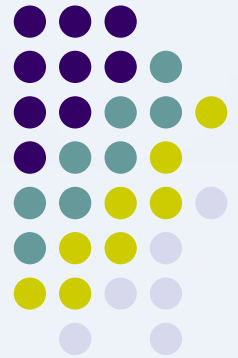


Bài 4. Phân tích các thuật toán

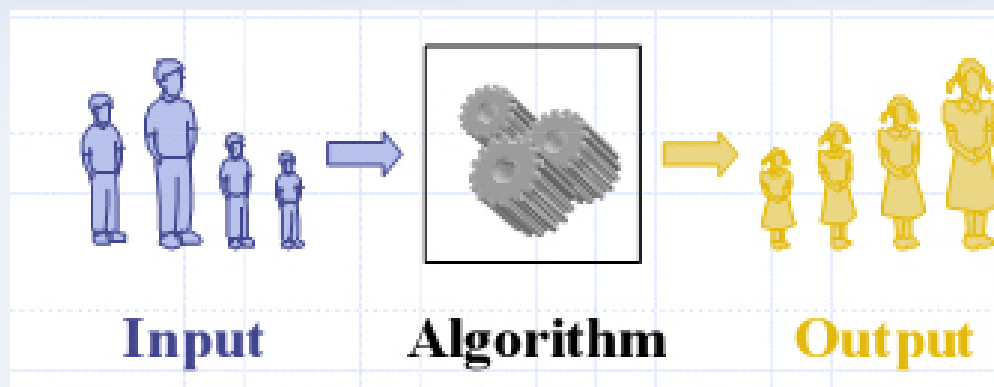
(Analysis of Algorithms)



30/08/2016

Phân tích thuật toán

1



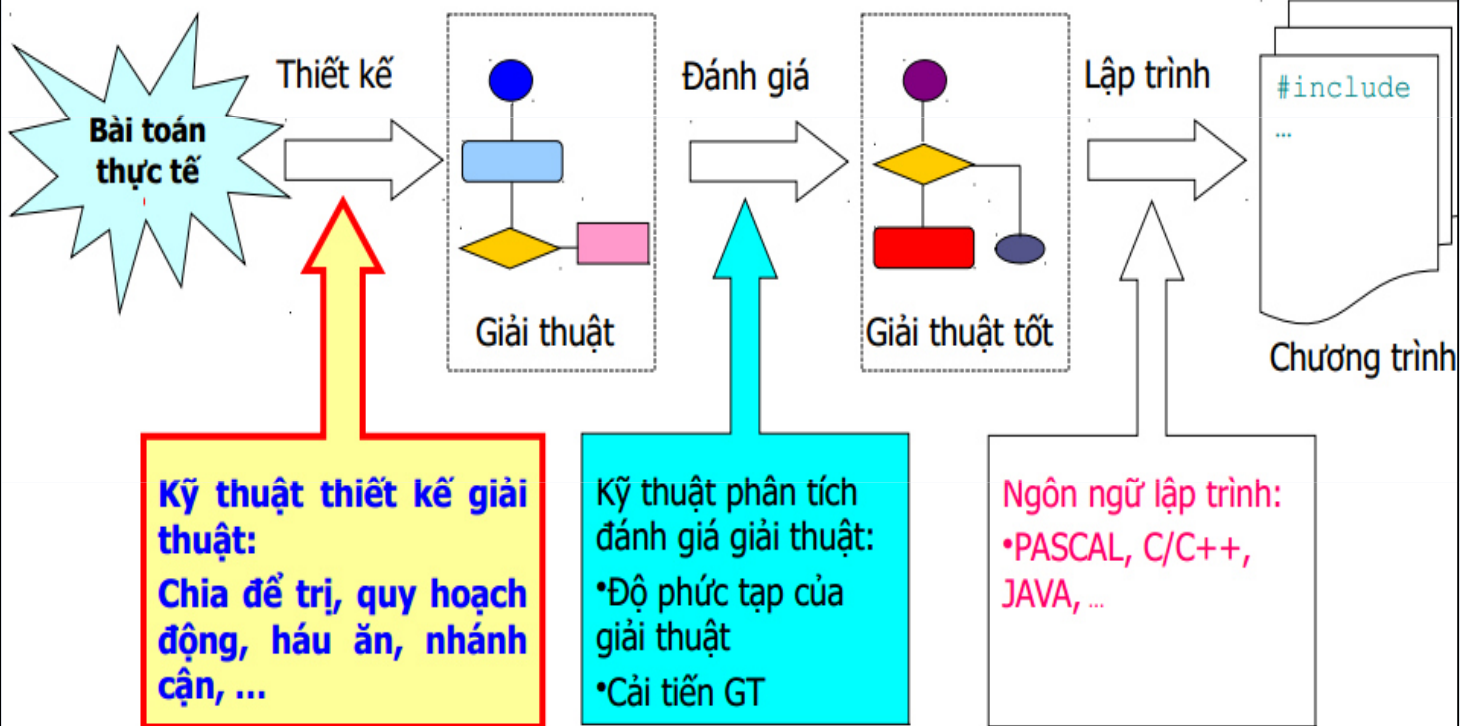
Thuật toán là một qui trình thực hiện từng bước, từng bước giải quyết một vấn đề trong một khoảng thời gian hữu hạn.

