Phân tích thuật toán (Algorithm Analysis)

Bài giảng môn Cấu trúc dữ liệu và giải thuật

Khoa Công nghệ thông tin
Trường Đại học Thủy Lợi

Nội dung

1. Phân tích thuật toán là gì?
2. Các ký hiệu tiệm cận
3. Tốc độ tăng của các hàm
4. Các ví dụ phân tích thuật toán

1. Phân tích thuật toán là gì?

Phân tích thuật toán

- Nhằm xác định **thời gian chạy** (độ phức tạp) của thuật toán dưới dạng một hàm f của kích thước đầu vào n.
  - Ví dụ: Thời gian tìm kiếm tuần tự một phần tử x trong một dãy n phần tử là f(n) = n (phép so sánh, trong trường hợp tồi/xấu nhất).
- Đơn vị thời gian:
  - Không phải là giờ, phút, giây.
  - Mà là **thao tác cơ bản**; ví dụ: cộng, nhân, so sánh…
  - Mỗi thao tác cơ bản có thời gian chạy là hằng (một lượng thời gian nhỏ không phụ thuộc vào kích thước đầu vào n).

Đếm số thao tác cơ bản

- Nhận diện các thao tác cơ bản trong thuật toán.
- Xác định thao tác cơ bản T chiếm nhiều thời gian chạy nhất so với các thao tác cơ bản còn lại.
  - Thao tác T này thường xuất hiện trong các vòng lặp.
- Đếm số lần thực hiện thao tác T, sẽ thu được hàm thời gian chạy f(n).
- Chú ý: Trong trường hợp khó tìm ra thao tác T, có thể đếm tất cả các thao tác cơ bản. Khi đó, sẽ thu được hàm $f'(n) \neq f(n)$, nhưng nếu áp dụng thêm phép **phân tích tiệm cận** (học sau) thì các kết quả cuối cùng sẽ giống nhau.

Ví dụ đếm số thao tác cơ bản

Ví dụ 1: In các phần tử (C++)

```
for (i = 0; i < n; i++)
  cout << a[i] << endl;
```

Số lần in ra màn hình = n

Ví dụ 2: Nhân ma trận tam giác dưới với véctơ (mã giả)

```
for i ← 1 to n
  c_i ← 0
for i ← 1 to n
  for j ← 1 to i
    c_i ← c_i + a_ij * b_j
```

Số phép nhân = $\sum_{i=1}^{n} i = n(n+1)/2$

Ví dụ 3: Kiểm tra tính sắp xếp (C++)

```
bool isSorted(int a[], int n)
{
   bool sorted = true;
   for (int i=0; i<n-1; i++)
      if (a[i] > a[i+1])
         sorted = false;
   return sorted;
}
```

Số phép so sánh (trong if) = n – 1

Có thể cải tiến thuật toán bên trên?

