

PHÂN TÍCH THIẾT KẾ GIẢI THUẬT

Giảng viên phụ trách:
NGUYỄN THÁI DƯ

Sự cần thiết phải phân tích giải thuật

- ◆ Với phần lớn các bài toán, thường có nhiều giải thuật khác nhau để giải một bài toán.
- ◆ Làm cách nào để chọn giải thuật tốt nhất để giải một bài toán?
- ◆ Làm cách nào để so sánh các giải thuật cùng giải được một bài toán?
=> Cần phải **đánh giá các giải thuật** để lựa chọn giải thuật tốt nhất.

Chương 1 KỸ THUẬT PHÂN TÍCH GIẢI THUẬT

- ◆ Sự cần thiết phải phân tích giải thuật
- ◆ Thời gian thực hiện của giải thuật
- ◆ Khái niệm độ phức tạp của giải thuật
- ◆ Cách tính độ phức tạp của:
 - Chương trình không gọi chương trình con
 - Chương trình có gọi chương trình con không đệ quy
 - Chương trình đệ quy

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:
 - > Thường chỉ sử dụng vài lần
 - Tính nhanh chóng (thời gian thực thi)
 - Quan trọng khi chương trình được thực thi nhiều lần, chương trình có khối lượng dữ liệu nhập lớn.
 - Hiệu quả thời gian thực hiện của giải thuật

Sự cần thiết phải phân tích giải thuật

◆ 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

- Thời gian thực hiện một chương trình là một hàm của kích thước dữ liệu vào, ký hiệu $T(n)$ trong đó n là kích thước (độ lớn) của dữ liệu vào.
- **Ví dụ :** Chương trình tính tổng của n số có thời gian thực hiện là $T(n) = cn$ trong đó c là một hằng số.
- Thời gian thực hiện chương trình là một hàm không âm, tức là $T(n) \geq 0 \forall n \geq 0$.

Đơn vị đo thời gian thực hiện

- Đơn vị của $T(n)$ không phải là đơn vị đo thời gian bình thường như giờ, phút giây...
- Thường được xác định bởi số các lệnh được thực hiện trong một máy tính lý tưởng.
- **Ví dụ:** Khi ta nói thời gian thực hiện của một chương trình là $T(n) = Cn$ thì có nghĩa là chương trình ấy cần Cn chỉ thị thực thi.

Thời gian thực hiện trong trường hợp xấu nhất

- ◆ Thời gian thực hiện chương trình không chỉ phụ thuộc vào kích thước mà còn phụ thuộc vào tính chất của dữ liệu vào.
- ◆ Dữ liệu vào có cùng kích thước nhưng thời gian thực hiện chương trình có thể khác nhau.
- ◆ *Ví dụ:* chương trình sắp xếp dãy số nguyên tăng dần. Nhập vào dãy số nguyên.
 => Dãy số nhập vào có thể: có thứ tự, chưa có thứ tự, có thứ tự tăng hoặc có thứ tự giảm.
 => Thời gian thực hiện trong các trường hợp: tốt nhất, xấu nhất, trung bình

Thời gian thực hiện trong trường hợp xấu nhất

- ◆ $T(n)$ là thời gian thực hiện chương trình **trong trường hợp xấu nhất** trên dữ liệu vào có kích thước n
- ◆ $T(n)$ là thời gian lớn nhất để thực hiện chương trình đối với mọi dữ liệu vào có cùng kích thước n .

Tỷ suất tăng

- ◆ $T(n)$ có tỷ suất tăng $f(n)$ nếu tồn tại hằng số C và N_0 sao cho $T(n) \leq Cf(n) \quad \forall n \geq N_0$
- ◆ Cho một hàm không âm $T(n)$, luôn tồn tại tỷ suất tăng $f(n)$ của nó
- ◆ *Ví dụ 1:* Giả sử $T(0) = 1$, $T(1) = 4$, tổng quát $T(n) = (n+1)^2$,
 - Đặt $N_0 = 1$ và $C = 4$ thì với mọi $n \geq 1$, ta có:
 - $T(n) = (n+1)^2 \leq 4n^2 \quad \forall n \geq 1,$
 \Rightarrow Tỷ suất tăng của $T(n)$ là n^2 .

