Kỹ thuật phân tích giải thuật

Phạm Nguyên Khang, Đỗ Thanh Nghị
Khoa CNTT – Đại học Cần Thơ
{pnkhang,dtnghi}@cit.ctu.edu.vn

Nội dung

- Tại sao cần phải phân tích giải thuật ?
- Tiêu chuẩn đánh giá giải thuật
- Phương pháp đánh giá
- Bài tập

Sự cần thiết phải phân tích giải thuật

- Đánh giá giải thuật
  - Tính đúng đắn
    - Chạy trên dữ liệu thử
    - Chứng minh lý thuyết (bằng toán học chẳng hạn)
  - Tính đơn giản
  - Tính nhanh chóng (thời gian thực thi)
    - Quan trọng khi chương trình được thực thi nhiều lần
    - Hiệu quả thời gian thực thi

Thời gian thực hiện của chương trình

- Đo thời gian thực hiện chương trình
  - Lập trình và đo thời gian thực hiện
  - Phụ thuộc vào tập lệnh của máy tính
  - Kỹ năng của người lập trình
  - Dữ liệu đầu vào

➔Tính độ phức tạp thời gian thực hiện của giải thuật = độ đo sự thực thi của giải thuật

Thời gian thực hiện của chương trình

- Đo thời gian thực hiện:
  - Hàm T(n) ≥ 0, với n là kích thước (độ lớn) của đầu vào
  - Ví dụ: T(n) = 3n
  - Đơn vị tính: số lệnh cơ bản, số chỉ thị, ...
  - Thời gian thực hiện trong các trường hợp: tốt nhất, xấu nhất, trung bình

➔So sánh T1(n) và T2(n)

Thời gian thực hiện của chương trình

- Tỷ suất tăng (growth rate):
  - T(n) có tỷ suất tăng f(n) nếu tồn tại hằng $C > 0$ và n0 sao cho $T(n) \leq Cf(n)\ \forall n \geq n0$
  - Cho một hàm không âm T(n), luôn tồn tại tỷ suất tăng f(n) của nó
  - Ví dụ: $T(0) = 1$, $T(1) = 4$, $T(n) = (n+1)^2$, ta có:

  $f(n) = n^2$ (với $C = 4$, $n0 = 1$)

Thời gian thực hiện của chương trình

??? Chứng minh rằng:

- Tỷ suất tăng của $T(n) = 3n^3 + 2n$ là n^3
  - Chọn C = ?, chọn n0 = ?
  - Chứng minh bằng quy nạp

  $T(n) \leq Cf(n) \ \forall n \geq n0$

Thời gian thực hiện của chương trình

??? Chứng minh rằng:

- Tỷ suất tăng của $T(n) = 3n^3 + 2n$ là n^3
  - Chọn C = 5, chọn n0 = 0
  - Chứng minh bằng quy nạp

    $3n^3 + 2n \leq 5n^3 \ \forall n \geq 0$

- Quy tắc:

  T(n) là đa thức của n thì tỷ suất tăng là bậc cao nhất của n

Độ phức tạp giải thuật

- Cho 2 giải thuật
  - P1 có $T1(n) = 100n^2$
  - P2 có $T2(n) = 5n^3$

??? Giải thuật nào chạy nhanh hơn

Xét nếu $n \leq 20$ thì $T1(n) \geq T2(n)$

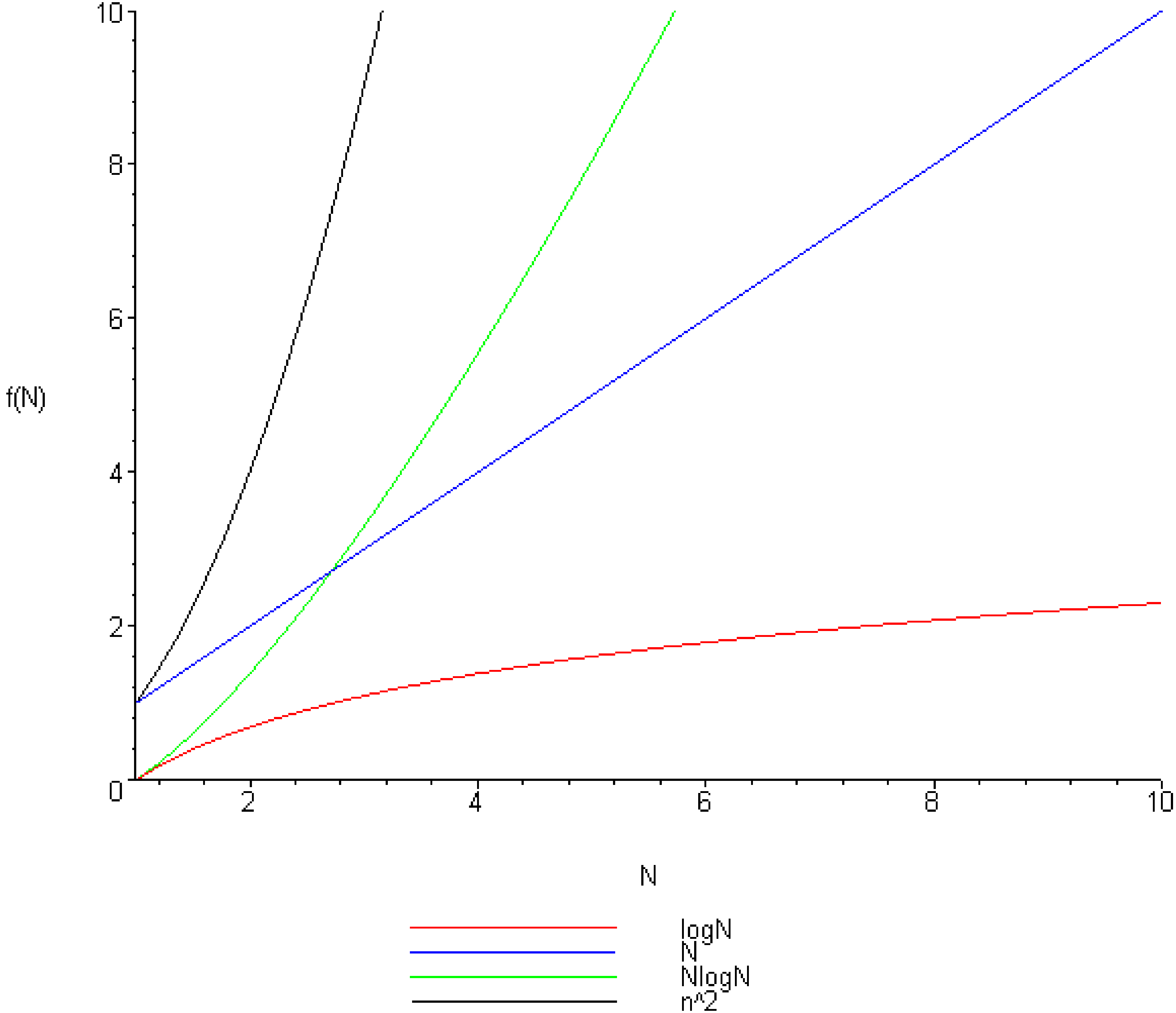
Xét nếu $n > 20$ thì $T1(n) < T2(n)$

➔ So sánh tỷ suất tăng hơn là so sánh trực tiếp các hàm T(n)

Độ phức tạp giải thuật

- Ký pháp Ô lớn (big O)
  - Nếu T(n) có tỷ suất tăng f(n) ➔ T(n) có độ phức tạp là f(n) và ký hiệu là O(f(n)), đọc là "ô f(n)".
- Ví dụ: $T(n) = (n + 1)^2$ có độ phức tạp $O(n^2)$
- Tính chất
  - O(cf(n)) = O(f(n)), c: hằng số
  - O(C) = O(1)
- Độ phức tạp của giải thuật: hàm chặn trên của thời gian
- Một số hàm thể hiện độ phức tạp thường gặp:

  $\log_2 n$, $n\log_2 n$, n^2, n^3, $2n$, $n!$, n^n, …


10
8
6
4
2
0
f(N)
2
4
6
8
10
N
logN
N
NlogN
n^2

Cách tính độ phức tạp

- Cho 2 chương trình:
  - P1 có thời gian thực hiện $T1(n) = O(f1(n))$
  - P2 có thời gian thực hiện $T2(n) = O(f2(n))$
- Quy tắc cộng:
  - Thời gian thực thi P1 và P2 nối tiếp nhau sẽ là:
    - $T(n) = T1(n) + T2(n) = O(\max(f1(n), f2(n))$
- Quy tắc nhân:
  - Thời gian thực thi P1 và P2 lồng nhau (vd: vòng lặp lồng nhau chẳng hạn):
    - $T(n) = T1(n) \times T2(n) = O(f1(n) \times f2(n))$