2. Các ký hiệu tiệm cận

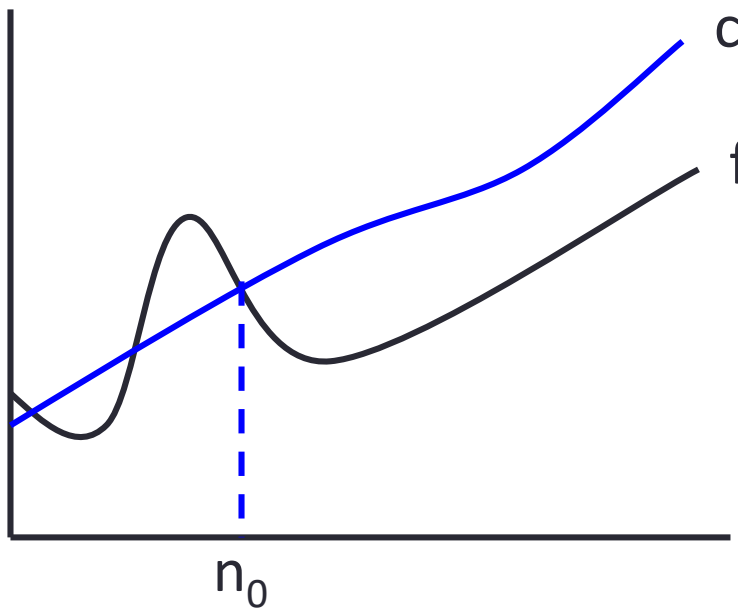
Phân tích tiệm cận

- Nhằm xem xét tốc độ tăng của hàm f(n) khi n dần tới $+\infty$.
- Cho phép quy các dạng hàm f(n) khác nhau về một số ít dạng cơ bản, như log n, n, n^2…
  - Giúp so sánh (cỡ) thời gian chạy của các thuật toán dễ hơn.
- Ta sẽ tìm hiểu ba cách phân tích tiệm cận tương ứng với ba ký hiệu tiệm cận sau đây:
  - Ô lớn: O $\rightarrow$ tìm cận trên của f(n)
  - Ô-mê-ga lớn: Ω $\rightarrow$ tìm cận dưới của f(n)
  - Tê-ta lớn: Θ $\rightarrow$ tìm cận chặt của f(n)

Ký hiệu O

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

khi và chỉ khi $\exists\ c > 0$ và $n_0 > 0$ sao cho $f(n) \leq cg(n)\ \forall n \geq n_0$

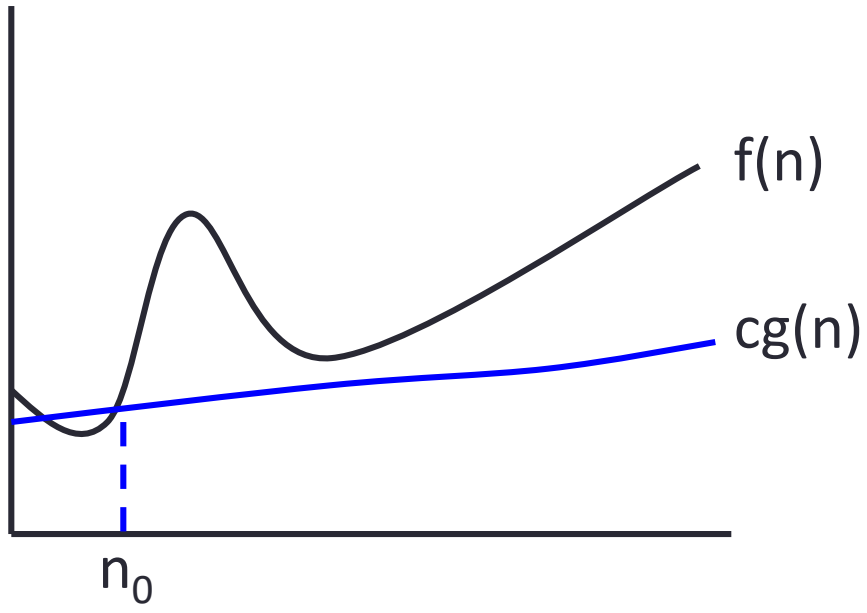


f(n) bị chặn trên bởi g(n) theo nghĩa tiệm cận

Ký hiệu Ω

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

khi và chỉ khi $\exists\ c > 0$ và $n_0 > 0$ sao cho $cg(n) \leq f(n)\ \forall n \geq n_0$