30/08/2016

Phân tích thuật toán

2

Từ bài toán đến chương trình



30/08/2016

Phân tích thuật toán

3

Tính hiệu quả của thuật toán

- Thuật toán đơn giản, dễ hiểu
- Thuật toán dễ cài đặt
- Thuật toán cần ít bộ nhớ
- Thuật toán chạy nhanh

- ➔ Khi cài đặt thuật toán chỉ để sử dụng một số ít lần thì ưu tiên tiêu chí 1 và 2
- ➔ Khi cài đặt thuật toán mà sử dụng rất nhiều lần, trong nhiều chương trình khác nhau: sắp xếp, tìm kiếm, đồ thị... thì ưu tiên tiêu chí 3 và 4

30/08/2016

Phân tích thuật toán

4

Các khía cạnh cần phân tích



□ Bộ nhớ (Space)

Xác định tổng dung lượng bộ nhớ cần thiết để lưu trữ toàn bộ dữ liệu đầu vào, trung gian và kết quả đầu ra.

❖ Ví dụ: Sắp xếp một dãy n phần tử.

Bộ nhớ cần cho bài toán là: Bộ nhớ lưu biến n , lưu n phần tử của dãy, lưu các biến i, j, tg (nếu là thuật toán Bubble Sort)

□ Thời gian chạy của thuật toán (Running time)

Thời gian chạy (Running time)



- Hầu hết các thuật toán thực hiện biến đổi các đối tượng đầu vào thành các đối tượng đầu ra.

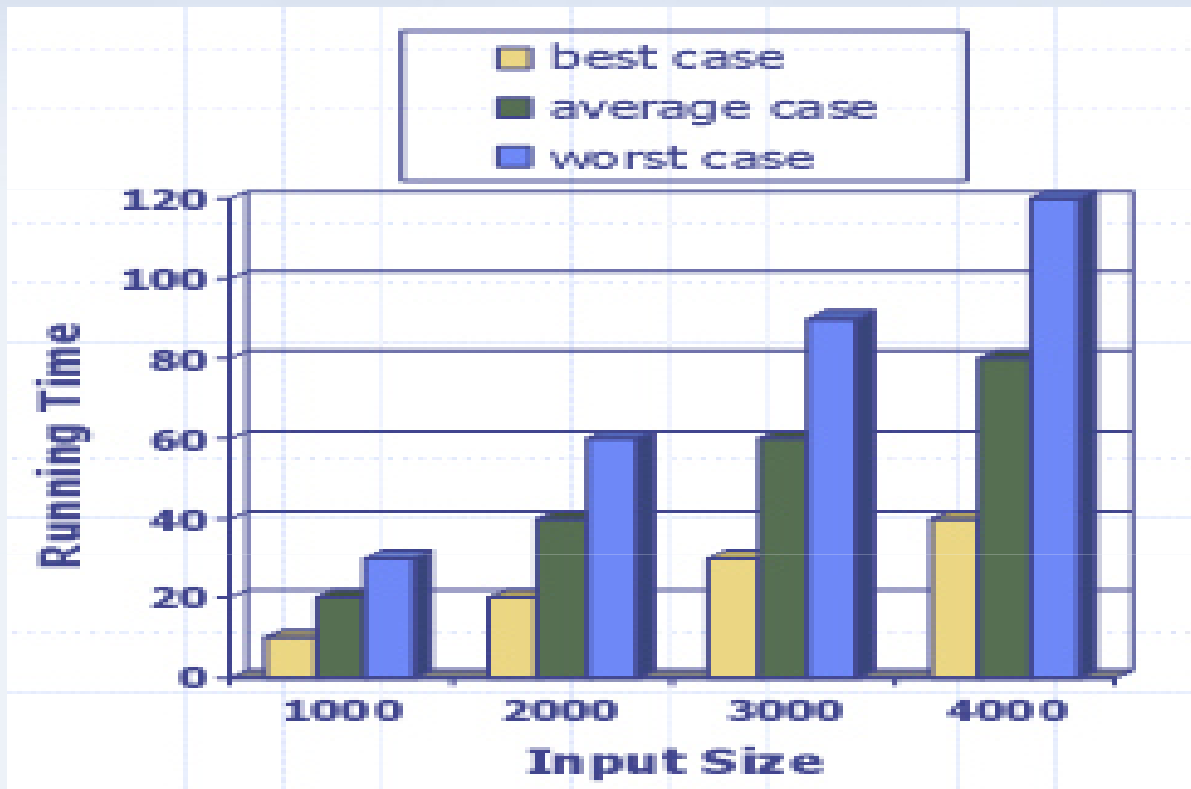
➡ Thời gian chạy của thuật được đặc trưng bởi kích thước của dữ liệu đầu vào.

- Chúng ta thường đi đánh giá thời gian chạy của thuật toán trong 3 trường hợp: **xấu nhất**, **trung bình** và **tốt nhất**.

➡ Thời gian chạy trung bình của thuật toán thường rất khó xác định

- Chúng ta tập trung vào phân tích thời gian chạy trong trường hợp **xấu nhất** (do dễ phân tích)

Thời gian chạy (Running time)



