GIỚI THIỆU PHÂN TÍCH THUẬT TOÁN

Bùi Tiến Lên

01/01/2017

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Phân tích thuật toán

Mục tiêu

- Hiểu được sự cần thiết về phân tích thuật toán
- Nắm được các tiêu chuẩn để đánh giá một giải thuật
- Hiểu được các khái niệm về độ phức tạp thuật toán

Phân tích thuật toán (cont.)

Để làm gì?

- Cùng một vấn đề, có thể giải quyết bằng nhiều giải thuật khác nhau
- Lựa chọn một giải thuật tốt "nhất" trong các giải thuật cho một bài toán
- Cải tiến giải thuật để có một giải thuật tốt hơn

Phân tích thuật toán (cont.)

Các tiêu chí đánh giá thuật toán

- Một thuật toán được xem là **đúng** nếu
  - **Tính đúng đắn**: Thuật toán phải chạy **đúng**
  - **Tính hữu hạn**: Thuật toán phải dừng sau một số bước hữu hạn
- Một thuật toán được xem là **tốt** nếu
  - **Bộ nhớ**: Sử dụng **ít** bộ nhớ (liên quan đến cấu trúc dữ liệu)
  - **Thời gian**: Thực hiện **nhanh**

Thời gian thực hiện chương trình

- Các **yếu tố** ảnh hưởng đến *thời gian thực hiện chương trình*
  - Cấu hình máy tính: tốc độ CPU, kích thước bộ nhớ...
  - Ngôn ngữ lập trình
  - Cấu trúc dữ liệu
  - Cài đặt chi tiết
  - ...

Thời gian thực hiện chương trình (cont.)

Định nghĩa 1

- **Thời gian thực hiện** hay **chi phí thực hiện** hay **độ phức tạp chương trình** là hàm của kích thước dữ liệu vào, ký hiệu $T(n)$ trong đó n là kích thước hay độ lớn của dữ liệu vào

Thời gian thực hiện chương trình (cont.)

Lưu ý

- Thời gian thực hiện chương trình là một hàm không âm, $T(n) \geq 0, \forall n \geq 0$.
- Đơn vị của $T(n)$ không phải là đơn vị thời gian vật lý như giờ, phút, giây, ... mà được đo bởi **số các lệnh cơ bản** (*basic operations*) được thực hiện trên **một máy tính lý tưởng**
- Các lệnh cơ bản là
  - các phép toán so sánh
  - các phép toán gán
  - các phép toán số học

Thời gian thực hiện chương trình (cont.)

Khi xác định $T(n)$ cố gắng đơn giản hóa các lệnh cơ bản

Chương trình 1: Tính tổng n số tự nhiên đầu tiên

```
1   sum = 0;
2   for (i = 0; i < n; i++)
3       sum = sum + i;
```

- số phép gán: T_1
- số phép so sánh: T_2
- số phép cộng và tăng: T_3
- Thời gian thực hiện thuật toán là $T(n) = T_1 + T_2 + T_3$

Phương pháp xác định thời gian thực hiện của chương trình

Các phương pháp tiếp cận để xác định

- Phương pháp **thực nghiệm**
- Phương pháp **toán học**
  - Phương pháp đếm
  - Phương pháp truy hồi
  - Phương pháp hàm sinh

Phương pháp xác định thời gian thực hiện của chương trình (cont.)

Do hàm $T(n)$ không chỉ phụ thuộc vào n mà còn phụ thuộc vào cấu trúc của dữ liệu. Do đó, trong phương pháp toán học, khi phân tích thuật toán người ta thường phân tích dựa trên 3 tình huống:

- **Trường hợp tốt nhất** (*best case*): Không phản ánh được cái "tốt" thật sự của chương trình
- **Trường hợp trung bình** (*average case*): Rất khó xác định chính xác
- **Trường hợp xấu nhất** (*worst case*): Cho một sự "bảo đảm".

Lưu ý

Trong thực tế, người lập trình chỉ nên đánh giá $T(n)$ cho trường hợp xấu nhất hoặc trung bình

Phương pháp xác định thời gian thực hiện của chương trình (cont.)