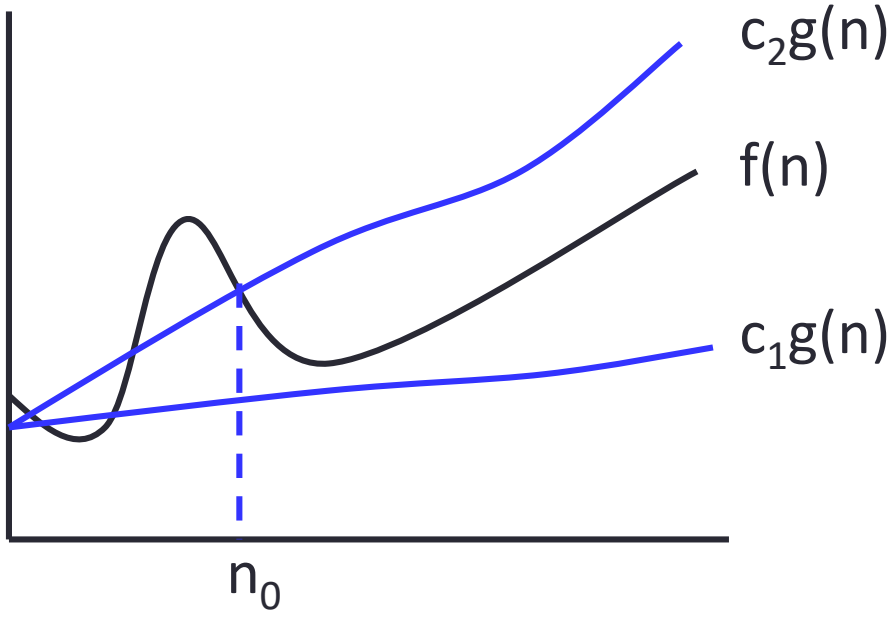
f(n) bị chặn dưới bởi g(n) theo nghĩa tiệm cận

Ký hiệu Θ

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

khi và chỉ khi $\exists\ c_1 > 0, c_2 > 0$ và $n_0 > 0$ sao cho

$c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall n \geq n_0$



f(n) có cùng tốc độ tăng với g(n) theo nghĩa tiệm cận

Ví dụ phân tích tiệm cận

$f(n) = 3n^2 + 17 =$

- $\Omega(1), \Omega(n), \Omega(n^2)$ $\rightarrow$ cận dưới
- $O(n^2), O(n^3), O(n^4)\ldots$ $\rightarrow$ cận trên
- $\Theta(n^2)$ $\rightarrow$ cận chặt

Hãy điền vào chỗ dấu chấm hỏi !

$f(n) = 1000\ n^2 + 17 + 0{,}001\ n^3 =$

- $\Omega(?)$ $\rightarrow$ cận dưới
- $O(?)$ $\rightarrow$ cận trên
- $\Theta(?)$ $\rightarrow$ cận chặt

Tính chất bắc cầu

- Nếu $f(n) = O(g(n))$ và $g(n) = O(h(n))$

  $\rightarrow f(n) = O(h(n))$

- Nếu $f(n) = \Omega(g(n))$ và $g(n) = \Omega(h(n))$

  $\rightarrow f(n) = \Omega(h(n))$

- Nếu $f(n) = \Theta(g(n))$ và $g(n) = \Theta(h(n))$

  $\rightarrow f(n) = \Theta(h(n))$

Một số tính chất khác

- Nếu $f(n) = a_0 + a_1 n + \ldots + a_k n^k$ $(a_k > 0)$

  $\rightarrow f(n) = O(n^k)$

- $\log^k n = O(n)$ với k là một hằng số

  (hàm lôgarít tăng chậm hơn hàm tuyến tính)

Chú ý:

- Trong môn học này, khi viết hàm lôgarít mà không chỉ rõ cơ số, ta ngầm hiểu cơ số là 2.
- Từ giờ trở đi, ta chỉ tập trung vào ký hiệu O.

3. Tốc độ tăng của các hàm

Tốc độ tăng của một số hàm cơ bản

Hàm	Tên
c	Hằng
log n	Lôgarít
$\log^2 n$	Lôgarít bình phương
n	Tuyến tính
n log n	
n^2	Bậc hai
n^3	Bậc ba
2^n	Hàm mũ

Hàm nào tăng chậm hơn?