30/08/2016

Phân tích thuật toán

7

Phương pháp đánh giá



1. Phương pháp thực nghiệm
2. Phương pháp phân tích lý thuyết

30/08/2016

Phân tích thuật toán

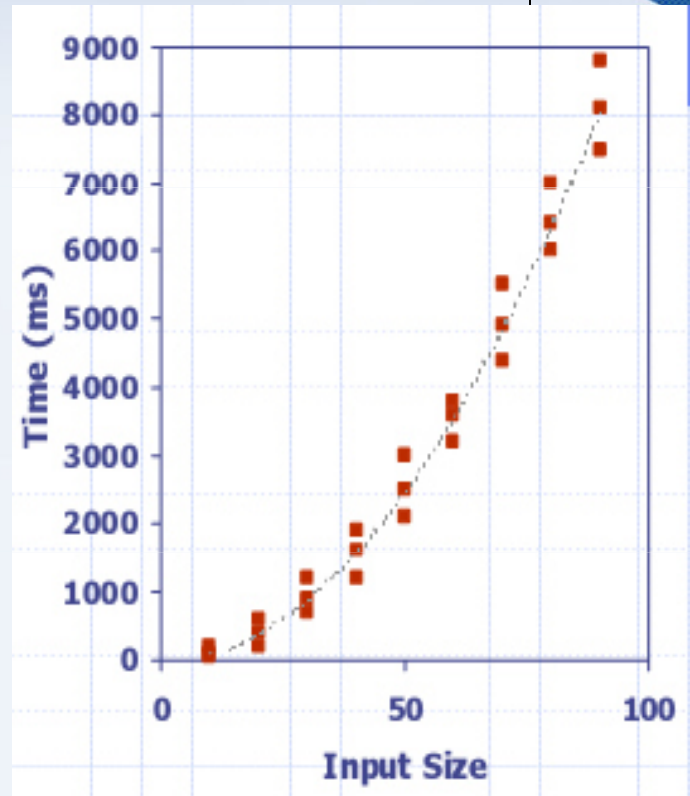
8

Phương pháp thực nghiệm



Các bước thực hiện:

- Viết một chương trình thể hiện thuật toán
- Chạy chương trình với các bộ dữ liệu đầu vào có kích thước khác nhau và tổng hợp lại.
- Sử dụng một hàm như một đồng hồ để lấy chính xác thời gian chạy của thuật toán.
- Vẽ đồ thị biểu diễn kết quả



Hạn chế của phương pháp thực nghiệm



1. Cần phải cài đặt thuật toán bằng một ngôn ngữ lập trình, nhưng một số thuật toán việc cài đặt là khó.
2. Kết quả thu được không thể biểu thị cho những bộ dữ liệu đầu vào chưa được thực nghiệm
3. Phụ thuộc và chương trình dịch
4. Phụ thuộc vào phần cứng của từng máy tính
5. Phụ thuộc kỹ năng của người lập trình

Phương pháp phân tích lý thuyết

- ❑ Sử dụng thuật toán được mô tả ở mức cao (giả mã) thay cho chương trình cài đặt.
- ❑ Mô tả thời gian chạy của thuật toán bằng một hàm phụ thuộc vào kích thước của dữ liệu đầu vào, n .
- ❑ Tính toán tất cả các khả năng của dữ liệu đầu vào
- ❑ Cho phép chúng ta đánh giá tốc độ của thuật toán không phụ thuộc vào phần cứng/môi trường phần mềm.

Giả mã (Pseudocode)

- ❑ Mô tả thuật toán ở mức trừu tượng cao
- ❑ Nhiều cấu trúc hơn ngôn ngữ tự nhiên
- ❑ Kém chi tiết hơn chương trình
- ❑ Sử dụng nhiều ký hiệu để mô tả

Ví dụ thuật toán tìm Max các phần tử của một mảng

Algorithm *arrayMax(A,n)*

Input: Mảng A có n số nguyên

Output: Giá trị lớn nhất của A

Max \leftarrow A[0]

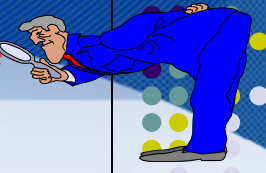
for $i \leftarrow 1$ **to** $n-1$ **do**

if $A[i] > \text{Max}$ **then**

 Max \leftarrow A[i]

return Max

Những chi tiết mô tả PseudoCode



❖ Cấu trúc điều khiển

- If then else
- while do
- For do
- Xuống dòng thay cho dấu {, }

❖ Khai báo phương thức

Algorithm **Phươngthức**([Danh sách đối])

Input:

output:

❖ Gọi hàm, phương thức

Biến.Phươngthức([Danh sách đối])

❖ Trả lại giá trị cho hàm

return Biểu_thức

❖ Các biểu thức

← Phép gán sánh

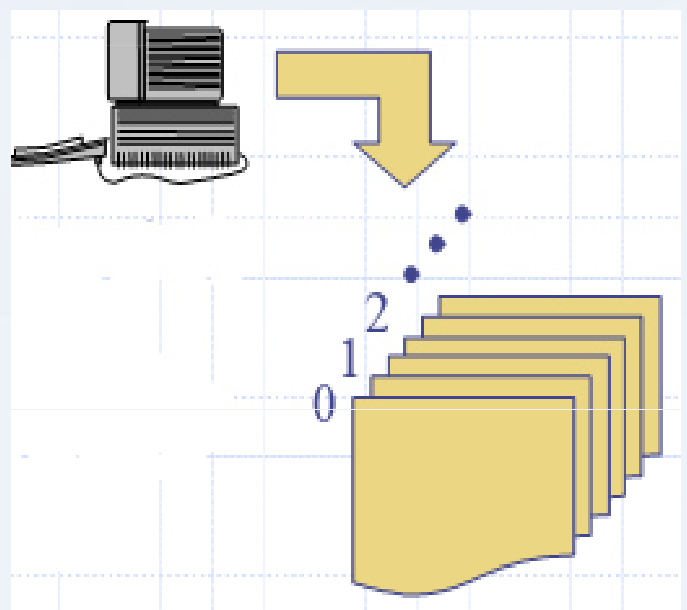
= Phép so sánh bằng

n^2 Cho phép viết số mũ

Mô hình máy truy nhập ngẫu nhiên (Random Access Machine (RAM) Model)