Cách tính độ phức tạp

- Quy tắc nhân:

```
for (i=1; i<= n; i++)
    for (j=1; j<=n; j++) {
        thực hiện công việc O(1)
    }
```

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

Cách tính độ phức tạp

```
for (i=1; i< n; i++)
    for (j=i+1; j<=n; j++) {
        thực hiện công việc O(1)
    }
```

Áp dụng quy tắc nhân được không?

T(n) ????

Cách tính độ phức tạp

- Quy tắc tổng quát:
  - Đọc (read, scanf), ghi (write, printf), lệnh gán, so sánh: thời gian là hằng số hay O(1)
  - Lệnh if:

    if (điều kiện)
        lệnh 1
    else
        lệnh 2

      - max (lệnh 1, lệnh 2) + điều kiện
  - Vòng lặp: tổng thời gian thực hiện thân vòng lặp
      - Trong trường hợp không xác định được số lần lặp ta phải lấy số lần lặp trong trường hợp xấu nhất.

Cách tính độ phức tạp

- Phương pháp thực hiện:
  - Xác định đầu vào: thường ký hiệu là n
  - Cách 1: dùng cho tất cả các loại chương trình
    - Tính thời gian thực hiện T(n) cho toàn bộ chương trình ➔ O(f(n)) từ T(n)
  - Cách 2: không áp dụng cho chương trình đệ quy
    - Chia chương trình nhiều đoạn nhỏ
    - Tính T(n) và O(f(n)) cho từng đoạn
    - Áp dụng quy tắc cộng, quy tắc nhân để có O(f(n)) cho cả chương trình

Cách tính độ phức tạp

Ví dụ:

```
void sap_xep(int a[], int n) {
  int tam;
  for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
      if (a[j] < a[i]) {
        tam  = a[i];
        a[i] = a[j];
        a[j] = tam;
      }
}
```

Cách tính độ phức tạp

Ví dụ:

```
int tim_kiem(int x, int a[], int n) {
  int found, i;
  found = 0;
  i = 0;
  while (i < n && !found)
    if (a[i] == x)
      found = 1;
    else
      i = i+1;
  return i;
}
```

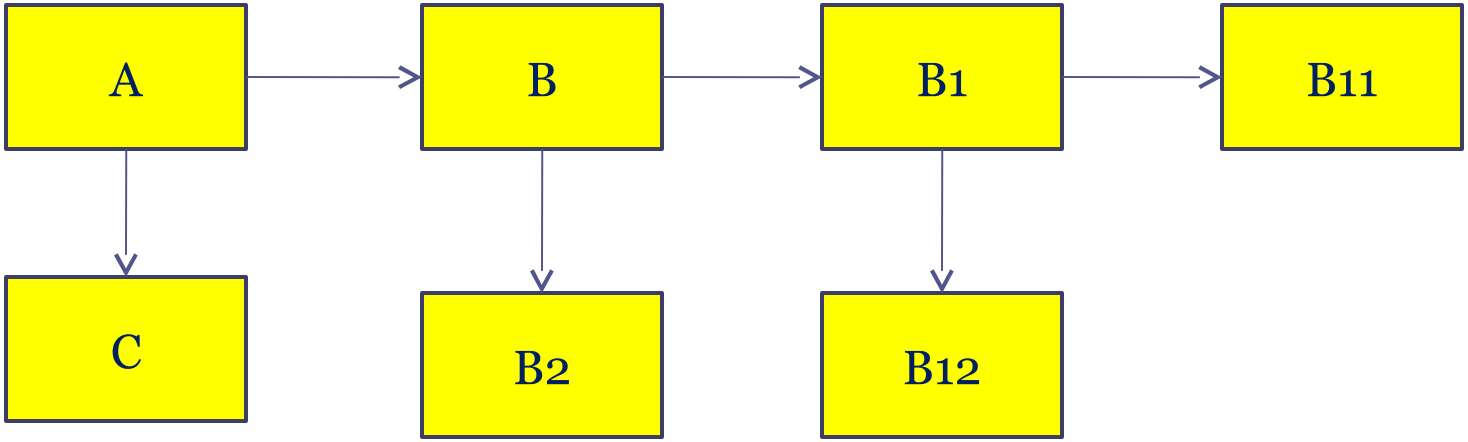
Cách tính độ phức tạp

Ví dụ:

```
int tim_kiem_nhi_phan(int x, int a[], int n) {
  int i = 0, j = n - 1;
  while (i <= j) {
    int m = (i + j)/2;
    if (x == a[m])
      return m;
    if (x < a[m])
      j = m - 1;
    else
      i = m + 1;
  }
  return -1; // khong tim thay
}
```