- $f(n) = n \log n$ và $g(n) = n^{1,5}$
- Lời giải:
  - Chú ý rằng $g(n) = n^{1,5} = n * n^{0,5}$.
  - Vì vậy, chỉ cần so sánh $\log n$ và $n^{0,5}$.
  - Tương đương với so sánh $\log^2 n$ và n.
  - Tham khảo tính chất trong slide trước: $\log^2 n$ tăng chậm hơn n.
  - Suy ra $f(n)$ tăng chậm hơn $g(n)$.

Ví dụ về tốc độ tăng của các hàm

- Xét một máy tính thực hiện được 1.000.000 thao tác cơ bản trong một giây.
- Khi thời gian chạy vượt quá 10^{25} năm, ta viết "very long".

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

4. Các ví dụ phân tích thuật toán

Vòng lặp

```
1  for (i = 0; i < n; i++)
2  {
3       x = a[i] / 2;
4       a[i] = x + 1;
5  }
```

- Có 4 thao tác cơ bản ở các dòng 3 và 4, gồm 2 phép gán, 1 phép chia và 1 phép cộng.
- Cả 4 thao tác cơ bản đó được lặp lại n lần.
- Thời gian chạy: t(n) = 4n = O(n)

Chú ý: Ở đây, ta bỏ qua 3 thao tác cơ bản điều khiển quá trình lặp ở dòng 1. Kết quả phân tích thuật toán sẽ không thay đổi nếu tính thêm cả 3 thao tác cơ bản đó.

Vòng lặp có câu lệnh break

```
1  for (i = 0; i < n; i++)
2  {
3      x = a[i] / 2;
4      a[i] = x + 1;
5      if (a[i] > 10) break;
6  }
```

- Có 5 thao tác cơ bản ở các dòng 3, 4, 5, gồm 2 phép gán, 1 phép chia, 1 phép cộng và 1 phép so sánh
- Không thể đếm chính xác số lần thực hiện 5 thao tác cơ bản đó vì ta không biết khi nào điều kiện a[i] > 10 xảy ra.
- Trong trường hợp tồi nhất, tức là điều kiện a[i] > 10 xảy ra ở bước lặp cuối cùng hoặc không bao giờ xảy ra, cả 5 thao tác cơ bản được lặp lại n lần.
- Thời gian chạy trong trường hợp tồi nhất: t(n) = 5n = O(n)

Các vòng lặp tuần tự

```
for (i = 0; i < n; i++)
{
    ... // giả sử có 3 thao tác cơ bản ở đây
}
for (i = 0; i < n; i++)
{
    ... // giả sử có 5 thao tác cơ bản ở đây
}
```

- Chỉ cần cộng thời gian chạy của các vòng lặp.
- Thời gian chạy tổng thể: t(n) = 3n + 5n = 8n = O(n)

Các vòng lặp lồng nhau

```
for (i = 0; i < n; i++)
{
    ... // giả sử có 2 thao tác cơ bản ở đây
    for (j = 0; j < n; j++)
        ... // giả sử có 3 thao tác cơ bản ở đây
}
```

- Phân tích các vòng lặp từ trong ra ngoài:
  - Vòng lặp bên trong thực hiện 3n thao tác cơ bản.
  - Mỗi bước lặp của vòng lặp bên ngoài thực hiện 2 + 3n thao tác cơ bản.
- Thời gian chạy tổng thể: $t(n) = (2 + 3n)n = 3n^2 + 2n = O(n^2)$

Câu lệnh if-else

```
1   if (x > 0)
2       i = 0;
3   else
4       for (j = 0; j < n; j++)
5           a[j] = j;
```