Tỷ suất tăng

- ◆ *Ví dụ 2:* chứng minh rằng:

Tỷ suất tăng của $T(n) = 3n^3 + 2n^2$ là n^3

Giải

- Chọn $N_0 = 0$ và $C = 5$ với mọi $n \geq 0$, ta có:
 $3n^3 + 2n^2 \leq 5n^3 \quad \forall n \geq 0$
 \Rightarrow Tỷ suất tăng của $T(n)$ là n^3 .

Quy tắc:

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

Khái niệm độ phức tạp của giải thuật

- ◆ Cùng một bài toán, có 2 thuật toán P1 và P2 với thời gian thực hiện là:
 - P1 có $T1(n) = 100n^2$
 - P2 có $T2(n) = 5n^3$
- ◆ Giải thuật nào chạy nhanh hơn ?
 - Xét nếu $n \leq 20$ thì $T1(n) > T2(n)$
 - Xét nếu $n > 20$ thì $T1(n) < T2(n)$
- Chọn P1 vì hầu hết các trường hợp $T1(n) < T2(n)$
- ◆ Dùng thời gian để lựa chọn rất khó khăn vì phải so sánh 2 đa thức

Khái niệm độ phức tạp của giải thuật

- P1 có $T_1(n) = 100n^2$ → tỷ suất tăng là n^2
- P2 có $T_2(n) = 5n^3$ → tỷ suất tăng là n^3
- ◆ Chỉ xét n^2 và n^3 thì rất dễ dàng lựa chọn vì $n^2 < n^3$

=> So sánh theo tỷ suất tăng hơn là so sánh trực tiếp các hàm $T(n)$

Khái niệm độ phức tạp của giải thuật

- ◆ Một số hàm thể hiện độ phức tạp thường gặp: $\log_2 n$, n , $n \log_2 n$, n^2 , n^3 , 2^n , $n!$, n^n .
- ◆ Các hàm: $\log_2 n$, n , $n \log_2 n$, n^2 , n^3 gọi là hàm đa thức
- ◆ Ba hàm cuối cùng: 2^n , $n!$, n^n gọi là dạng hàm mũ,
- ◆ Nhận xét:
 - Một giải thuật mà thời gian thực hiện có độ phức tạp là một hàm đa thức thì chấp nhận được tức là có thể cài đặt để thực hiện
 - Các giải thuật có độ phức tạp hàm mũ thì phải tìm cách cải tiến giải thuật