Rất khó có thể tính được chính xác $T(n)$. Do đó, $T(n)$ thường được thể hiện qua các **hàm ước lượng**. Có 3 cách ước lượng cơ bản cho $T(n)$

- Ước lượng O
- Ước lượng Ω
- Ước lượng Θ

Ước lượng O

Lịch sử của ký hiệu

- Ký hiệu Big-0 được giới thiệu bởi Paul Bachmann [Apostol, 1976].
- Ký hiệu này sau đó được phổ biến rộng rãi bởi nhà toán học Edmund Landau, nên còn được gọi là ký hiệu Landau [Landau et al., 1958].
- Donald Knuth là người đưa ký hiệu này vào ngành Khoa học máy tính [Knuth, 1976].

Ước lượng O (cont.)

Định nghĩa 2

Cho $T(n)$ và $g(n)$ là hai hàm số. Ta nói $T(n) = O(g(n))$ nếu tồn tại các số dương c và K sao cho:

$$T(n) \leq cg(n), \forall n \geq K \tag{1}$$

- Cách đọc: $T(n)$ là "big-o" của $g(n)$
- Ý nghĩa: $g(n)$ là giới hạn trên của $T(n)$

Ước lượng O (cont.)

Lưu ý

Khi áp dụng ước lượng O vào việc ước lượng độ phức tạp của giải thuật, ta nên chọn dạng $g(n)$ sao cho càng đơn giản càng tốt. Nhờ vậy, ta có thể ước lượng độ phức tạp của giải thuật một cách đơn giản hơn.

Ước lượng O (cont.)

Bảng 1: Các hàm ước lượng cơ bản $g(n)$

hàm	**dạng hàm**
constant function	1
logarithm	$\log n$
linear function	n
n-log-n function	$n \log n$
quadratic function	n^2
cubic function	n^3
exponential function	2^n
permuation function	$n!$

Ước lượng O (cont.)

Bài tập

Chứng minh $2n^4 + 3n^3 + 5n^2 + 2n + 3$ là $O(n^4)$

Ước lượng O (cont.)

Chứng minh

Chú ý rằng $2n^4 + 3n^3 + 5n^2 + 2n + 3 \leq$
$2n^4 + 3n^4 + 5n^4 + 2n^4 + 3n^4 = (2 + 3 + 5 + 2 + 3)n^4 = 15n^4$ với
$n \geq 1$ ■

Ước lượng O (cont.)

Định lý 1

Chứng minh $f(n) = a_0 + a_1 n + ... + a_d n^d$ *với* $a_d > 0$ *là* $O(n^d)$

Ước lượng O (cont.)

Chứng minh

Chú ý rằng, với $n \geq 1$ chúng ta có $1 \leq n \leq n^2 \leq ... \leq n^d$. Do đó

$$a_0 + a_1 n + a_2 n^2 ... + a_d n^d \leq (|a_0| + |a_1| + |a_2| + ... + |a_d|) n^d$$

Vậy $f(n) = O(n^d)$ với $c = |a_0| + |a_1| + |a_2| + ... + |a_d|$ và $K = 1$ ■

Ước lượng O (cont.)

Bài tập

Hãy xác định ước lượng O đơn giản nhất cho các hàm sau

- $5n^2 + 3n \log n + 2n + 5$
- $20n^3 + 10n \log n + 5$
- $3 \log n + 2$
- 2^{n+2}
- $2n + 100 \log n$

Ước lượng Ω

Định nghĩa 3

Cho $T(n)$ và $g(n)$ là hai hàm số. Ta nói $T(n) = \Omega(g(n))$ nếu tồn tại các số dương c và K sao cho

$$T(n) \geq cg(n), \forall n \geq K \tag{2}$$

- Cách đọc: $T(n)$ là "big-omega" của $g(n)$
- Ý nghĩa: $g(n)$ là giới hạn dưới của $T(n)$

Ước lượng Θ

Định nghĩa 4

Cho $T(n)$ và $g(n)$ là hai hàm số. Ta nói $T(n) = \Theta(g(n))$ nếu tồn tại các số dương c_1, c_2 và K sao cho

$$c_1 g(n) \leq T(n) \leq c_2 g(n), \forall n \geq K \tag{3}$$

- Cách đọc: $T(n)$ là "big-theta" của $g(n)$
- Ý nghĩa: $g(n)$ là giới hạn chặt của $T(n)$