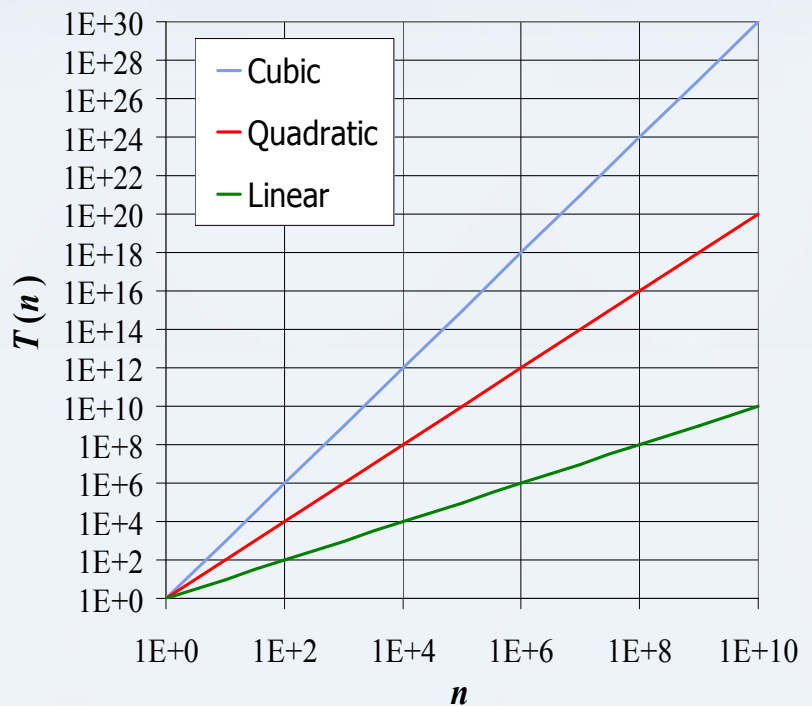
- ❑ Một CPU
- ❑ Không giới hạn số ô nhớ
- ❑ Mỗi ô nhớ có thể lưu một số nguyên hoặc 1 ký tự
- ❑ Mỗi ô nhớ được đánh số và để truy nhập đến mỗi ô nhớ sẽ mất một đơn vị thời gian



Bảng hàm quan trọng sử dụng trong phân tích thuật toán



- Hàm hằng ≈ 1
- Hàm Logarit $\approx \log n$
- Hàm tuyến tính $\approx n$
- N-Log-N $\approx n \log n$
- Hàm bậc 2 $\approx n^2$
- Hàm bậc 3 $\approx n^3$
- Hàm mũ $\approx 2^n$
- Trong biểu đồ log-log, độ nghiêng của đường thẳng tương ứng với tốc độ phát triển của hàm



30/08/2016

Phân tích thuật toán

15

Các phép toán cơ sở



1. Các phép toán cơ sở được thực hiện bởi thuật toán được xem là **như nhau**
2. Độc lập với ngôn ngữ lập trình
3. Không cần thiết xác định chính xác số lượng các phép toán
4. Giả thiết mỗi phép toán mất **một khoảng thời xác định** để thực hiện trong mô hình RAM

Các phép toán cơ sở

- Định giá một biểu thức
- Gán giá trị cho một biến
- Đưa vào/truy cập một phần tử mảng
- Gọi hàm
- Trả lại giá trị cho hàm (**return**)

30/08/2016

Phân tích thuật toán

16

Xác định số phép toán cơ sở

- Bằng cách duyệt thuật toán giả mã, chúng ta có thể xác định được số phép tính tối đa mà thuật toán có thể phải thực hiện
- Từ đó ta xây dựng được một hàm thể hiện thời gian chạy của thuật toán phụ thuộc vào kích thước dữ liệu vào.

Ví dụ:

Algorithm **arrayMax(A,n)**

Số phép toán

Max \leftarrow A[0]

2

for i \leftarrow 1 to n-1 do

2+n

if A[i] > Max then

2(n-1)

Max \leftarrow A[i]

2(n-1)

return Max

1

Ước lượng thời gian chạy

- Thuật toán **ArrayMax** thực hiện **5n+1** phép tính cơ bản trong trường hợp xấu nhất
- Định nghĩa:
 - a = Khoảng thời gian ngắn nhất cần để thực hiện một phép tính cơ bản
 - b = Khoảng thời gian dài nhất cần để thực hiện một phép tính cơ bản
- Ký hiệu T(n) là thời gian chạy trong trường hợp xấu nhất của thuật toán **ArrayMax** thì:
 $a(5n+1) < T(n) < b(5n+1)$
- Do đó thời gian chạy T(n) được bao bởi 2 đường tuyến tính

Thời gian chạy của các lệnh



1. Các phép toán sơ cấp: $O(1)$

2. Lệnh gán: $X = \langle \text{Biểu thức} \rangle$

Thời gian: là tg thực hiện biểu thức

3. Lệnh lựa chọn:

if (điều kiện)	→	$T_0(n)$
lệnh 1	→	$T_1(n)$
else		
lệnh 2	→	$T_2(n)$

Thời gian: $T_0(n) + \max(T_1(n), T_2(n))$

Thời gian chạy của các lệnh



4. Các lệnh lặp: for, while, do..while:

$$\sum_{i=1}^{X(n)} (T_0(n) + T_i(n))$$

$X(n)$: Số vòng lặp

$T_0(n)$: Điều kiện lặp

$T_i(n)$: Thời gian thực hiện vòng lặp thứ i

Nếu tg thực hiện thân vòng lặp không đổi thì tg thực hiện vòng lặp = số lần lặp x tg thực hiện thân vòng lặp