Cách tính độ phức tạp

- Chương trình có gọi chương trình con (không đệ quy)
- Quy tắc: tính từ trong ra ngoài



- C, B2, B12, B11
- B1
- B
- A

Cách tính độ phức tạp

- Chương trình đệ quy
  - Lập phương trình đệ quy T(n)
  - Giải phương trình đệ quy tìm nghiệm
  - Suy ra tỷ suất tăng f(n) hay O(f(n))

Ví dụ:

```
int giai_thua(int n) {
  if (n == 0)
    return 1;
  else
    return n * giai_thua(n - 1);
}
```

Giải phương trình đệ quy

- Phương pháp truy hồi
  - Triển khai T(n) theo T(n - 1) rồi T(n - 2) ... cho đến T(1) hoặc T(0)
  - Suy ra nghiệm
- Phương pháp đoán nghiệm
  - Dự đoán nghiệm f(n)
  - Áp dụng định nghĩa tỷ suất tăng và chứng minh f(n) là tỷ suất tăng của T(n)
- Áp dụng công thức đối với lớp phương trình đệ quy đã có lời giải

Phương pháp truy hồi

- Triển khai T(n) theo T(n-1) rồi đến T(n-2) tiếp đến … cho đến T(1)

Ví dụ 1

- Giải phương trình đệ quy:

  $T(1) = C1$

  $T(n) = 2T(n - 1) + C2$

- Ta có:

  $T(n) = 2T(n - 1) + C2$

  $= 2(2T(n - 2) + C2) + C2 = 2^2T(n - 2) + 2C2 + C2$

  $= 2^2(2T(n - 3) + C2) + 2C2 + C2$

  $= 2^3T(n - 3) + 2^2C2 + 2C2 + C2$

  $= \ldots$

  $= 2^kT(n - k) + 2^{k-1}C2 + 2^{k-2}C2 + \ldots + 2C2 + C2$

  – Quá trình dừng lại khi $n - k = 1$ hay $k = n - 1$, khi đó:

  $T(n) = 2^{n-1}T(1) + C2\,(2^{n-1} - 1)$

  $= C1\,2^{n-1} + C2\,(2^{n-1} - 1) = O(2^n)$

Ví dụ 2

Giải phương trình

$$C_n = C_{n-1} + n \qquad n \geq 2$$

$$C_1 = 1$$

Triển khai phương trình

$$
\begin{aligned}
C_n &= C_{n-1} + n \\
&= C_{n-2} + (n-1) + n \\
&= C_{n-3} + (n-2) + (n-1) + n
\end{aligned}
$$

...

$$
\begin{aligned}
&= C_1 + 2 + \ldots + (n-2) + (n-1) + n \\
&= 1 + 2 + \ldots + (n-1) + n \\
&= n(n+1)/2 \\
&= n^2/2
\end{aligned}
$$

Ví dụ 3

Giải phương trình

$$C_n = C_{n/2} + 1 \qquad n \geq 2$$

$$C_1 = 0$$

Đặt $n = 2^k$

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

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

$= C(2^{k-3}) + 3$

.

.

$= C(2^0) + k$

$= C_1 + k = k$

$C_n = k = \log n$

$C_n \approx \log n$

Ví dụ 4

Giải phương trình

$$C_n = 2C_{n/2} + n \quad \text{for} \quad n \geq 2$$

$$C_1 = 0$$

Đặt $n = 2^k$

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

$$C(2^k)/2^k = C(2^{k-1})/\ 2^{k-1} + 1$$

$$= C(2^{k-2})/\ 2^{k-2} + 1 + 1$$

.

