2. Thuật toán và độ phức tạp

1.3. Ký hiệu tiệm cận

1.4. Giả ngôn ngữ

Ký hiệu tiệm cận

Asymptotic Notation



Θ, Ω, O

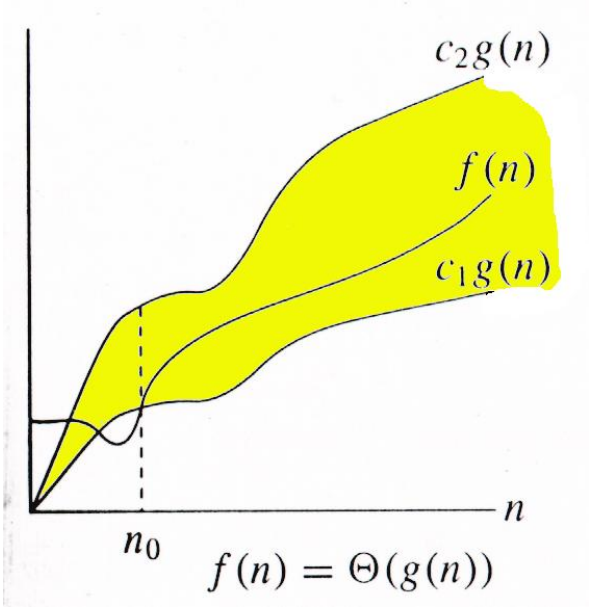
- Được sử dụng để mô tả thời gian tính của thuật toán
- Thay vì nói chính xác thời gian tính, ta nói $\Theta(n^2)$
- Được xác định đối với các hàm nhận giá trị nguyên không âm
- Dùng để so sánh tốc độ tăng của hai hàm

Ký hiệu Θ

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

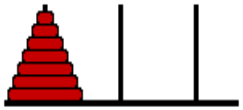
$\Theta(g(n)) = \{f(n)$: tồn tại các hằng số c_1, c_2 và n_0 sao cho

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



Ta nói rằng $g(n)$ là đánh giá tiệm cận đúng cho $f(n)$

Ví dụ



$$10n^2 - 3n = \Theta(n^2) \ ?$$

- Với giá trị nào của các hằng số n_0, c_1, và c_2 thì bất đẳng thức trong định nghĩa là đúng?
- Lấy c_1 bé hơn hệ số của số hạng với số mũ cao nhất, còn c_2 lấy lớn hơn, ta có

$$n^2 \leq 10n^2 - 3n \leq 11n^2, \textit{ với mọi } n \geq 1.$$

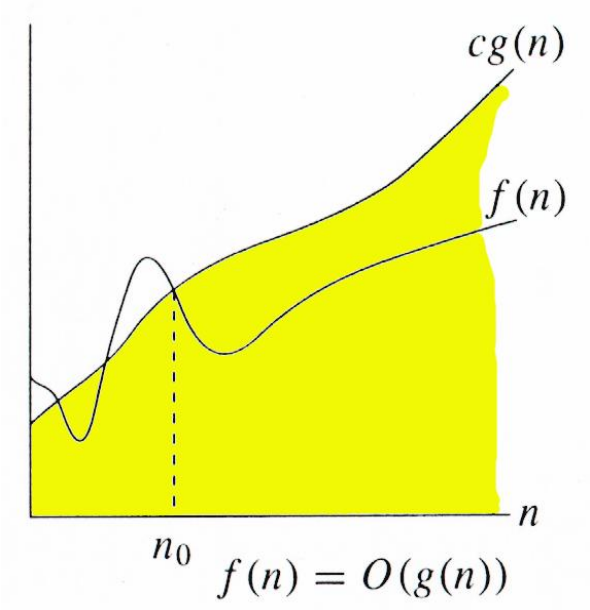
- *Đối với hàm đa thức: Để so sánh tốc độ tăng cần nhìn vào số hạng với số mũ cao nhất*

Ký hiệu O (*đọc là ô lớn - big O*)

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:

$f(n) \leq cg(n)$ với mọi $n \geq n_0 \}$



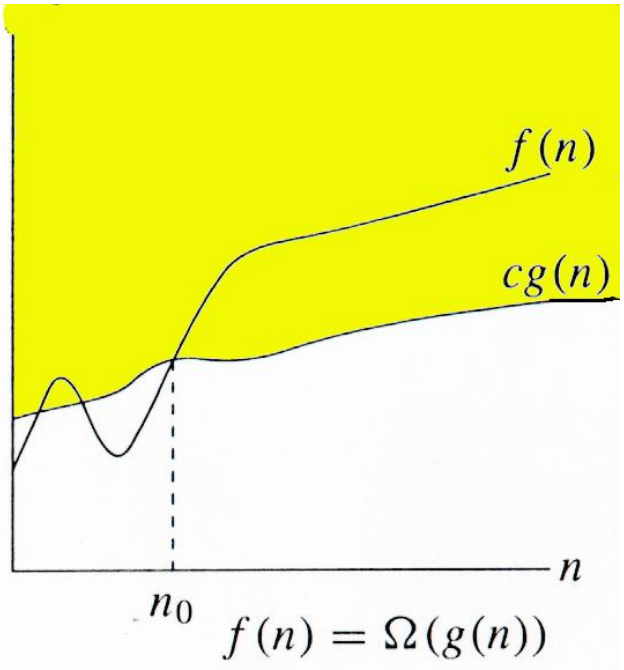
Ta nói $g(n)$ là *cận trên tiệm cận* của $f(n)$