Quy tắc xác định thời gian thực hiện của các lệnh

- **Câu lệnh rẽ nhánh**: Một cấu trúc rẽ nhánh P có hai lệnh con P_1 và P_2 và thời gian thực hiện của hai lệnh con $T_1(n)$ và $T_2(n)$ thì thời gian thực hiện cho lệnh rẽ nhánh là

$$T(n) = \max(T_1(n), T_2(n)) \tag{4}$$

- **Câu lệnh tuần tự**: Một cấu trúc tuần tự P có hai lệnh con P_1 và P_2 và thời gian thực hiện của hai lệnh con $T_1(n)$ và $T_2(n)$ thì thời gian thực hiện cho lệnh tuần tự là

$$T(n) = T_1(n) + T_2(n) \tag{5}$$

Thực hành phân tích thời gian thực hiện chương trình

Xét các trường hợp sau

- Chương trình không có chương trình con
- Chương trình có gọi chương trình con không đệ qui
- Chương trình đệ qui

Trường hợp 1: Thuật toán tìm kiếm tuần tự

Phân tích thời gian thực hiện

Chương trình 2: Hàm tìm kiếm tuần tự

```
1  int LinearSearch(int n, int a[], int key)
2  {
3      int i = 0;
4      while (i < n)
5      {
6          if (a[i] == key)
7              return i;
8          i++;
9      }
10     return -1;
11 }
```

Trường hợp 1: Thuật toán tìm kiếm tuần tự (cont.)

Xét *trường hợp xấu nhất*

- Lệnh gán
- Lệnh lặp
- Tóm lại độ phức tạp của hàm là $O(n)$

Trường hợp 1: Thuật toán Bubble Sort

Phân tích thời gian thực hiện chương trình

Chương trình 3: Bubble Sort sắp xếp n phần tử

```
1  void BubbleSort(int n, int a[])
2  {
3      int i, j, temp;
4      for (i = 0; i <= n - 2; i++)
5          for (j = n - 1; j >= i + 1; j++)
6              if (a[j].key < a[j - 1].key)
7              {
8                  temp = a[j - 1];
9                  a[j - 1] = a[j];
10                 a[j] = temp;
11             }
12 }
```

Trường hợp 1: Thuật toán Bubble Sort (cont.)

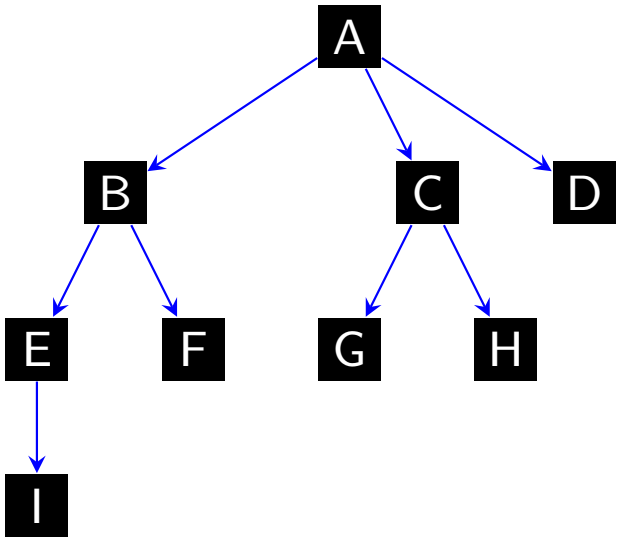
Đây là chương trình sử dụng các vòng lặp xác định. Chương trình gồm lệnh lặp for (dòng 4) lồng lệnh lặp for (dòng 5) có khối lệnh điều kiện if (dòng 6) gồm 3 lệnh con bên trong. Xét *trường hợp xấu nhất*

- Trước hết, cả 3 lệnh gán (dòng 8, 9, 10) có T_1. Do đó, lệnh điều kiện (dòng 6) tốn T_2 thời gian.
- lệnh lặp for (dòng 5) lặp $n - i$ lần, do đó lệnh lặp có T_3
- lệnh lặp for (dòng 4) lặp n lần, do đó lệnh lặp này tốn T_4

$$T(n) = ?$$

Trường hợp 2: Chương trình có gọi chương trình con

Một chương trình trong đó có gọi các chương trình con có thể biểu diễn bằng một cây lời gọi chương trình con