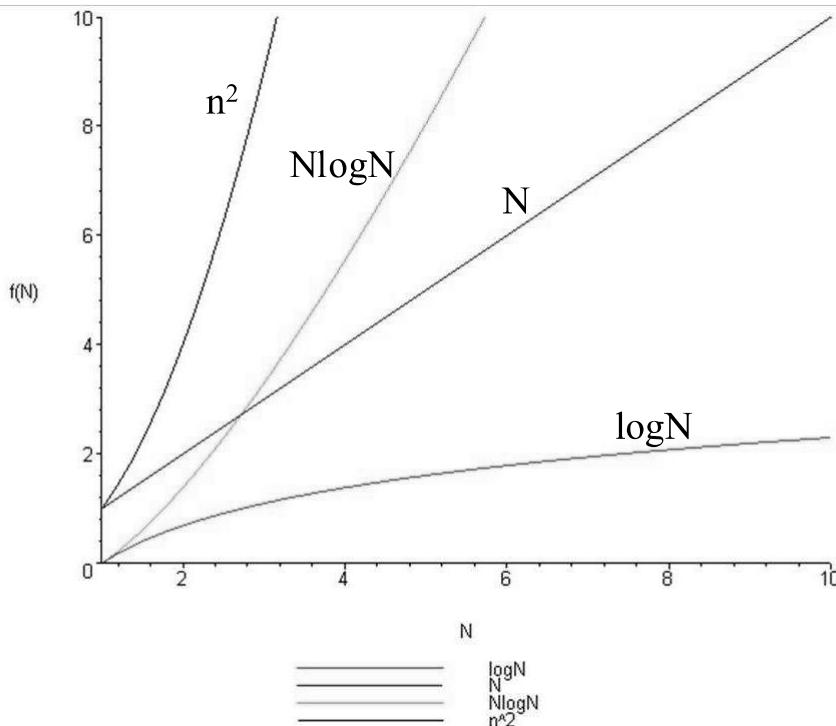
Khái niệm độ phức tạp của giải thuật

◆ Cho một hàm $T(n)$, $T(n)$ gọi là có độ phức tạp $f(n)$ nếu tồn tại các hằng C , N_0 sao cho $T(n) \leq Cf(n)$ với mọi $n \geq N_0$ (tức là $T(n)$ có tỷ suất tăng là $f(n)$) và ký hiệu $T(n)$ là $O(f(n))$ (đọc là “ô của $f(n)$ ”).

◆ Ví dụ: $T(n) = (n + 1)^2$ có tỷ suất tăng là n^2 nên hàm $T(n)$ có độ phức tạp $O(n^2)$

Tính chất

- $O(C.f(n)) = O(f(n))$ với C là hằng số
- $O(C) = O(1)$



Cách tính độ phức tạp

- ◆ Chương trình không gọi chương trình con
- ◆ Chương trình có gọi chương trình con không đệ quy
- ◆ Chương trình đệ quy

Quy tắc tổng quát để phân tích một chương trình không có chương trình con

- ◆ Các lệnh gán, nhập, xuất, return: O(1)
- ◆ Một chuỗi tuần tự các lệnh: Quy tắc cộng
 - if (điều kiện) CV1
 - else CV2
- ◆ Câu trúc IF: Max (*điều kiện, CV1, CV2*)
 - ☞ Thường thời gian kiểm tra điều kiện là O(1)
- ◆ Câu trúc 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.

Cách tính độ phức tạp

- ◆ Trước hết ta có hai quy tắc quan trọng là quy tắc cộng và quy tắc nhân
- ◆ Cho 2 đoạn chương trình:
 - P1 có thời gian thực hiện $T_1(n) = O(f_1(n))$
 - P2 có thời gian thực hiện $T_2(n) = O(f_2(n))$
- ◆ Quy tắc cộng:
 - Thời gian thực thi P1 và P2 nối tiếp nhau sẽ là:
 - $T(n) = T_1(n) + T_2(n) = O(\max(f_1(n), f_2(n)))$
- ◆ Quy tắc nhân:
 - Thời gian thực thi P1 và P2 lồng nhau:
 - $T(n) = T_1(n) \times T_2(n) = O(f_1(n) \times f_2(n))$

Vòng lặp xác định số lần lặp

- ◆ For (*i=a; i<=b; i++*) : Số lần lặp = $b-a+1$
- ◆ For (*i=a; i<b; i++*) : Số lần lặp = $b-a$
- ◆ For (*i=1; i<=n; i=2*i*)
 - Sau lần lặp thứ 1: $i = 2$
 - Sau lần lặp thứ 2: $i = 4$
 -
 - Sau lần lặp thứ k: $i = 2^k$
 - Lặp kết thúc khi $i = 2^k = n$
 - $k = \log_2 n \Rightarrow$ số lần lặp = $\log_2 n$

Vòng lặp không xác định số lần lặp

- ◆ While (điều kiện)
 - ◆ Do... while...
 - ◆ Xác định số lần lặp trong trường hợp xấu nhất

Nguyễn Thái Dư - AGU

21

Cách tính độ phức tạp

- ◆ Ví dụ: Tính thời gian thực hiện của các đoạn chương trình sau:

