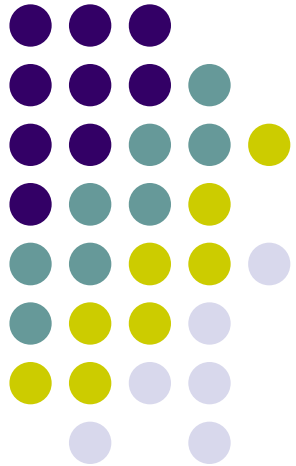
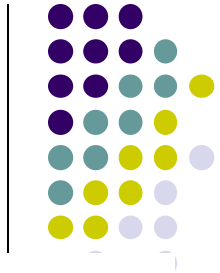
Kỹ thuật phân tích và thiết kế thuật toán

1. **Độ tăng của hàm**
2. **Thuật toán và độ phức tạp**
3. **Các phương pháp thiết kế thuật toán**
4. **Một số ví dụ**



Độ tăng của hàm



Hàm số f: R→R, g: R→R, x → +∞

Def.

f(x) là O-lớn của g(x) khi x→∞,

kí hiệu f(x) = O(g(x)), hoặc f(x) là O(g(x)),

nếu ∃ C >0, N >0

sao cho ∀ x >N:

$$|f(x)| \leq C.|g(x)|$$

Ví dụ 1:

$f(x) = x^2+2x+3$

$g(x) = x^2$

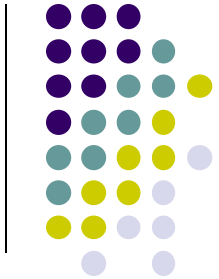
Chọn: C = 1, N = 1

$\forall\ x>1:\ f(x) \leq x^2 + 2x^2 + 3x^2 = 6x^2 \Rightarrow f(x) = O(x^2)$.

C = 1, N = 0

$\forall\ x>0:\ x^2 < f(x) \Rightarrow x^2 = O(f(x))$

Độ tăng của hàm



Ví dụ 2:

$f(x) = kx^2$

$g(x) = x^3$

Chọn: C = 1, N = |k|

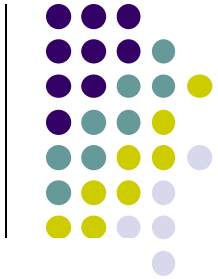
$\forall x > |k|$: $f(x) = kx^2 \leq x.x^2 = 1.x^3 \Rightarrow kx^2$ là $O(x^3)$

Ví dụ 3:

1/(1+x2) = O(1)

sin(x) = O(1)

Độ tăng của hàm



Mệnh đề 1.1. $f(x) = a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n = O(x^n)$

Chứng minh:

Kí hiệu $C = |a_0| + |a_1| + |a_2| + ... + |a_{n-1}| + |a_n|$

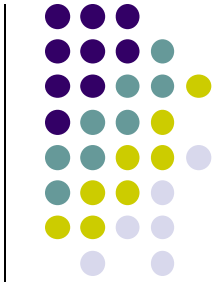
$N = 1$

Với $x > N=1$ ta có $x^k < x^n$, với $k< n$,

suy ra

$$
\begin{aligned}
|f(x)| &< |a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n| \\
&\le |a_0| + |a_1x| + |a_2x^2| + ... + |a_{n-1}x^{n-1}| + |a_nx^n| \\
&= |a_0| + |a_1|x + |a_2|x^2 + ... + |a_{n-1}|x^{n-1} + |a_n|\, x^n \\
&\le |a_0|x^n + |a_1|x^n + |a_2|x^n + ... + |a_{n-1}|x^n + |a_n|\, x^n \\
&= \{|a_0| + |a_1| + |a_2|. + ... + |a_{n-1}| + |a_n|\}x^n \\
&\le Cx^n
\end{aligned}
$$

Độ tăng của hàm



Ví dụ 4.

S(n) = 1 + 2+.... + n

= n(n+1)/2

Vậy S(n) = $O(n^2)$

Ví dụ 5.

f(n) = n!

$n! < n^n$ $\Rightarrow$ n! = $O(n^n)$.

$\Rightarrow$ log(*n*!) = O(*n* log *n*).

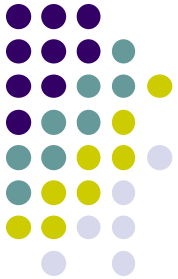
Công thức xấp xỉ Stirling:

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

Đánh giá khác:

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \leq n! \leq \sqrt{2\pi n}\left(\frac{n}{e}\right)^{n + 1/12n}$$

Độ tăng của hàm



Def.