Tốc độ phát triển của thời gian chạy

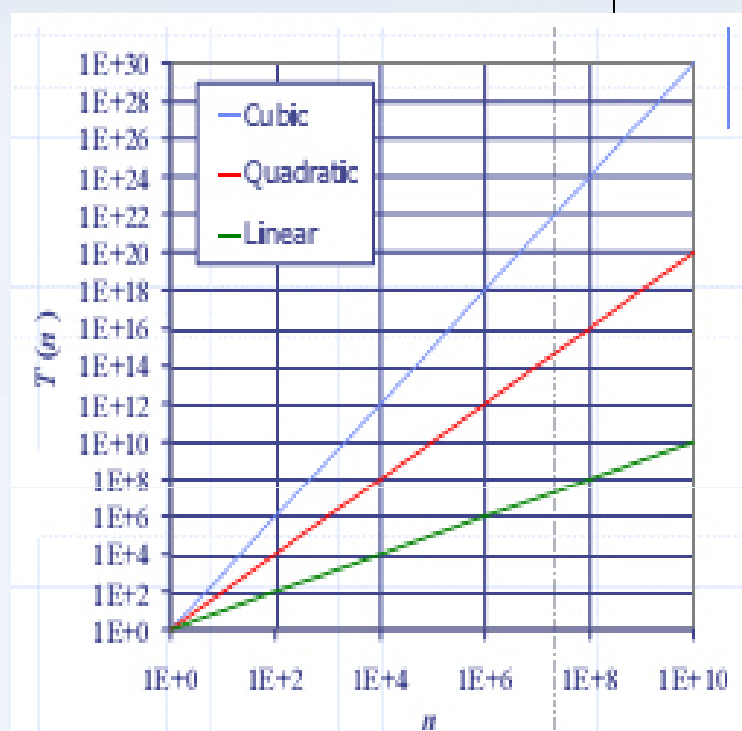


- ❖ Khi thay đổi Phần cứng/Môi trường phần mềm
 - Ảnh hưởng đến $T(n)$ là 1 hằng số, nhưng không làm thay đổi tốc độ phát triển của $T(n)$
- ❖ Tốc độ phát triển tuyến tính của $T(n)$ là bản chất của thuật toán **Arraymax**.

Tốc độ phát triển TG của thuật toán



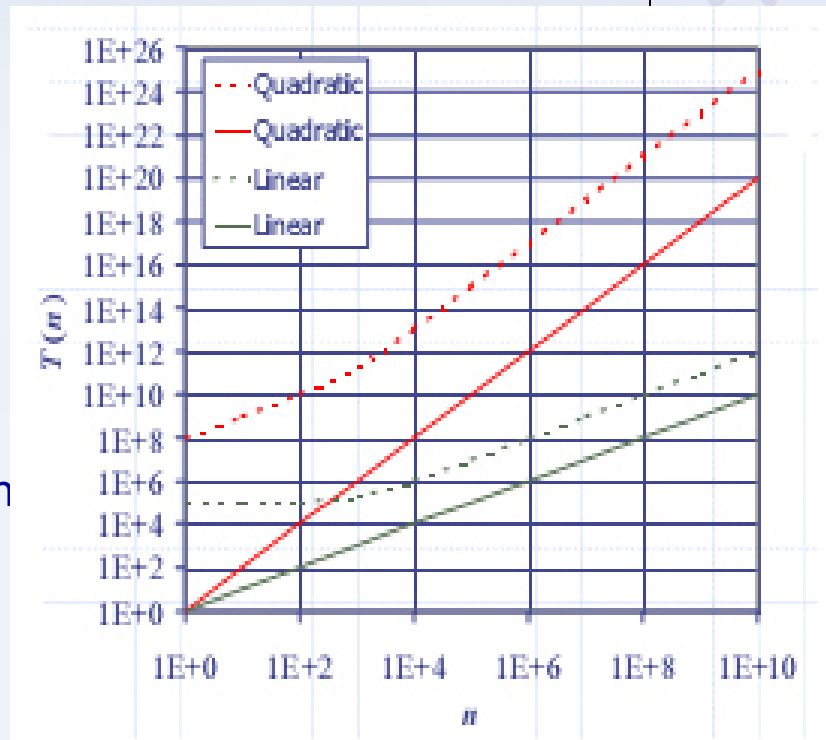
- Các hàm thể hiện tốc độ phát triển TG, ví dụ như:
 - Tuyến tính : n
 - Bậc 2 : n^2
 - Bậc 3 : n^3
- Trong biểu đồ, độ nghiêng của các đường thể hiện tốc độ phát triển của các hàm



Hệ số hằng



- Tốc độ phát triển của hàm không bị ảnh hưởng bởi:
 - Hệ số hằng và
 - Số hạng bậc thấp
- Ví dụ:
 - $10^2n + 10^5$ là hàm tuyến tính
 - $10^2n^2 + 10^5n$ là hàm bậc 2



30/08/2016

Phân tích thuật toán

23

Ký hiệu ô-lớn (Big-Oh)



- Cho hàm $f(n)$ và $g(n)$, chúng ta nói rằng $f(n)$ có ô lớn là $O(g(n))$, nếu tồn tại hằng số dương c và số nguyên n_0 sao cho:

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

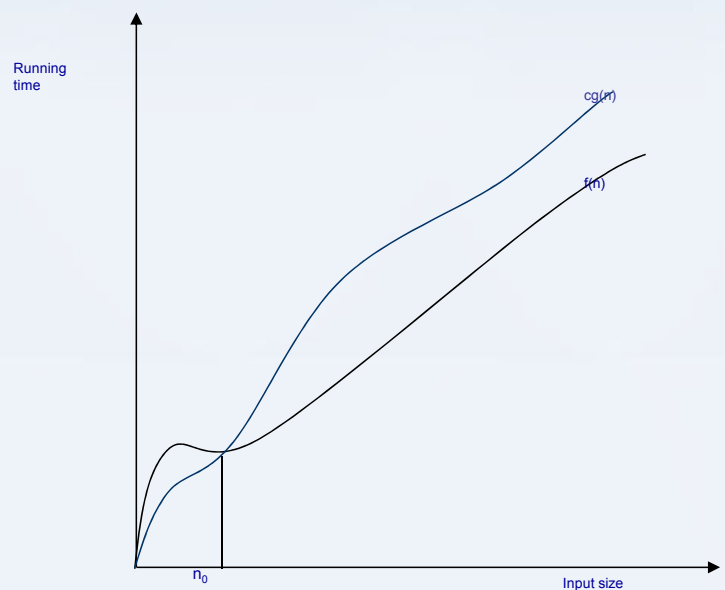
- Ví dụ: $2n + 10$ là $O(n)$

Thật vậy: $2n + 10 \leq cn$

$$10 \leq (c-2)n$$

$$10/(c-2) \leq n$$

Chọn $c=3$ và $n_0=10$



30/08/2016

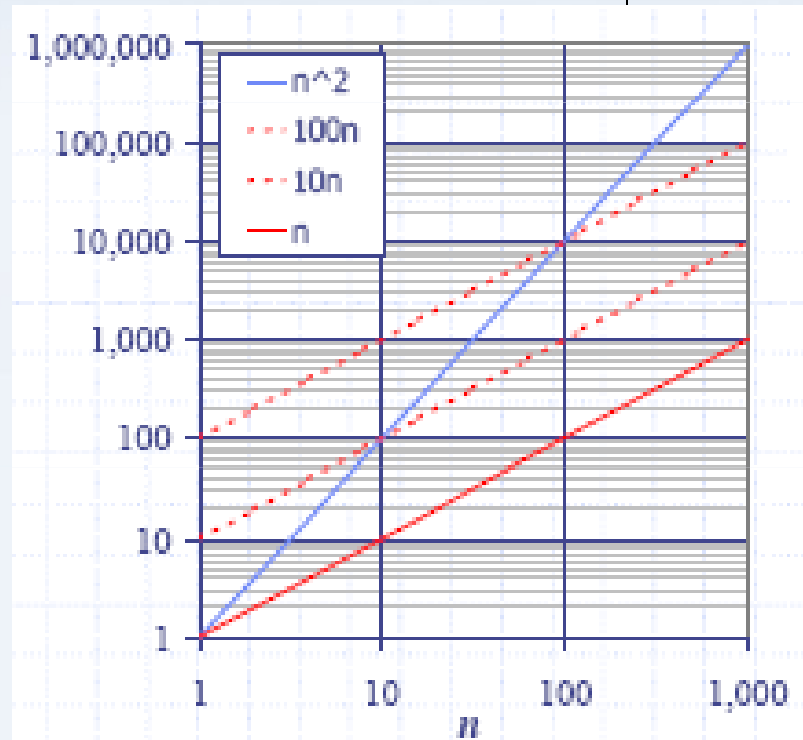
Phân tích thuật toán

24



Ví dụ:

- Hàm n^2 không là $O(n)$ vì:
 - $n^2 \leq cn$
 - $n \leq c$
- Không thể xác định được hằng số c thỏa mãn điều kiện trên



30/08/2016

Phân tích thuật toán

Thêm một số ví dụ về ô-lớn



- $7n-2$
 $7n-2$ là $O(n)$
Vì: chọn hằng số $c=7$ và $n_0=1$ khi đó $7n-2 \leq cn \quad \forall n \geq n_0$
- $3n^3+20n^2+5$
 $3n^3+20n^2+5$ là $O(n^3)$
Vì nếu chọn $c=4$ và $n_0=21$ khi đó $3n^3+20n^2+5 \leq cn^3 \quad \forall n \geq n_0$
- $3\log n + \log \log n$
 $3\log n + \log \log n$ là $O(\log n)$
Vì nếu chọn $c=4$ và $n_0=2$ khi đó $3\log n + \log \log n \leq c \cdot \log n \quad \forall n \geq n_0$

30/08/2016

Phân tích thuật toán

26



Thêm một số ví dụ về ô-lớn

1. $3n^2 - 15n^4 + 4n - 83$
2. $999n^2 - (3+2002*n)*n^4 + (1/5+ n)*n^5 + 2014$
3. $4\log n + n^3 + 543n^2 - 75n - 2012$
4. $54n - 2*n*(n-1)$
5. $2\log n - 1654n$



Ô-lớn và tốc độ phát triển giá trị

- Ký hiệu Ô-lớn chỉ ra một cận trên của tốc độ phát triển giá trị của một hàm
- Ta nói “ $f(n)$ là $O(g(n))$ ” có nghĩa là tốc độ phát triển giá trị của $f(n)$ không lớn hơn tốc độ phát triển của $g(n)$.
- Chúng ta có thể sử dụng ký hiệu Ô-lớn để xếp hạng các hàm theo thứ tự tốc độ phát triển giá trị nó.

	$f(n)$ là $O(g(n))$	$g(n)$ là $O(f(n))$
Tốc độ $g(n)$ lớn hơn	Đúng	Không
Tốc độ bằng nhau	Đúng	Đúng

Qui tắc xác định Ô-lớn



- **Nếu $f(n)$ là đa thức bậc d thì $f(n)$ là $O(n^d)$**
 - Bỏ qua các số hạng bậc thấp
 - Bỏ qua các hệ số hằng
- **Sử dụng lớp hàm nhỏ nhất có thể**
 - Ta nói “ $2n$ là $O(n)$ ” thay cho “ $2n$ là $O(n^2)$ ”
- **Sử dụng lớp hàm đơn giản nhất có thể**
Ta nói “ $3n+5$ là $O(n)$ ” thay cho “ $3n+5$ là $O(3n)$ ”

Phân tích tiệm cận



- Việc phân tích thời gian chạy tiệm cận của một thuật toán được xác định bằng ký hiệu Ô-lớn (O)
- **Thực hiện phân tích:**
 - Tìm số phép toán cơ bản cần phải thực hiện trong trường hợp xấu nhất, thể hiện bằng một hàm phụ thuộc vào kích thước của dữ liệu đầu vào.
 - Diễn tả hàm bằng ký hiệu Ô-lớn
- Các hệ số hằng và các số hạng bậc thấp bị bỏ qua khi xác định số phép toán cơ bản.