- Thuật toán tính tổng các số từ 1 đến n

{1} s=0; Lệnh {1} và {2} nối tiếp nhau. Lệnh {1} tốn O(1)

{2} for (i= 1; i<=n; i++)

{3} s=s+i;

Lệnh {3} tồn O(1)

Quy tắc tổng quát để phân tích một chương trình không có chương trình con

- ◆ Trình tự đánh giá:
 - Nối tiếp: Từ trên xuống
 - Lồng nhau: Từ trong ra

Nguyễn Thái Dư - AGU

22

Cách tính độ phức tạp

```
{1} s=0;  
{2} for (i= 1;i<=n;i++)  
{3}     s=s+i;
```

- Lệnh {1} tồn O(1) thời gian
 - Lệnh {3} tồn O(1) thời gian
 - Vòng lặp {2} lặp n lần, mỗi lần tồn O(1) do đó vòng lặp {2} tồn $O(n \cdot 1) = O(n)$.
 - Lệnh {1}, {2} nối tiếp nhau $O(\max(1, n)) = O(n)$

=> **độ phức tạp là O(n)**

Nguyễn Thái Dư - AGU

23

Nguyễn Thái Dư - AGU

24

Cách tính độ phức tạp

❖ **Ví dụ:** Tính thời gian thực hiện của đoạn chương trình sắp xếp “nối bọt”

```
void BubbleSort(int a[], int n)
{
    int i, j, temp;
    for(i=0; i<(n-1); i++) //1
        for(j=n-1; j>i+1; j--) //2
            if(a[j-1] > a[j]) { //3
                temp = a[j-1]; //4
                a[j-1] = a[j]; //5
                a[j] = temp; //6
            }
}
```

- Cả ba lệnh đổi chỗ {4} {5} {6} tốn O(1) thời gian, do đó lệnh {3} tốn O(1).
- Vòng lặp {2} thực hiện (n-i) lần, mỗi lần O(1). Do đó vòng lặp {2} tốn O((n-i).1)=O(n-i).
- Vòng lặp {1} lặp (n-1) lần vậy độ phức tạp của giải thuật là:
$$T(n) = \sum_{i=1}^{n-1} (n-i) = (n-1) + (n-2) + \dots + (n-k-1) + \dots + 1 = \frac{n(n-1)}{2}$$

$$T(n) = \sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2} = O(n^2)$$

Nguyễn Thái Dư - AGU

25

Cách tính độ phức tạp

❖ **Ví dụ:** Tìm kiếm tuần tự. Hàm tìm kiếm Search nhận vào một mảng a có n số nguyên và một số nguyên x, hàm sẽ trả về giá trị logic TRUE nếu tồn tại một phần tử a[i] = x, ngược lại hàm trả về FALSE.

- Giải thuật tìm kiếm tuần tự là lần lượt so sánh x với các phần tử của mảng a,
 - Bắt đầu từ a[1], nếu tồn tại a[i] = x thì dừng và trả về TRUE,
 - Ngược lại nếu tất cả các phần tử của a đều khác X thì trả về FALSE.

Nguyễn Thái Dư - AGU

26

Cách tính độ phức tạp

❖ **Ví dụ:** Hàm tìm kiếm tuần tự.

```
int LinearSearch (int a[], int n, int x)
{
    int i=0; //1
    found =0; //2
    while ((i<n) && ! found) //3
        if (a[i]==x) found =1; //4
        else i++;
    return found; //5
}
```

Các lệnh {1}, {2}, {3} và {5} nối tiếp nhau,

Ba lệnh {1}, {2} và {5} đều có độ phức tạp O(1)
Do đó độ phức tạp của hàm Search chính là độ phức tạp lớn nhất trong 4 lệnh này.