.

$$= k$$

$$\Rightarrow C(2^k) = k.2^k$$

$$C_n = n\log n$$

Ví dụ 5

Giải phương trình

$$C(n) = 2C(n/2) + 1 \qquad \text{for } n \geq 2$$

$$C(1) = 0$$

Đặt $n = 2^k$

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

$$C(2^k)/ 2^k = 2C(2^{k-1})/ 2^k + 1/2^k$$

$$= C(2^{k-1})/ 2^{k-1} + 1/2^k$$

$$=[C(2^{k-2})/ 2^{k-2} + 1/2^{k-1}]+ 1/2^k$$

.

.

$$=C(2^{k-i})/ 2^{k-i} + 1/2^{k-i+1} + \ldots + 1/2^k$$

Cuối cùng, khi i = n -1, ta được:

$$C(2^k)/2^k = C(2)/2 + 1/4 + \ldots + 1/2^k = 1/2 + 1/4 + \ldots + 1/2^k \approx 1$$

$$\Rightarrow C(2^k) \approx 2^k = n$$

Ví dụ 6

Giải phương trình

$$C_n = 2C_{n-1} + 1 \qquad n \geq 2$$

$$C_1 = 1$$

$$C_n + 1 = 2C_{n-1} + 2 = 2\,(C_{n-1} + 1)$$

$$= 2\,(2\,(C_{n-2} + 1)) = 2^2(C_{n-2} + 1)$$

...

$$= 2^{n-1}(C_1 + 1) = 2^{n-1}(1 + 1) = 2^n$$

$$C_n = 2^n - 1$$

Ví dụ 7

Giải phương trình

$$C(n) = 2C(\sqrt{n}) + logn$$

Đặt m = logn => n = 2^m
C(2^m) = 2C($2^{m/2}$) + m
Đặt S(m) = C(2^m)
S(m) = 2S(m/2) + m = mlogm
C(n) = logn loglogn

Chuỗi thông dụng

$S = 1 + 2 + 3 + \ldots + n = n(n+1)/2 \approx n^2/2$

$S = 1 + 2^2 + 3^2 + \ldots + n^2 = n(n+1)(2n+1)/6 \approx n^3/3$

$S = 1 + a + a^2 + a^3 + \ldots + a^n = (a^{n+1} - 1)/(a-1)$

Nếu $0 < a < 1$ thì

$S \leq 1/(1-a)$

và khi $n \to \infty$ thì

S tiến về $1/(1-a)$

$S = 1 + 1/2 + 1/3 + \ldots + 1/n = \ln(n) + \gamma$

Hằng số Euler **$\gamma \approx 0.577215665$**

$S = 1 + 1/2 + 1/4 + 1/8 + \ldots + 1/2^n + \ldots \approx 2$

Bài tập

- Tính độ phức tạp của lời giải đệ quy
  - Tính giai thừa của **n**
  - Tháp Hà nội với số tầng tháp là **n**
  - Tìm kiếm nhị phân của dãy gồm **n** số được sắp theo thứ tự tăng dần

Phương pháp đoán nghiệm

- Thử đoán 1 nghiệm f(n)
- Sau đó tìm cách chứng minh $T(n) \leq f(n)$
  - Chứng minh mình bằng quy nạp
- Thông thường ta chọn f(n) có dạng: n, logn, nlogn, 2^n, …

Lời giải tổng quát

- Giải thuật chia để trị:
  - Phân rã bài toán lớn thành các bài toán con
    - Một bài toán lớn có kích thước n, thành a bài toán con có kích thước n/b
  - Tổng hợp các lời giải của các bài toán con để có được lời giải của bài toán lớn
    - Thời gian tổng hợp a bài toán con tốn d(n) thời gian
- Phương trình đệ quy cho giải thuật trên:
  - $T(1) = 1$
  - $T(n) = aT(n/b) + d(n)$

Lời giải tổng quát

- Áp dụng phương pháp truy hồi:

$$
\begin{aligned}
T(n) &= aT(n/b) + d(n) \\
&= a[aT(n/b/b) + d(n/b)] + d(n) \\
&= a^2T(n/b^2) + ad(n/b) + d(n) \\
&= a^2[aT(n/b^3) + d(n/b^2)] + ad(n/b) + d(n) \\
&= a^3T(n/b^3) + a^2d(n/b^2) + ad(n/b) + d(n) \\
&= \ldots \\
&= a^kT(n/b^k) + \sum a^i d(n/b^i)
\end{aligned}
$$