Ký hiệu Ω

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

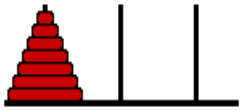
$\Omega(g(n)) = \{f(n)$: tồn tại các hằng số dương c và n_0 sao cho:

$cg(n) \leq f(n)$ với mọi $n \geq n_0 \}$



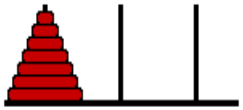
Ta nói $g(n)$ là *cận dưới tiệm cận* cho $f(n)$

Cách nói về thời gian tính



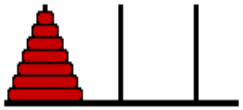
- Nói "Thời gian tính là $O(f(n))$" hiểu là: Đánh giá trong tình huống tồi nhất (worst case) là $O(f(n))$. Thường nói: "Đánh giá thời gian tính trong tình huống tồi nhất là $O(f(n))$"
  - *Nghĩa là thời gian tính trong tình huống tồi nhất được xác định bởi một hàm nào đó* $g(n) \in O(f(n))$
- "Thời gian tính là $\Omega(f(n))$" hiểu là: Đánh giá trong tình huống tốt nhất (best case) là $\Omega(f(n))$. 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* $g(n) \in \Omega(f(n))$

Một số lớp thuật toán



- **Một số lớp thuật toán đặc biệt:**
  - $O(1)$: hằng số (constant)
  - $O(\log n)$: logarithmic
  - $O(n)$: tuyến tính (linear)
  - $O(n \log n)$: trên tuyến tính (superlinear)
  - $O(n^2)$: bình phương (quadratic)
  - $O(n^3)$: bậc ba (cubic)
  - $O(a^n)$: hàm mũ (exponential) ($a > 1$)
  - $O(n^k)$: đa thức (polynomial) ($k \geq 1$)

NỘI DUNG



1.1. Ví dụ mở đầu

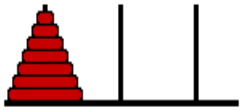
1.2. Thuật toán và độ phức tạp

1.3. Ký hiệu tiệm cận

1.4. Giả ngôn ngữ

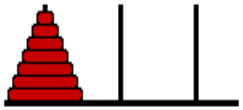
Tham khảo tài liệu của PGS. TS. Nguyễn Đức Nghĩa

Mô tả thuật toán: giả ngôn ngữ



- Để mô tả thuật toán có thể sử dụng một ngôn ngữ lập trình nào đó. Tuy nhiên điều đó có thể làm cho việc mô tả thuật toán trở nên phức tạp đồng thời rất khó nắm bắt.
- Vì thế, để mô tả thuật toán người ta thường sử dụng ***giả ngôn ngữ*** (***pseudo language***), trong đó cho phép vừa mô tả thuật toán bằng ngôn ngữ đời thường vừa sử dụng những cấu trúc lệnh tương tự như của ngôn ngữ lập trình.
- Dưới đây ta liệt kê một số câu lệnh chính được sử dụng trong giáo trình để mô tả thuật toán.

Mô tả thuật toán: phỏng ngôn ngữ



- **Khai báo biến**

  integer x,y;

  real u, v;

  boolean a, b;

  char c, d;

  datatype x;

- **Câu lệnh gán**

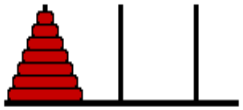
  x = expression;

  hoặc

  x ← expression;

  Ví dụ: x ← 1+4; y=a*y+2;

Mô tả thuật toán: giả ngôn ngữ

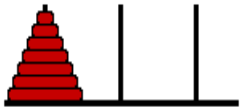


- **Cấu trúc điều khiển**

if condition **then**
    dãy câu lệnh
else
    dãy câu lệnh
endif;

while condition **do**
    dãy câu lệnh
endwhile;

Mô tả thuật toán: phỏng ngôn ngữ



repeat
dãy câu lệnh;
until condition;

for i=n1 **to** n2 [step d]
dãy câu lệnh;
endfor;

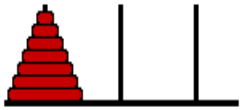
- **Vào-Ra**

read(X); /* X là biến đơn hoặc mảng */
print(data**) hoặc print(**thông báo**)**

Câu lệnh case:

Case
cond1: stat1;
cond2: stat2;
.
.
.
condn: stat n;
endcase;

Mô tả thuật toán: giả ngôn ngữ



- **Hàm và thủ tục (Function and procedure)**

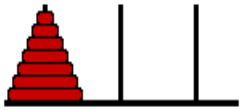
Function name(các tham số)
begin
    mô tả biến;
    các câu lệnh trong thân của hàm;
    return (giá trị)
end;

Procedure name(các tham số)
begin
    mô tả biến;
    các câu lệnh trong thân của hàm;
end;

Truyền tham số:

- Tham trị
- Tham biến
- Biến cục bộ
- Biến toàn cục

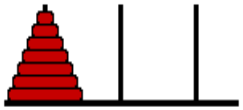
Mô tả thuật toán: phỏng ngôn ngữ



- **Ví dụ:** Thuật toán tìm phần tử lớn nhất trong mảng A(1:n)

```
Function max(A(1:n))
begin
   datatype x;  /* để giữ giá trị lớn nhất tìm được */
   integer i;
   x=A[1];
   for i=2 to n do
       if x < A[i] then
           x=A[i];
       endif
   endfor ;
  return (x);
end max;
```

Mô tả thuật toán: giả ngôn ngữ



- ***Ví dụ:*** Thuật toán hoán đổi nội dung hai biến

```
Procedure swap(x, y)
begin
    temp=x;
    x = y;
    y = temp;
end swap;
```