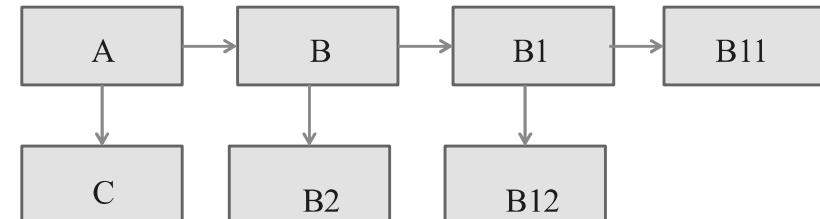
Lệnh {4} có độ phức tạp O(1)
Lệnh {3} thực hiện n lần
Vậy ta có $T(n) = O(n)$.

Nguyễn Thái Dư - AGU

27

Cách tính độ phức tạp

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



- Để tính thời gian thực hiện của A, ta tính theo các bước sau:
 - Tính thời gian thực hiện của C, B2, B11 và B12.
 - Tính thời gian thực hiện của B1.
 - Tính thời gian thực hiện của B.
 - Tính thời gian thực hiện của A.

Nguyễn Thái Dư - AGU

28

Cách tính độ phức tạp

◆ **Ví dụ:** Ta có thể viết lại chương trình sắp xếp bubble như sau:

- Trước hết chúng ta viết thủ tục HoanDoi để thực hiện việc hoàn đổi hai phần tử cho nhau,
- Sau đó trong thủ tục Bubble, khi cần ta sẽ gọi đến thủ tục HoanDoi này.

```
void HoanDoi(int &x, int &y)
{
    int tam = x; x = y; y = tam; }
```

Cách tính độ phức tạp

```
void BubbleSort(int a[], int n)
{
    int i, j, temp;
    for(i=0; i<(n-1); i++) //1
        for(j=n-1; j>i+1; j--) //2
            if(a[j-1] > a[j]) HoanDoi(a[j-1], a[j]); //3
}
```

Cách tính độ phức tạp

```
void HoanDoi(int &x, int &y)
{
    int tam = x; x = y; y = tam; }

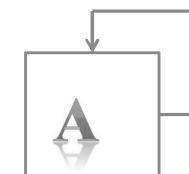
void BubbleSort(int a[], int n)
{
    int i, j, temp;
    for(i=0; i<(n-1); i++) //1
        for(j=n-1; j>i+1; j--) //2
            if(a[j-1] > a[j])
                HoanDoi(a[j-1], a[j]); //3
}
```

- Chương trình Bubble gọi chương trình con Swap
- Thời gian thực hiện của Swap là O(1) vì nó chỉ bao gồm 3 lệnh gán
- Trong Bubble, lệnh {3} gọi Swap nên chỉ tốn O(1)
- Lệnh {2} thực hiện n-i lần, mỗi lần tốn O(1), nên tốn O(n-i).
- Lệnh {1} thực hiện n-1 lần nên

$$T(n) = \sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2} = O(n^2)$$

Phân tích các chương trình đệ quy

- ◆ Với các chương trình có gọi các chương trình con đệ quy, ta không thể áp dụng cách tính như trình bày phần trước vì một chương trình đệ quy sẽ gọi chính bản thân nó.
- ◆ Chương trình đệ quy có thể minh họa như hình ảnh sau:



Phân tích các chương trình đệ quy

- 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:

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

Thành lập phương trình đệ quy

- ◆ Thông thường đối với một chương trình đệ quy để giải bài toán kích thước n ,
 - phải có ít nhất một trường hợp dừng ứng với một n cụ thể
 - và lời gọi đệ quy để giải bài toán kích thước k ($k < n$).

Thành lập phương trình đệ quy

- ◆ Phương trình đệ quy là một phương trình biểu diễn mối liên hệ giữa $T(n)$ và $T(k)$, trong đó
 - $T(n)$ là thời gian thực hiện chương trình với kích thước dữ liệu nhập là n ,
 - $T(k)$ thời gian thực hiện chương trình với kích thước dữ liệu nhập là k , với $k < n$.
 - Để thành lập được phương trình đệ quy, ta phải căn cứ vào chương trình đệ quy.