1. Hàm $f(x) = \Omega(g(x))$

   Nếu $\exists$ C>0 và dãy $x_1, x_2, x_3, \ldots \to +\infty$: $|f(x_i)| > C|g(x_i)|$

   Ví dụ:

   $x = \Omega(\log(x))$

2. Hàm f(x) tăng theo hàm mũ

   nếu $\exists$ c >1, d>1:

   $f(x) = \Omega(c^x)$ và $f(x) = O(d^x)$

   Ví dụ,

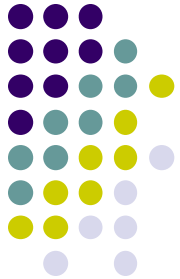
   a) $f(x) = e^{2x}$

      c=e, $d=e^2$: $e^{2x} = \Omega(e^x)$ và $e^{2x} = O(e^{2x})$

   b) $f(n) = n! = 1.2.3\ldots.(n-1).n = \Omega(2^n)$

      $x_i = i$, $f(x_i) = i!$, $g(x_i) = 2^i \Rightarrow f(x_i)=i! > 2^i = g(x_i)$.

Độ tăng của hàm



Mệnh đề 1.2.

$$f(x) = O(u(x)),\ g(x) = O(v(x)) \Rightarrow (f+g)(x) = O\big(\max\{|u(x)|,|v(x)|\}\big)$$

Chứng minh:

$\exists\ C_1, N_1, C_2, N_2$:

$\exists\ x > N,\ N = \max\{N_1, N_2\},\quad C = \max\{C_1, C_2\}$

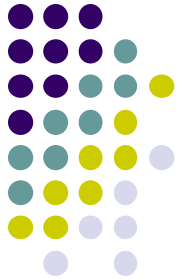
$$\Rightarrow \quad |f(x)| \le C|u(x)|,$$

$$|g(x)| \le C|v(x)|$$

$$\text{hay} \quad |f(x)+g(x)| \le C \max\big\{|u(x)| + |v(x)|\big\}$$

Suy ra $(f+g)(x) = O\big(\max\{u(x),v(x)\}\big)$

Độ tăng của hàm



Mệnh đề 1.3.

$$f(x) = O(u(x)),\ g(x) = O(v(x)) \Rightarrow (fg)(x) = O\big(u(x).v(x)\big)$$

Chứng minh:

$\exists\ C_1, N_1, C_2, N_2$:

$\exists\ x > N,\ N = \max\{N_1, N_2\},\quad C = C_1.C_2$

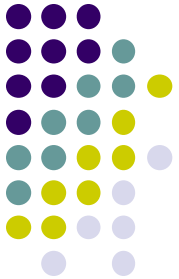
$\Rightarrow\quad |f(x)| \le C_1|u(x)|,$

$|g(x)| \le C_2|v(x)|$

hay $|f(x).g(x)| \le C.|u(x)|.|v(x)|$

Suy ra $(fg)(x) = O\big(u(x).v(x)\big)$

Độ tăng của hàm



Ví dụ 6.

$f(n) = n\log(n!) + (3n^2 + 2n)\log n$

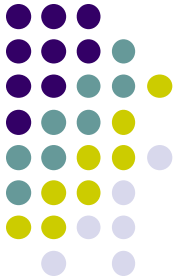
$\log(n!) = O(n\log n) \Rightarrow n\log(n!) = O(n^2\log n)$

$(3n^2 + 2n) = O(n^2) \Rightarrow (3n^2 + 2n)\log n = O(n^2\log n)$

Vậy

$f(n) = O(n^2\log n)$

Độ tăng của hàm



Ví dụ 7.

$f(n) = (n+3).\log(n^2+4) + 5n^2$

$\log (n^2+4) = O(\log n)$

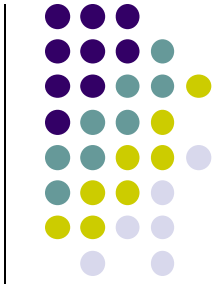
$n+3 = O(n)$ $\Rightarrow$ $(n+3)\log (n^2+4) = O(n\log n)$

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

Vậy

$f(n) = O(n^2)$

Thuật toán và độ phức tạp



Thuật toán

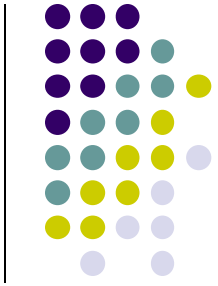
Thuật toán là một bản liệt kê các chỉ dẫn, các quy tắc cần thực hiện theo từng bước xác định nhằm giải quyết một bài toán đã cho trong một khoảng thời gian hữu hạn.