Hình 1: Cây lời gọi chương trình con

Phân tích thời gian thực hiện

Ta có

1. $T(A) = T(B) + T(C) + T(D) + T_A$
2. $T(B) = T(E) + T(F) + T_B$
3. $T(C) = T(G) + T(H) + T_C$
4. $T(E) = T(I) + T_E$

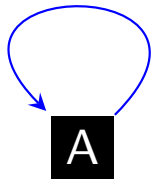
Vậy có thời gian thực hiện là

$$T(A) = T(D)+T(F)++T(G)+T(H)+T(I)+T_A+T_B+T_C+T_E$$

Trường hợp 3: Chương trình đệ qui

Chương trình gọi lại chính nó là một chương trình đệ qui. Để phân tích chương trình đệ qui cần:

1. Thành lập phương trình đệ qui
2. Giải phương trình đệ qui, nghiệm của phương trình đệ qui được xem là thời gian thực hiện của chương trình



Hình 2: Chương trình đệ qui

Thành lập phương trình đệ qui

Phương trình đệ qui biểu diễn mối liên hệ giữa $T(n)$ và $T(k)$ trong đó $T(n)$ và $T(k)$ là thời gian thực hiện tương ứng với "dữ liệu có kích thước" là n và k. Để thành lập phương trình đệ qui ta phải căn cứ vào cấu trúc chương trình đệ qui

- Thời gian thực hiện cho phần dừng
- Thời gian thực hiện cho phần đệ qui
  Dạng tổng quát của phương trình đệ qui

$$T(n) = \begin{cases} T & n = 0 \\ f(T(0)...T(n-1)) & n > 0 \end{cases}$$

- T là thời gian thực hiện ứng với phần dừng
- Hàm f thông thường là một đa thức với $T(k)$

Ví dụ về hàm đệ qui

Ví dụ 1

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

$$T(n) = \begin{cases} 2 & n = 0 \\ 2T(n-1) + 3 & n > 0 \end{cases}$$

$$T(n) = \begin{cases} 2 & n = 1 \\ T(n-1) + T(n/2) + 3 & n > 1 \end{cases}$$

Case Study: Hàm tính giai thừa

Chương trình 4: Hàm tính giai thừa

```
1  int factorial(int n)
2  {
3      if (n == 0)
4          return 1;
5      else
6          return (n * factorial(n - 1));
7  }
```

Case Study: Hàm tính giai thừa (cont.)

Phương trình đệ qui của hàm tính giai thừa

- Khi $n = 0$ chỉ thực hiện lệnh trả về (dòng 2) do đó thời gian thực thi chương trình là $O(1) = C_1$
- Khi $n > 1$ chương trình gọi đệ đi tính cho $n - 1$ và thực hiện phép nhân do đó thời gian thực thi chương trình là $T(n-1) + C_2$
- Vậy phương trình đệ qui cho hàm tính giai thừa là:

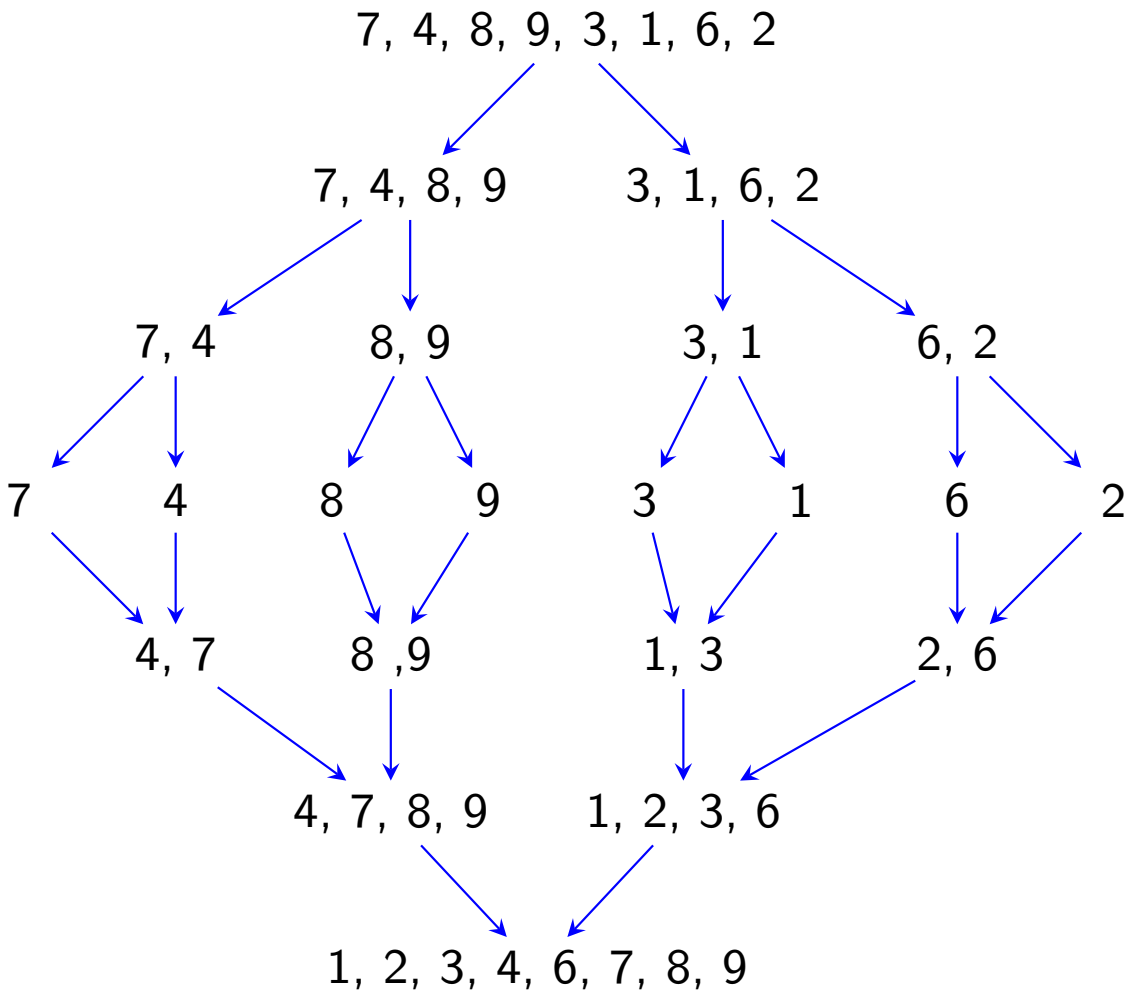
$$T(n) = \begin{cases} C_1 & n = 0 \\ T(n-1) + C_2 & n > 0 \end{cases}$$

Case Study: Merge Sort

Chương trình 5: Merge Sort

```
1  List MergeSort(List L)
2  {
3      List L1, L2;
4      if (L.length <= 1)
5          return L;
6      else
7      {
8          Split(L, L1, L2);
9          return (Merge(MergeSort(L1), MergeSort(L2)));
10     }
11 }
```

Case Study: Merge Sort (cont.)



Case Study: Merge Sort (cont.)

Hình 3: Thuật toán Merge Sort cho 8 phần tử {7, 4, 8, 9, 3, 1, 6, 2}

Phương trình đệ qui của Merge Sort

- $T(n)$ là thời gian thực hiện MergeSort trên một dãy n phần tử
- $T(n/2)$ là thời gian thực hiện MergeSort trên một dãy $n/2$ phần tử
- Khi dãy L có độ dài $n = 1$ thì chương trình chỉ thực hiện lệnh trả về (dòng 4) có thời gian là $T = C_1$
- Khi dãy L có độ dài $n > 1$ thì chương trình gọi đệ quy MergeSort hai lần cho hai dãy L1, L2 có độ dài là $n/2$ do đó thời gian thực thi sẽ là $2T(n/2)$
- Ngoài ra còn phải tốn thời gian cho việc chia danh sách (Split) và trộn hai danh sách (Merge). Có thể dễ dàng thấy thời gian này là $T = C_2 n$
- Vậy ta có phương trình đệ qui như sau:

$$T(n) = \begin{cases} C_1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + C_2 n & n > 1 \end{cases}$$

Giải phương trình đệ qui

Giải phương trình đệ qui là tìm dạng không đệ qui của phương trình. Có ba phương pháp để giải phương trình đệ qui

- Phương pháp truy hồi
- Phương pháp đoán nghiệm
- Phương pháp tổng quát (cho một lớp các phương trình đệ qui)

Phương pháp truy hồi

- Dùng phương pháp thay thế vào vế bên phải của phương trình cho đến khi không còn thay thế được nữa
- Từ đó suy ra phương trình dạng không đệ qui

Giải phương trình đệ qui cho hàm giai thừa

Hàm đệ qui cho hàm giai thừa là

$$T(n) = \begin{cases} C_1 & n = 0 \\ T(n-1) + C_2 & n > 0 \end{cases}$$

Giải phương trình đệ qui cho hàm giai thừa (cont.)