- Có 3 thao tác cơ bản: x > 0 (dòng 1), i = 0 (dòng 2) và a[j] = j (dòng 5).
- Trong trường hợp tồi nhất, tức là điều kiện x > 0 sai:
  - Phép gán i = 0 chạy 0 lần.
  - Phép gán a[j] = j chạy n lần (vì nằm trong vòng lặp).
- Thời gian chạy trong trường hợp tồi nhất: t(n) = 1 + n = O(n)

Hàm đệ quy

```
1  long factorial(int n)
2  {
3      if (n <= 1)
4          return 1;
5      else
6          return n * factorial(n - 1);
7  }
```

- Nếu n = 1, chỉ mất 1 phép so sánh n <= 1 ở dòng 3.
- Nếu n > 1:
  - Dòng 3 có 1 phép so sánh (và bị sai nên nhảy đến dòng 6).
  - Dòng 6 có 1 phép nhân, 1 phép trừ và 1 lời gọi hàm đệ quy tốn thời gian t(n-1).

Hàm đệ quy (tiếp)

- Suy ra thời gian chạy của thuật toán:

$t(1) = 1$ (với $n = 1$)

$t(n) = 3 + t(n - 1)$ (với $n > 1$)

$= 3 + 3 + t(n - 2)$

$= 3 + 3 + 3 + t(n - 3)$

...

$= 3k + t(n - k)$

- Chọn $k = n - 1$, khi đó:

$t(n) = 3(n - 1) + t(1) = 3n - 2 = O(n)$

Tìm kiếm tuần tự

```
for (i = 0; i < n; i++)
{
    if (a[i] == x) return i;
}
return -1;
```

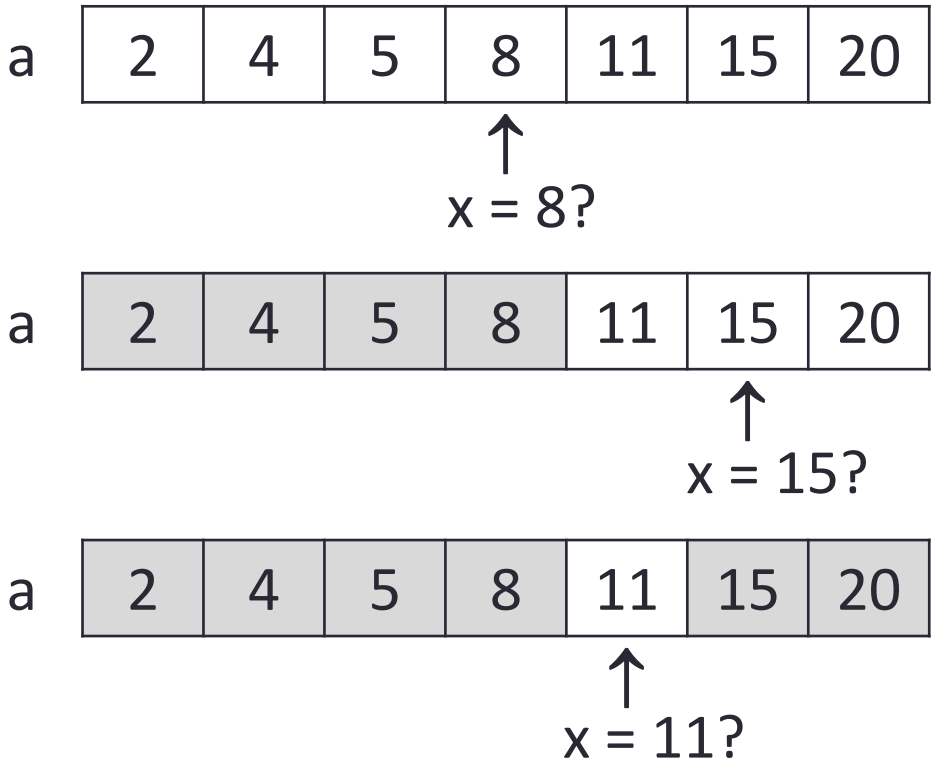
- Trong trường hợp tồi nhất, tức là x nằm ở cuối mảng hoặc x không có trong mảng, ta phải thực hiện n phép so sánh a[i] == x.
- Thời gian chạy trong trường hợp tồi nhất: t(n) = n = O(n)