  - Quá trình kết thúc khi $n/b^k = 1$ hay $k = \log_b n$

    $T(n) = a^k + \sum a^i d(n/b^i)$

Lời giải tổng quát

- Nghiệm thuần nhất (homogeneous solutions):

$$a^k = n^{\log_b a}$$

- d(n): hàm tiến triển (driving function)
- Nghiệm chính xác sẽ là nghiệm chính xác nếu d(n) = 0, với mọi n
- Nếu d(n) > 0, ta có nghiệm riêng:

$$\sum_{i=0}^{k-1} a^i d\left(\frac{n}{b^i}\right) = \sum_{i=0}^{k-1} a^i d\left(\frac{b^k}{b^i}\right) = \sum_{i=0}^{k-1} a^i d(b^{k-i})$$

Lời giải tổng quát

- Nếu nghiệm thuần nhất lớn nghiệm riêng thì độ phức tạp là nghiệm thuần nhất
- Nếu nghiệm riêng lớn hơn nghiệm thuần nhất thì độ phức tạp là nghiệm riêng

- Tuy nhiên, tính nghiệm không phải lúc nào cũng dễ!

Lời giải tổng quát

- Ta sẽ tính nghiệm riêng trong trường hợp d(n) có dạng đặc biệt
- Hàm nhân, hàm có tính chất nhân (multiplicative function):
  - Hàm d(n) có tính nhân nếu và chỉ nếu d(x.y) = d(x).d(y)
  - Ví dụ:
    - $d(n) = n^2$ là hàm nhân vì $d(x.y) = (x.y)^2 = x^2.y^2 = d(x).d(y)$
    - $d(n) = 3n^2$ không phải là hàm nhân

Lời giải tổng quát

- Nếu d(n) là hàm nhân, ta có nghiệm riêng:

$$\sum_{i=0}^{k-1} a^i d(b^{k-i}) = \sum_{i=0}^{k-1} a^i \left[d(b)\right]^{k-i} = \left[d(b)\right]^k \sum_{i=0}^{k-1} \left[\frac{a}{d(b)}\right]^i$$

$$= \frac{a^k - \left[d(b)\right]^k}{\dfrac{a}{d(b)} - 1}$$

Lời giải tổng quát

- Nếu a > d(b), $a^k > [d(b)]^k$

$$T(n) = O(a^k) = O(a^{\log_b^n}) = O(n^{\log_b^a})$$

- Nếu a < d(b)

$$T(n) = O(d(b)^k) = O(d(b)^{\log_b^n}) = O(n^{\log_b^{d(b)}})$$

- Nếu a = d(b)

$$\sum_{i=0}^{k-1} a^i d(b^{k-i}) = \sum_{i=0}^{k-1} a^i [d(b)]^{k-i} = [d(b)]^k \sum_{i=0}^{k-1} \left[\frac{a}{d(b)}\right]^i$$

$$= d(b)^k k = a^k k$$

$$T(n) = O(n^{\log_b^a} \log n)$$

Bài tập

- Giải các phương trình đệ quy sau với T(1) = 1

1/- $T(n) = 4T(\frac{n}{2}) + n$

2/- $T(n) = 4T(\frac{n}{2}) + n^2$

3/- $T(n) = 4T(\frac{n}{2}) + n^3$

4/ $T(1) = 1$

$$T(n) = 2T(\frac{n}{2}) + n\log n$$

Ví dụ: GPT với T(1) = 1 và

$$1/\ \ T(n) = 4T\left(\frac{n}{2}\right) + n$$

- Phương trình có dạng phương trình tổng quát.
- d(n)=n là hàm nhân.
- a = 4 và b = 2.
- d(b) = b = 2 < a.
- $T(n) = O(n^{\log_b a}) = O(n^{\log 4}) = O(n^2)$.

Ví dụ: GPT với T(1) = 1 và

$$2/\quad T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

- Phương trình có dạng phương trình tổng quát.
- $d(n)=n^2$ là hàm nhân.
- $a = 4$ và $b = 2$.
- $d(b) = b^2 = 4 = a$.
- $T(n) = O(n^{\log_b a}\log_b n)$

  $= O(n^{\log 4}\log n) = O(n^2\log n)$.