Lời giải

$$
\begin{aligned}
T(n) &= T(n-1) + C_2 \\
T(n) &= (T(n-2) + C_2) + C_2 = T(n-2) + 2C_2 \\
T(n) &= (T(n-3) + C_2) + 2C_2 = T(n-3) + 3C_2 \\
&\dots \\
T(n) &= T(n-i) + iC_2 \\
&\dots \\
T(n) &= C_1 + nC_2
\end{aligned}
$$

Vậy thời gian thực thi của hàm giai thừa là

$$T(n) = C_1 + nC_2 = O(n)$$

■

Giải phương trình đệ qui cho hàm MergeSort

Phương trình đệ qui của MergeSort

$$T(n) = \begin{cases} C_1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + C_2 n & n > 1 \end{cases}$$

Giải phương trình đệ qui cho hàm MergeSort (cont.)

Lời giải

$$
\begin{aligned}
&T(n) = 2T\left(\tfrac{n}{2}\right) + C_2 n \\
&T(n) = 2\left[2T\left(\tfrac{n}{4}\right) + C_2\tfrac{n}{2}\right] + C_2 n = 4T\left(\tfrac{n}{4}\right) + 2C_2 n \\
&T(n) = 4\left[2T\left(\tfrac{n}{8}\right) + C_2\tfrac{n}{4}\right] + C_2 n = 8T\left(\tfrac{n}{8}\right) + 3C_2 n \\
&\ldots \\
&T(n) = 2^i T\left(\tfrac{n}{2^i}\right) + iC_2 n
\end{aligned}
$$

Quá trình thay thế này kết thúc khi $\frac{n}{2^i} = 1$ suy ra $2^i = n, i = \log n$. Khi đó ta có

$$T(n) = nT(1) + \log n C_2 n = C_1 n + C_2 n \log n = O(n \log n)$$

■

Phương pháp đệ qui tổng quát

Ý tưởng

- Để giải bài toán đệ qui kích thước n ta chia bài toán thành a bài toán con, mỗi bài toán con có kích thước n/b. Giải các bài toán con này và tổng hợp kết quả để cho lời giải bài toán ban đầu
- Với các bài toán con tiếp tục áp dụng kỹ thuật chi nhỏ cho đến khi bài toán con có kích thước là 1. Kĩ thuật này dẫn đến một lời giải đệ qui

Phương pháp đệ qui tổng quát (cont.)

Lưu ý

- Giả thiết mỗi bài toán có kích thước là 1 cần 1 đơn vị thời gian
- Giả thiết thời gian để chia bài toán thành bài toán con và tổng hợp các kết quả bài toán thành kết quả cho bài toán lớn là $d(n)$

Phương trình đệ qui cho bài toán tổng quát

- Gọi $T(n)$ là thời gian để giải bài toán kích thước n
- $T(n/b)$ là thơi gian để giải bài toán con có kích thước n/b
- Khi $n = 1$ theo giả thiết thời gian giải bài toán có kích thước 1 là 1 đơn vị, nghĩa là $T(1) = 1$
- Khi $n > 1$ bài toán được chia thành a có kích thước n/b nên thời gian thực thi sẽ là $aT(n/b)$
- Ngoài ra, còn phải tính đến thời gian chia và tổng hợp bài toán là $d(n)$. Vậy ta có phương trình đệ qui

$$T(n) = \begin{cases} 1 & n = 1 \\ aT\left(\frac{n}{b}\right) + d(n) & n > 1 \end{cases}$$

Giải phương trình đệ qui

Ta áp dụng phương pháp thế ta có

$$T(n) = aT\left(\frac{n}{b}\right) + d(n)$$
$$T(n) = a\left(aT\left(\frac{n}{b^2}\right) + d\left(\frac{n}{b}\right)\right) + d(n) = a^2 T\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + dn$$
...