Bài toán: Tìm phần tử lớn nhất trong một dãy hữu hạn các số nguyên

Input: Mảng a có *n* nguyên;
Output: k – chỉ số p/tử lớn nhất trong mảng a;

1) k = 1;
2) for (i=2; i≤n; i++)
    if (a_k<a_i)
        k = i;

Thuật toán và độ phức tạp

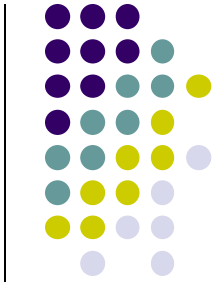


Tính chất:

- *Dữ liệu vào*: Một thuật toán phải mô tả rõ các giá trị đầu vào từ một tập hợp các dữ liệu xác định.
- *Dữ liệu đầu ra*: Giá trị đầu ra chính là nghiệm của bài toán.

- *Tính xác định*: Các bước của thuật toán phải được xác định một cách chính xác, các chỉ dẫn phải rõ ràng, có thể thực hiện được.
- *Tính hữu hạn*: Thuật toán phải kết thúc sau một số hữu hạn bước.
- *Tính đúng đắn*: Thuật toán phải cho kết quả đúng theo yêu cầu của bài toán đặt ra.
- *Tính tổng quát*: Thuật toán phải áp dụng được cho mọi bài toán cùng loại, với mọi dữ liệu đầu vào như đã được mô tả.

Thuật toán và độ phức tạp



Input: Mảng a có *n* nguyên;
Output: k – chỉ số p/tử lớn nhất trong mảng

```
1) k = 1;
2) for  (i=2; i≤n; i++)
       if (a_k<a_i)
           k = i;
```

Tính xác định: Mỗi bước của thuật toán chỉ gồm các phép gán, mệnh đề kéo theo;

Tính hữu hạn: Thuật toán dừng sau khi tất cả các thành phần của mảng đã được kiểm tra; Do có đúng n-1 bước, mỗi bước có số xác định phép toán so sánh và gán nên số phép toán cần thực hiện là hữu hạn.

Tính đúng đắn:

Ở bước thứ 1, phần tử lớn nhất được xác định là k=1.

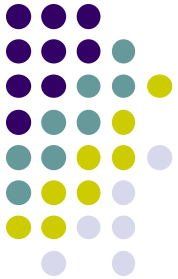
Giả sử ở bước *i*, đã xác định được k-là chỉ số phần tử lớn nhất trong i phần tử đầu tiên;

Ở bước i+1, so sánh a_k với a_i: nếu $a_k < a_i$, đặt lại k = i;

Bằng phép quy nạp, suy ra, sau bước *n*, k-là chỉ số phần tử lớn nhất trong *n* phần tử

Tính tổng quát: Thuật toán cho phép tìm phần tử lớn nhất của dãy *n* số nguyên bất kỳ.

Thuật toán và độ phức tạp



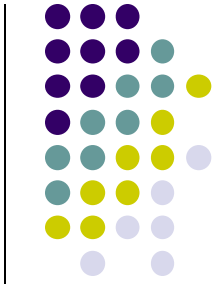
Hiệu quả của thuật toán:

được đo bởi thời gian tính khi các giá trị đầu vào có kích thước xác định;

xem xét theo thước đo dung lượng bộ nhớ đã sử dụng để tính toán khi kích thước đầu vào đã xác định.

(độ phức tạp thời gian và độ phức tạp không gian)

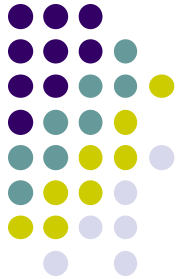
Thuật toán và độ phức tạp



Phân tích:

- Kích thước dữ liệu đầu vào:
  a) **Số lượng phần tử: Ví dụ, mảng có *n* phần tử.**
  b) **Số bit biểu diễn dữ liệu đầu vào: thừa số trong phép nhân 2 số nguyên lớn**
- Độ phức tạp thời gian của một thuật toán thường được biểu diễn thông qua số phép toán trong khi thực hiện thuật toán khi các giá trị dữ liệu đầu vào có kích thước xác định.
- Dùng số phép tính làm thước đo độ phức tạp thời gian thay cho thời gian thực của máy tính.
- Để phân tích thuận tiện, không phân rã các phép toán sơ cấp thành các phép toán bit sơ cấp.
- Đánh giá khả năng xấu nhất của thuật toán, hoặc khả năng trung bình.