Thành lập phương trình đệ quy

- ◆ Để thành lập phương trình đệ quy, ta gọi
 - $T(n)$ là thời gian để giải bài toán kích thước n
 - $T(k)$ là thời gian để giải bài toán kích thước k
- ◆ Khi đệ quy dừng, ta phải xem xét khi đó chương trình làm gì và tốn hết bao nhiêu thời gian, chẳng hạn thời gian này là $c(n)$
- ◆ Khi đệ quy chưa dừng thì phải xem có bao nhiêu lời gọi đệ quy với kích thước k ta sẽ có bấy nhiêu $T(k)$
 - Ngoài ra ta còn phải xem xét đến thời gian để phân chia bài toán và tổng hợp các lời giải, chẳng hạn thời gian này là $d(n)$.

Thành lập phương trình đệ quy

- ◆ Dạng tổng quát của một phương trình đệ quy sẽ là:

$$T(n) = \begin{cases} C(n) \\ F(T(k)) + d(n) \end{cases}$$

- ◆ Trong đó:

- $C(n)$ là thời gian thực hiện chương trình ứng với trường hợp đệ quy dừng
- $F(T(k))$ là một đa thức của các $T(k)$.
- $d(n)$ là thời gian để phân chia bài toán và tống hợp các kết quả.

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

- ❖ Gọi $T(n)$ là thời gian thực hiện việc tính n giai thừa
- ❖ $T(n-1)$ là thời gian thực hiện việc tính $n-1$ giai thừa
- ❖ Trường hợp $n=0$ thì chương trình chỉ thực hiện một lệnh gán $Giai_thua:=1$ nên tốn $O(1)$, do đó ta có $T(0) = C_1$
- ❖ Trường hợp $n>0$ chương trình phải gọi đệ quy $Giai_thua(n-1)$, việc gọi đệ quy này tốn $T(n-1)$
 - Sau khi có kết quả của việc gọi đệ quy, chương trình phải nhân kết quả đó với n và gán cho $Giai_thua$
 - Thời gian để thực hiện phép nhân và phép gán là một hằng C_2 .
- ❖ Vậy ta có phương trình đệ quy để tính thời gian thực hiện của chương trình đệ quy $Giai_thua$ là:
$$T(n) = \begin{cases} C_1 & \text{nếu } n = 0 \\ T(n-1) + C_2 & \text{nếu } n > 0 \end{cases}$$

Thành lập phương trình đệ quy

- ◆ **Ví dụ:** Xét hàm tính giai thừa viết bằng giải thuật đệ quy như sau:

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

Giải phương trình đệ quy

- ◆ Có ba phương pháp giải phương trình đệ quy:
 1. Phương pháp truy hồi
 2. Phương pháp đoán nghiệm.
 3. Lời giải tổng quát của một lớp các phương trình đệ quy.

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 $T(n)$
- Lời giải tổng quát cho một lớp các phương trình đệ quy

Phương pháp truy hồi

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

Phương pháp truy hồi

- ◆ Ví dụ: Giải phương trình

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

- ◆ Ta có $T(n) = T(n-1) + C_2$

$$T(n) = [T(n-2) + C_2] + C_2 = T(n-2) + 2C_2$$

$$T(n) = [T(n-3) + C_2] + 2C_2 = T(n-3) + 3C_2$$

.....

$$T(n) = T(n-i) + iC_2$$

- ◆ Quá trình trên kết thúc khi $n - i = 0$ hay $i = n$. Khi đó ta có

$$T(n) = T(0) + nC_2 = C_1 + nC_2 = O(n)$$

Phương pháp truy hồi

- ◆ Ví dụ: Giải phương trình

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

- ◆ Ta có

$$T(n) = 2T\left(\frac{n}{2}\right) + C_2 n$$

$$T(n) = 2[2T\left(\frac{n}{4}\right) + C_2 \left(\frac{n}{2}\right)] + C_2 n = 4T\left(\frac{n}{4}\right) + 2C_2 n$$

$$T(n) = 4[2T\left(\frac{n}{8}\right) + C_2 \left(\frac{n}{4}\right)] + 2C_2 n = 8T\left(\frac{n}{8}\right) + 3C_2 n$$

.....