Phân tích tiệm cận



- **Ví dụ:**
 - Chúng ta đã xác định thuật toán ArrayMax thực hiện tối đa $5n+1$ phép toán cơ bản
 - Chúng ta nói rằng thuật toán ArrayMax chạy trong thời gian $O(n)$

Ví dụ: Tính trung bình các phần tử đầu dãy (prefix average)



- Để minh họa phân tích tiệm cận chúng ta phân tích hai thuật toán tính trung bình các phần tử đầu dãy sau:
- Hãy tính trung bình i phần tử đầu của một mảng, với $i=0, \dots, n-1$. Trung bình i phần tử đầu của dãy X là:
$$A[i] = (X[0] + X[1] + \dots + X[i-1]) / (i+1)$$



Thuật toán độ phức tạp bậc hai

- Thuật toán được định nghĩa như sau:

Algorithm prefixAverage(X, n)

Input: mảng X có n số nguyên

Output: Mảng trung bình các phần tử đầu dãy của X

$A \leftarrow \text{new int}[n];$	n
for i $\leftarrow 0$ to n-1 do	n+3
s $\leftarrow X[0];$	2n
for j $\leftarrow 1$ to i do	$1 + 2 + \dots + (n-1) + n + n$
s $\leftarrow s + X[j];$	$3(1+2+\dots+(n-1))$
A[i] $\leftarrow s/(i+1);$	4n
return A;	1

30/08/2016

Phân tích thuật toán

33



Thuật toán độ phức tạp bình phương

- Tổng số phép toán tối đa thuật toán thực hiện là:

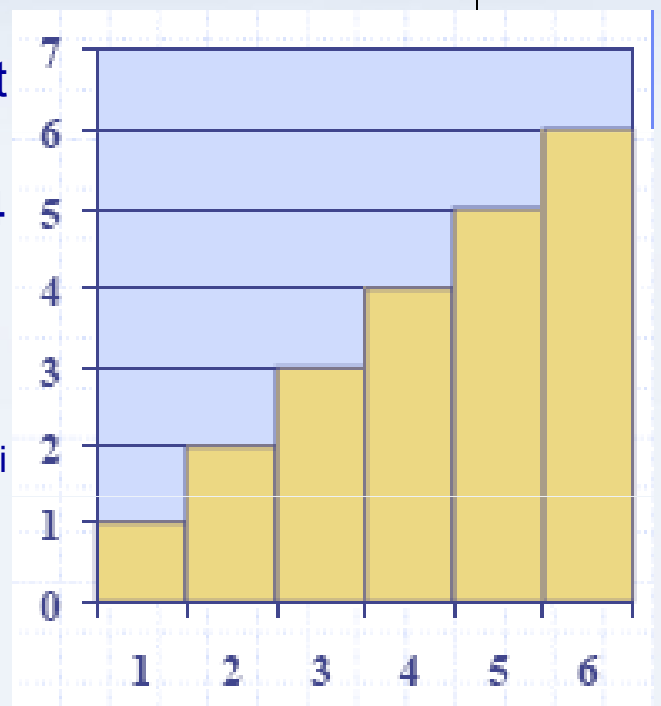
$$T(n) = 4(1+2+\dots+(n-1))+10n+4$$

$$T(n) = 4(1+2+\dots+n) + 6n+4$$

- Tổng của n số nguyên đầu là $n(n+1)/2$

- Hình bên minh họa tốc độ gia tăng thời gian thực hiện của thuật toán

$$T(n) = 2n^2 + 8n + 4$$



30/08/2016

Phân tích thuật toán

34

Thời gian chạy của thuật toán



- Thời gian chạy của thuật toán **prefixAverages1** là:
 $O(2n^2 + 8n + 4)$
- Do đó thuật toán **prefixAverages1** có thời gian chạy là
 $O(n^2)$

Thuật toán độ phức tạp bậc nhất (tuyến tính)



- Thuật toán được mô tả như sau:

Algorithm prefixAverage(X, n)

Input: mảng X có n số nguyên

Output: Mảng trung bình các phần tử đầu dãy của X

$A \leftarrow \text{new int}[n];$	n
$s \leftarrow 0;$	1
for i $\leftarrow 0$ to n-1 do	n+3
$s \leftarrow s + X[i];$	3n
$A[i] \leftarrow s/(i+1);$	4n
return A;	1

- Tổng số phép toán tối đa cần phải thực hiện là
 - $T(n) = 9n + 5$
- Độ phức tạp tiệm cận của thuật toán **prefixAverages2** là $O(n)$

Xác định độ phức tạp của thuật toán

• Quy tắc cộng

Nếu một thuật toán thực hiện hai đoạn chương trình P1, P2 rời nhau và có độ phức tạp tương ứng là $O(g(n))$ và $O(f(n))$. Khi đó độ phức tạp của thuật toán là:

$$T(n) = O(\max\{g(n), f(n)\}).$$

• Ví dụ:

```
for i = 1 to n do  
  input a[i];
```

```
Min ← a[0];
```

```
for i=1 to n-1 do  
  if a[i]<min then  
    min←a[i];
```