Ví dụ: GPT với T(1) = 1 và

$$3/\ \ T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

- Phương trình có dạng tổng quát.
- $d(n)=n^3$ là hàm nhân.
- $a = 4$ và $b = 2$.
- $d(b) = b^3 = 8 > a$.
- $T(n) = O(n^{\log_b d(b)}) = O(n^{\log 8}) = O(n^3)$.

Bài tập

Bài 2: Giải các phương trình đệ quy sau với $T(1) = 1$ và

a) $T(n) = 3T(n/2) + n$

b) $T(n) = 3T(n/2) + n^2$

c) $T(n) = 8T(n/2) + n^3$

Bài 3: Giải các phương trình đệ quy sau với $T(1) = 1$ và

a) $T(n) = 4T(n/3) + n$

b) $T(n) = 4T(n/3) + n^2$

c) $T(n) = 9T(n/3) + n^2$

Bài tập

Bài 4: Giải các phương trình đệ quy sau với $T(1) = 1$ và

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

b) $T(n) = 2T(n/2) + \log n$

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

d) $T(n) = 2T(n/2) + n^2$

Bài 5: Giải các phương trình đệ quy sau bằng phương pháp đoán nghiệm:

a) $T(1) = 2$ và $T(n) = 2T(n-1) + 1$ với $n > 1$

b) $T(1) = 1$ và $T(n) = 2T(n-1) + n$ với $n > 1$

Ví dụ: GPT với T(1) = 1 và

$$T(n) = 2T\left(\frac{n}{2}\right) + n\log n$$

- PT thuộc dạng phương trình tổng quát nhưng d(n) = nlogn không phải là một hàm nhân.
- NTN = $n^{\log_b a} = n^{\log 2} = n$
- Do d(n) = nlogn không phải là hàm nhân nên ta phải tính nghiệm riêng bằng cách xét trực tiếp

Ví dụ (tt)

$$NR = \sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right) = \sum_{j=0}^{k-1} 2^j 2^{k-j} \log 2^{k-j}$$

$$NR = 2^k \sum_{j=0}^{k-1} (k - j) = 2^k \frac{k(k+1)}{2} = O(2^k k^2)$$

- Theo giải phương trình đệ quy tổng quát thì $n = b^k$ nên $k = \log_b n$, ở đây do $b = 2$ nên
  $2^k = n$ và $k = \log n$,
- $NR = O(n\log^2 n) > NTN$
- $T(n) = O(n\log^2 n)$.

BT4-1: GPT với T(1) = 1 và

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

- Phương trình có dạng phương trình tổng quát
- d(n)=1 là hàm nhân
- a = 1 và b = 2
- d(b) = 1 = a
- $T(n) = O(n^{\log_b a} \log_b n) = O(n^{\log 1} \log n) = O(\log n)$

BT4-2: GPT với T(1) = 1 và

$$T(n) = 2T\left(\frac{n}{2}\right) + \log n$$

- Phương trình có dạng tổng quát
- d(n)=logn không phải là hàm nhân
- NTN = $O(n^{\log_b a}) = O(n^{\log 2}) = O(n)$
- Tính trực tiếp nghiệm riêng

Ví dụ (tt)

$$NR = \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j \log 2^{k-j}$$

$$NR = \sum_{j=0}^{k-1} 2^j (k - j) = \sum_{j=0}^{k-1} k2^j - \sum_{j=0}^{k-1} j2^j$$

$$NR = O(k\sum_{j=0}^{k-1} 2^j) = O(k\frac{2^k - 1}{2 - 1})$$

$$NR = O(k2^k) = O(n\log n) > n = NTN$$

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

Bài tập

Bài 8: Xét công thức truy toán để tính số tổ hợp chập k của n như sau:

$$C_n^k = \begin{cases} 1 \quad \text{nêu } k = 0 \text{ hoac } k = n \\ C_{n-1}^{k-1} + C_{n-1}^{k} \text{ nêu } 0 < k < n \end{cases}$$

a) Viết một hàm đệ quy để tính số tổ hợp chập k của n.

b) Tính thời gian thực hiện của giải thuật nói trên.

Bài tập

Tính độ phức tạp thời gian của đoạn chương trình sau theo *n*:

```
int max(int a[], int n) {
  if (n == 1)
    return a[0];
  if  (a[n-1] > max(a, n - 1))
    return a[n-1];
  return max(a, n-1);
}
```