Tìm kiếm nhị phân

- Cho mảng a đã sắp xếp tăng dần.
- Tìm x trong mảng a:
  - So sánh x với phần tử ở chính giữa mảng a[mid] (mid là vị trí chính giữa).
  - Nếu x < a[mid], tìm x ở nửa bên trái của mảng.
  - Nếu x > a[mid], tìm x ở nửa bên phải của mảng.
  - Nếu x = a[mid], báo cáo vị trí tìm được x là mid.
  - Nếu không còn phần tử nào để xét, báo cáo không tìm được x.

Tìm kiếm nhị phân – ví dụ

Giả sử x = 11 và ta phải tìm x trong mảng a bên dưới



Tìm kiếm nhị phân – mã giả

```
function binarySearch(a, n, x)
{
    left ← 0, right ← n – 1
    while (left ≤ right)
    {
        mid ← (left + right) / 2
        if      (x < a[mid])  right ← mid – 1
        else if (x > a[mid])  left  ← mid + 1
        else                  return mid
    }
    return –1
}
```

Tìm kiếm nhị phân – phân tích

- Nếu n = 1, chỉ mất một phép so sánh x với phần tử duy nhất của mảng.
- Nếu n > 1, mất một phép so sánh x với phần tử chính giữa mảng, sau đó là mất thời gian tìm x trong một nửa (trái hoặc phải) của mảng.
- Suy ra thời gian chạy của thuật toán:

$t(1) = 1$ (với n = 1)

$t(n) = 1 + t(n/2)$ (với n > 1)

$= 1 + 1 + t(n/4)$

$= 1 + 1 + 1 + t(n/8)$

...

$= k + t(n/2^k)$

- Chọn k = log n, khi đó:

$t(n) = \log n + t(1) = \log n + 1 = O(\log n)$

Bài tập 1

Xét các thuật toán có thời gian chạy như bên dưới. Hỏi mỗi thuật toán chậm đi bao nhiêu lần khi gấp đôi kích thước đầu vào?

a. n
b. n^2
c. n^3
d. $100n^2$
e. $n \log n$
f. 2^n

Bài tập 2

Sắp xếp các hàm sau đây theo thứ tự tăng dần của tốc độ tăng; nghĩa là, hàm f(n) sẽ được xếp trước hàm g(n) nếu f(n) = O(g(n)).

$f_1(n) = n^{2,5}$

$f_2(n) = \sqrt{2n}$

$f_3(n) = n + 10$

$f_4(n) = 10^n$

$f_5(n) = 100^n$

$f_6(n) = n^2 \log n$

Bài tập 3

Phân tích thời gian chạy (dùng ký hiệu O) của thuật toán tìm phần tử lớn nhất.

```
max ← a0
for i ← 1 to n-1
    if (ai > max)
        max ← ai
```

Bài tập 4

Phân tích thời gian chạy (dùng ký hiệu O) của các đoạn chương trình C++ sau đây.

(a)
```
sum = 0;
for (i = 0; i < n; i++)
    sum++;
```

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

Bài tập 4

Phân tích thời gian chạy (dùng ký hiệu O) của các đoạn chương trình C++ sau đây.

(c)
```
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < n*n; j++)
        sum++;
```

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

Bài tập 4

Phân tích thời gian chạy (dùng ký hiệu O) của các đoạn chương trình C++ sau đây.

(e)
```
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < i*i; j++)
        for (k = 0; k < j; k++)
            sum++;
```

(f)
```
sum = 0;
for (i = 0; i < n; i++)
    for (j = 0; j < i*i; j++)
        if (j % i == 0)
            for (k = 0; k < j; k++)
                sum++;
```