P1 có thời gian chạy là $O(n)$

P2 có thời gian chạy là $O(1)$

P3 có thời gian chạy là $O(n)$

Vậy thời gian chạy của cả thuật toán là: $T(n) = O(\max\{1, n, n\}) = O(n)$

Xác định độ phức tạp của thuật toán

• Quy tắc nhân

Nếu một thuật toán thực hiện hai đoạn chương trình P1, P2, có độ phức tạp tương ứng là $O(g(n))$ và $O(f(n))$ và P2 lồng trong P1. Khi đó độ phức tạp của thuật toán là:

$$T(n) = O(g(n) * f(n)).$$

• Ví dụ:

```
for i = 1 to n do  
  input a[i];
```

```
for i ← 1 to n-1 do  
  for j ← i+1 to n do  
    if a[i]<a[j] then  
      tg ← a[i];  
      a[i] ← a[j];  
      a[j] ← tg;
```

P1 có thời gian chạy là $O(n)$

P2 có thời gian chạy là $O(n)$

P3 có thời gian chạy là $O(n)$

Xác định độ phức tạp của thuật toán



- Ta thấy đoạn chương trình P3 lồng trong đoạn chương trình P2. Áp dụng qui tắc nhân thì độ phức tạp của đoạn chương trình P2 và P3 là: $O(n*n)$ hay $O(n^2)$.
- Áp dụng qui tắc cộng cho đoạn chương trình gồm P1, P2, P3 thì ta được độ phức tạp của thuật toán: $T(n) = O(n^2)$.

Một số hàm sử dụng để đánh giá tốc độ gia tăng thời gian chạy.



Constant	Logarithm	Linear	n-log-n	Quadratic	cubic	exponent
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

Một số hàm sử dụng để đánh giá tốc độ gia tăng thời gian chạy.

Thời gian chạy	Cỡ dữ liệu vào					
	10	20	30	40	50	60
n	0,00001 giây	0,00002 giây	0,00003 giây	0,00004 giây	0,00005 giây	0,00006 giây
n ²	0,0001 giây	0,0004 giây	0,0009 giây	0,0016 giây	0,0025 giây	0,0036 giây
n ³	0,001 giây	0,008 giây	0,027 giây	0,064 giây	0,125 giây	0,216 giây
n ⁵	0,1 giây	3,2 giây	24,3 giây	1,7 phút	5,2 phút	13 phút
2 ⁿ	0,001 giây	1,0 giây	17,9 phút	12,7 ngày	35,7 năm	366 thế kỷ
3 ⁿ	0,059 giây	58 phút	6,5 năm	3855 thế kỷ	2.10 ⁸ thế kỷ	1,3. 10 ¹³ thế kỷ

30/08/2016

Phân tích thuật toán

41

Tóm lại:

1. Thời gian thực hiện của mỗi lệnh cơ sở là $O(1)$.
2. Thời gian thực hiện của một chuỗi tuần tự các lệnh được xác định bằng qui tắc cộng. Như vậy thời gian này là thời gian thi hành một lệnh nào đó lâu nhất trong chuỗi lệnh.
3. Thời gian thực hiện cấu trúc IF là thời gian lớn nhất thực hiện lệnh sau ĐK hoặc sau ELSE và thời gian kiểm tra điều kiện. Thường thời gian kiểm tra điều kiện là $O(1)$.
4. Thời gian thực hiện vòng lặp là tổng (trên tất cả các lần lặp) thời gian thực hiện thân vòng lặp. Nếu thời gian thực hiện thân vòng lặp không đổi thì thời gian thực hiện vòng lặp là tích của số lần lặp với thời gian thực hiện thân vòng lặp.

30/08/2016

Phân tích thuật toán

42

Bài tập: Tính độ phức tạp thuật toán



1. Thuật toán tạo ma trận đơn vị A cấp n

```
(1)  for (i = 0 ; i < n ; i++)  
(2)      for (j = 0 ; j < n ; j++)  
(3)          A[i][j] = 0;  
  
(4)  for (i = 0 ; i < n ; i++)  
(5)      A[i][i] = 1;
```

Bài tập: Tính độ phức tạp thuật toán



2. Thuật toán tạo ma trận đơn vị A cấp n (v2)

```
(1)  for (i = 0 ; i < n ; i++)  
(2)      for (j = 0 ; j < n ; j++)  
(3)          if (i == j)  
(4)              A[i][j] = 1;  
(5)          Else  
(6)              A[i][j] = 0;
```


Bài tập: Tính độ phức tạp thuật toán



3. Thuật toán tính tổng

```
1) sum = 0;
2) for ( i = 0; i < n; i ++ )
3)     for ( j = i + 1; j <= n; j ++ )
4)         for ( k = 1; k < 10; k ++ )
5)             sum = sum + i * j * k ;
```

Bài tập: Tính độ phức tạp thuật toán



4. Thuật toán tính tổng (v2)

```
1. for ( i = 0; i < n; i ++ )
2.     for ( j = 0; j < m; j ++ ) {
3.         int x = 0;
4.         for ( k = 0; k < n; k ++ )
5.             x = x + k;
6.         for ( k = 0; k < m; k ++ )
7.             x = x + k;
8.     }
```


Bài tập: Tính độ phức tạp thuật toán



5. Tính

$$e^x = 1 + (x/1!) + (x^2x/2!) + (x^3x^2x/3!) + \dots + (x^n/n!)$$

```
double SumDevideFactorial(int n){
    double S = 1;
    double p = 1;
    for(int i = 0; i < n; i++){
        for(int j = 1; j < i; j++){
            p = p*x/ j;
            S += p;
        }
    }
    return S;
}
```

Bài tập: Tính độ phức tạp thuật toán



5. Tính

$$e^x = 1 + (x/1!) + (x^2x/2!) + (x^3x^2x/3!) + \dots + (x^n/n!)$$

Thuật giải 2: Kế thừa bước trước để tính bước sau

```
double SumDevideFactorial(int n){
    double S = 1;
    double p = 1;
    for(int i = 1; i < n; i++){
        p = p*x/ i;
        S += p;
    }
    return S;
}
```