Và ta có

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

Giả sử $n = b^k$, quá trình thay thế trên sẽ kết thúc khi $i = k$. Khi đó ta có

$$T\left(\frac{n}{b^i}\right) = T\left(\frac{n}{b^k}\right) = T\left(\frac{b^k}{b^k}\right) = T(1) = 1$$

Giải phương trình đệ qui (cont.)

Tiếp tục theo thế vào $T(n)$ ta được

$$T(n) = a^k + \sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right) \tag{6}$$

Trong đó, phần $T_1(n) = a^k$ được gọi là nghiệm thuần nhất và phần $T_2(n) = \sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right)$ được gọi là nghiệm riêng Thời gian thực hiện của chương trình là $max(T_1, T_2)$

Hàm nhân

Định nghĩa 5

Một hàm $f(n)$ được gọi là **hàm nhân** (*multiplicative function*) nếu nó thỏa $f(m.n) = f(m).f(n)$ với mọi số nguyên dương m, n

Ví dụ 2

- Hàm $f(n) = n^k$ là một hàm nhân
- Hàm $f(n) = \log n$ không phải là một hàm nhân

Tính nghiệm riêng khi $d(n)$ là hàm nhân

Giả sử $d(n)$ là hàm nhân thì ta có đó $d(b^i) = d(b.b...b) = d(b).d(b)...d(b) = d(b)^i$. Nghiệm riêng được tính như sau:

$$T_2(n) = \sum_{j=0}^{k-1} a^j d\left(b^{k-j}\right)$$

$$T_2(n) = d(b)^k \sum_{j=0}^{k-1} \left(\frac{a}{d(b)}\right)^j$$

$$T_2(n) = d(b)^k \frac{\left(\frac{a}{d(b)}\right)^k - 1}{\frac{a}{d(b)} - 1}$$

Vậy nghiệm riêng là:

$$T_2(n) = \frac{a^k - d(b)^k}{a - d(b)} d(b) \tag{7}$$

Các trường hợp của nghiệm riêng

Có ba trường hợp cụ thể

- Trường hợp 1: $a > d(b)$. Trong công thức 7 thì $a^k > d(b)^k$ nên nghiệm thuần nhất sẽ đóng vai trò chủ đạo. Do đó độ phức tạp

$$T(n) = T_1(n) = O(a^k) = O(n^{\log_b a})$$

- Trường hợp 2: $a < d(b)$. Trong công thức 7 thì $a^k < d(b)^k$ nên nghiệm thuần nhất sẽ đóng vai trò chủ đạo. Do đó độ phức tạp

$$T(n) = T_2(n) = O(d(b)^k) = O(n^{\log_b d(b)})$$

Các trường hợp của nghiệm riêng (cont.)

- Trường hợp 3: $a = d(b)$. Công thức 7 không xác định cho nên ta sẽ tính nghiệm riêng trực tiếp

$$T_2(n) = d(b)^k \sum_{j=0}^{k-1} \left(\frac{a}{d(b)} \right)^j = a^k \sum_{j=0}^{k-1} 1 = a^k k$$

Do $n = b^k$ nên $k = \log_b n$ vậy độ phức tạp là

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

Ví dụ minh họa

Ví dụ 3

Xét hàm đệ qui

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

Ví dụ minh họa (cont.)

Lời giải

1. Phương trình đã cho có dạng phương trình tổng quát với $a = 4, b = 2, d(n) = n$
2. Nhận thấy $d(b) = d(2) = 2 < 4 = a$
3. Do đó

$$T(n) = O\left(n^{\log_b a}\right) = O\left(n^2\right)$$

■

Ví dụ minh họa (cont.)

Ví dụ 4

Xét hàm đệ qui

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

Ví dụ minh họa (cont.)

Lời giải

1. Phương trình đã cho có dạng phương trình tổng quát với $a = 4, b = 2, d(n) = n^2$
2. Nhận thấy $d(b) = d(2) = 4 = a$
3. Do đó

$$T(n) = O\left(n^{\log_b a} \log_b n\right) = O\left(n^2 \log n\right)$$

■

Ví dụ minh họa (cont.)

Ví dụ 5

Xét hàm đệ qui

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

Ví dụ minh họa (cont.)