Thuật toán và độ phức tạp

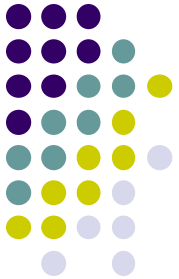


Def.

Một thuật toán được gọi là có độ phức tạp đa thức, hay còn gọi là có thời gian đa thức, nếu số các phép tính cần thiết khi thực hiện thuật toán không vượt quá $O(n^k)$, với k nguyên dương nào đó, còn n là kích thước của dữ liệu đầu vào.

Các thuật toán với $O(b^n)$, trong đó n là kích thước dữ liệu đầu vào, còn b là một số nguyên dương nào đó gọi là các thuật toán có độ phức tạp hàm mũ hoặc thời gian mũ.

Thuật toán và độ phức tạp



Ví dụ 8.

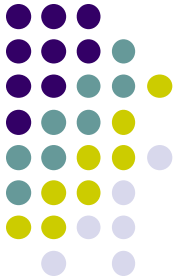
Input: Mảng a có *n* số nguyên;

Output: k – chỉ số p/tử lớn nhất trong mảng a;

```
1) k = 1;
2) for  (i=2; i≤n; i++)
       if (a_k<a_i)
              k = i;
```

Độ phức tạp tính toán: O(n)

Thuật toán và độ phức tạp



Ví dụ 9.

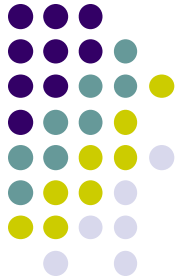
Input: Mảng a có *n* số nguyên được sắp xếp tăng dần, số nguyên b;

Output: k – chỉ số p/tử có giá trị bằng b, k = 0 nếu không tìm được;

```
d = 1;   c = n;
found =false;
while  ((d ≤c) & !found )
        k = (d + c)  div  2;
        if  (b==a(k)) found = true;
         else
                    if (b< a(k))  c= k –1;
                    else          d = k +1;
if  (!found )  k = 0;
```

Độ phức tạp tính toán: O(log*n*)

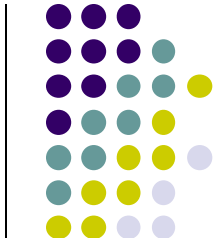
Thuật toán và độ phức tạp



Một vài loại thường gặp:

O(1)	Độ phức tạp hằng số
O(log*n*)	Độ phức tạp logarit
O(*n*)	Độ phức tạp tuyến tính.
O(n^k)	Độ phức tạp đa thức
O(*n*log*n*)	Độ phức tạp *n*log*n*
O(b^n),b>1	Độ phức tạp hàm mũ
O(n!)	Độ phức tạp giai thừa

Thuật toán và độ phức tạp



Ví dụ 10.

Đánh giá số phép chia số nguyên của thuật toán Euclid để tìm ước số chung lớn nhất của hai số nguyên a và b, a>b. Đáp số: O(log*b*).

Thuật toán:

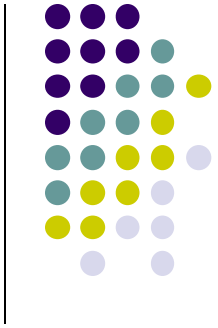
```
x = a; y = b;
while (y>0)
    r = x mod y;
    x = y;
    y = r;
```

Ví dụ: a = 120; b = 48

x	y
120	48
48	24
24	0

USCLN(120, 48) = 24

Thuật toán và độ phức tạp



Chứng minh: Đặt $r_0=a$ và $r_1=b$,

$r_0 = r_1q_2 + r_2$

$r_1 = r_2q_3 + r_3$

.....

$r_{k-2} = r_{k-1}q_k + r_k$, $0 < r_k < r_{k-1}$

$r_{k-1} = r_kq_{k+1}$

Số phép chia: k+1

Ước chung LN: r_k

Hiển nhiên: $q_{k+1} \geq 2$, $q_i \geq 1$, i = 0,1,2,...k

Đặt $f_0 = 0, f_1 = 1;$

$r_k \geq 1 = f_1 + f_0 = f_2$

$r_{k-1} \geq 2r_k \geq f_2 + f_1 = f_3$

$r_{k-2} \geq r_{k-1} + r_k \geq f_3 + f_2 = f_4$

.....