Lời giải

1. Phương trình đã cho có dạng phương trình tổng quát với $a = 4, b = 2, d(n) = n^3$
2. Nhận thấy $d(b) = d(2) = 8 > 4 = a$
3. Do đó

$$T(n) = O\left(n^{\log_b d(b)}\right) = O\left(n^3\right)$$

■

Tính nghiệm riêng khi $d(n)$ không phải là hàm nhân

Trong trường hợp $d(n)$ không phải là hàm nhân thì không thể áp dụng ba trường hợp nói trên để tính nghiệm riêng. Do đó sẽ tính trực tiếp nghiệm thuần nhất $T_1(n)$ và nghiệm riêng $T_2(n)$, sau đó tính $max(T_1(n), T_2(n))$

Ví dụ minh họa

Ví dụ 6

Xét hàm đệ qui

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

Lời giải

- Nghiệm thuần nhất có

$$T_1(n) = n^{\log_b a} = n^{\log_2 2} = n$$

Ví dụ minh họa (cont.)

- Do $d(n) = n \log n$ không phải là hàm nhân nên ta tính nghiệm riêng trực tiếp

$$T_2(n) = \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j 2^{k-j} \log 2^{k-j}$$
$$T_2(n) = 2^k \sum_{j=0}^{k-1} (k-j) = 2^k \frac{k(k+1)}{2}$$
$$T_2(n) = O\left(2^k k^2\right)$$

- Vì $n = b^k$ và $k = \log_b n$, trong trường hợp này $n = 2^k$ và $k = \log n$
- Vậy nghiệm riêng

$$T_2(n) = O\left(n \log^2 n\right)$$

Ví dụ minh họa (cont.)

- Tóm lại, nghiệm riêng có vài trò thống trị nên dạng không đệ qui của hàm được xấp xỉ

$$T(n) = O\left(n \log^2 n\right)$$

■

Bài tập luyện tập

Tìm thời gian thực hiện cho đoạn chương trình sau theo tham số n

```
1  for(i = 0; i < n; i++)
2      for (j = 0; j < n; j++)
3          b[i][j] += c;
4
5  for(i = 0; i < n; i++)
6      for (j = i+1; j < n; j++)
7          b[i][j] -= c;
```

Bài tập luyện tập (cont.)

Tìm thời gian thực hiện cho đoạn chương trình cộng ma trận theo tham số n

```
1  for(i = 0; i < n; i++)
2      for (j = 0; j < n; j++)
3          a[i][j] = b[i][j]+c[i][j];
```

Bài tập luyện tập (cont.)

Tìm thời gian thực hiện cho đoạn chương trình nhân ma trận theo tham số n

```
1  for(i = 0; i<n; i++)
2      for (j = 0; j<n; j++)
3          for(k=a[i][j]=0; k<n; k++)
4              a[i][j]+=b[i][k]+c[k][j];
```

Bài tập luyện tập (cont.)

Tìm dạng không đệ qui của các hàm đệ qui sau

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

Bài tập luyện tập (cont.)

Phân tích thời gian thực thi theo tham số n (số đĩa)

```
1  void HanoiTower(int n, int a, int b, int c)
2  {
3      if (n > 0)
4      {
5          HanoiTower(n - 1, a, c, b);
6          cout << "move from " << a << " to " << c <<
               endl;
7          HanoiTower(n - 1, b, a, c);
8      }
9  }
10 void Swap(int &a, int &b)
11 {
12     int t = a;
13     a = b;
14     b = t;
15 }
```

Bài tập luyện tập (cont.)

Phân tích thời gian thực thi theo tham số n (số phần tử của mảng a)

```
1  void Permute(int k, int n, int a[])
2  {
3      if (k == 0)
4      {
5          for (int i = 0; i < n; i++)
6              cout << a[i] << " ";
7          cout << endl;
8      }
9      else
10     {
11         for (int i = 0; i < k; i++)
12         {
13             Swap(a[i], a[k - 1]);
14             Permute(k - 1, n, a);
15             Swap(a[i], a[k - 1]);
16         }
```

Bài tập luyện tập (cont.)

```
17          }
18  }
```

Tài liệu tham khảo


Apostol, T. M. (1976).
Introduction to analytic number theory.
Springer.


Knuth, D. E. (1976).
Big omicron and big omega and big theta.
ACM Sigact News, 8(2):18–24.


Landau, E., Goodman, J. E., Bateman, P. T., and Kohlbecker, E. E. (1958).
Elementary number theory.
Chelsea Publishing Company New York.