$r_3 \geq r_4 + r_5 \geq f_{k-3} + f_{k-2} = f_{k-1}$

$r_2 \geq r_3 + r_4 \geq f_{k-2} + f_{k-1} = f_k$

$r_1 \geq r_2 + r_3 \geq f_{k-1} + f_k = f_{k+1}$

Cuối cùng

$b \geq r_2 + r_3$

$\{f_k\}$ - dãy Fibonacci.

Do

$$f_k > ((1+\sqrt{(5)})/2)^{k-2}$$

$\Rightarrow$ $$b > ((1+\sqrt{(5)})/2)^{k-1},$$

hay

$$\log b > (n-1)\ \lg(1+\sqrt{5})/2$$

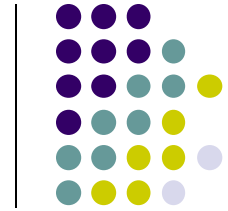
$\log(1+\sqrt{(5)})/2 \approx 0.208 > 1/5$

$\log b > (k-1).\ 1/5$

Suy ra: $k < 1 + 5 \log b$.

Vậy, số phép toán là O(logb).

Thuật toán và độ phức tạp



Ví dụ 11:

Cho một số nguyên n. Hãy tìm số nguyên m lớn nhất mà khi biểu diễn nó theo cơ số 16 thì có các chữ số khác nhau đôi một và tổng các chữ số (ở cơ số 16) đúng bằng n.

Ví dụ: n = 21, m = 6543210

n = 31, m = F643210

Số lớn nhất có thể thành lập từ các chữ số khác nhau trong hệ đếm 16

là FEDCBA9876543210

hay 18,364,757,930,599,073,800

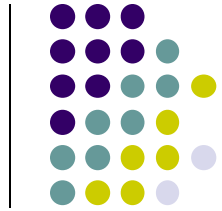
Số n có giá trị lớn hơn 128 thì không cần kiểm tra.

Sử dụng thuật toán tìm kiếm tuần tự thì ta sẽ phải duyệt 18,364,757,930,599,073,800 trường hợp. Mỗi trường hợp phải đổi số tương ứng ra cơ số 16, tính tổng các chữ số và so sánh với n.

Cuối cùng là phải tìm số lớn nhất thoả mãn cả hai điều kiện: chữ số khác nhau và tổng chữ sioos bằng n.

Nếu giả định mỗi giây có thể kiểm tra được 1,000,000 trường hợp thì phải mất 5,101,321,647 giờ, hay 212,555,069 ngày, hay 582,343 năm.

Thuật toán và độ phức tạp



Ví dụ 11:

Cho một số nguyên n. Hãy tìm số nguyên m lớn nhất mà khi biểu diễn nó theo cơ số 16 thì có các chữ số khác nhau đôi một và tổng các chữ số (ở cơ số 16) đúng bằng n.

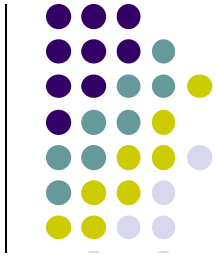
Ví dụ:

n = 21, m = 6543210
n = 31, m = F643210

InPut: Số n và mảng A còn trống.
OutPut: Mảng A chứa các chữ số của m

```
Procedure  Tim_so_m(n:byte);
        For i:=0 to 15 do a[i]:=0;
        i:=0;
        While n > a[i] do
              Inc(i);
              a[i]      :=  i;
                  n :=  n-i;
        j:=15;
        While n>0 do
             t:= min {n, j-a[i] };
             a[i] :=  a[i] + t;
             n:= n - t;
              Dec(j);
              Dec(i);
```

Thuật toán và độ phức tạp



Bài tập:

1. Xây dựng các thuật toán tìm số bit 1 trong một số xâu bit s, số nguyên n. So sánh các thuật toán này.
2. Mô tả thuật toán chèn một số nguyên vào một mảng đã được sắp xếp tăng dần.
3. Mô tả thuật toán tìm các số cực đại trong một dãy hữu hạn các số thực.
4. Mô tả thuật toán tìm từ dài nhất trong một xâu kí tự (ta hiểu từ là một xâu các chữ cái liên tiếp).
5. Mô tả thuật toán tìm kiếm tam phân trên một mảng được sắp xếp tăng dần các số nguyên.
6. Mô tả thuật toán tìm một dãy con liên tiếp không giảm từ một dãy số nguyên cho trước (ở đây ta hiểu dãy con liên tiếp là dãy con gồm các phần tử liên tiếp của dãy ban đầu).
7. Xây dựng thuật toán nhân hai số nguyên, biết rằng mỗi số có thể có tới 100 chữ số.