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + iC_2 n$$

- ◆ Quá trình kết thúc khi $\frac{n}{2^i} = 1$ hay $2^i = n$ và do đó $i = \log_2 n$. Khi đó ta có:

$$T(n) = nT(1) + \log_2 n C_2 n = C_1 n + C_2 n \log_2 n = O(n \log n).$$

- Thủ 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 , $\log n$, $n \log n$, 2^n , ...

Bài toán đệ quy tổng quát

- ◆ Để giải một bài toán kích thước n , ta chia bài toán đã cho 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 lại để được kết quả của bài toán đã cho.
- ◆ Với các bài toán con chúng ta cũng sẽ áp dụng phương pháp đó để tiếp tục chia nhỏ ra nữa cho đến các bài toán con kích thước 1. Kỹ thuật này sẽ dẫn chúng ta đến một giải thuật đệ quy.
- ◆ Giả thiết rằng mỗi bài toán con kích thước 1 lấy một đơn vị thời gian
- ◆ Giả thiết thời gian để chia bài toán kích thước n thành các bài toán con kích thước n/b và tổng hợp kết quả từ các bài toán con để được lời giải của bài toán ban đầu là $d(n)$.

- ◆ Trong mục này, chúng ta sẽ nghiên cứu các phần sau:
 - Bài toán đệ quy tổng quát.
 - Thành lập phương trình đệ quy tổng quát.
 - Giải phương trình đệ quy tổng quát.
 - Các khái niệm về nghiệm thuần nhất, nghiệm riêng và hàm nhán.
 - Nghiệm của phương trình đệ quy tổng quát khi $d(n)$ là hàm nhán.
 - Nghiệm của phương trình đệ quy tổng quát khi $d(n)$ không phải là hàm nhán.

Thành lập phương trình đệ quy tổng quát

- ◆ Nếu gọi $T(n)$ là thời gian để giải bài toán kích thước n
- ◆ Thì $T(n/b)$ là thời gian để giải bài toán con kích thước n/b .
- ◆ Khi $n = 1$ theo giả thiết trên thì thời gian giải bài toán kích thước 1 là 1 đơn vị, tức là $T(1) = 1$.
- ◆ Khi n lớn hơn 1, ta phải giải đệ quy a bài toán con kích thước n/b , mỗi bài toán con tốn $T(n/b)$ nên thời gian cho a lời giải đệ quy này là $aT(n/b)$.
- ◆ Ngoài ra ta còn phải tốn thời gian để phân chia bài toán và tổng hợp các kết quả, thời gian này theo giả thiết trên là $d(n)$. Vậy ta có phương trình đệ quy:

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

Giải phương trình đệ quy tổng quát

$$T(n) = aT\left(\frac{n}{b}\right) + d(n)$$

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

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

$$\begin{aligned} T(n) &= a^2 \left[aT\left(\frac{n}{b^3}\right) + d\left(\frac{n}{b^2}\right) \right] + ad\left(\frac{n}{b}\right) + d(n) \\ &= a^3 T\left(\frac{n}{b^3}\right) + a^2 d\left(\frac{n}{b^2}\right) + ad\left(\frac{n}{b}\right) + d(n) \end{aligned}$$

.....

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

Nguyễn Thái Dư - AGU

49

Nghiệm thuần nhất và nghiệm riêng

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

Nghiệm thuần nhất
 $a^k = n^{\log_b a}$

Nghiệm riêng

Nghiệm của phương trình là: MAX(NTN,NR).

Nguyễn Thái Dư - AGU

51

Giải phương trình đệ quy tổng quát (tt)

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

Giả sử $n = b^k$, quá trình suy rộng trên sẽ kết thúc khi $i = k$. Khi đó ta được:

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

Thay vào trên ta có:

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

Nguyễn Thái Dư - AGU

50

Hàm nhân

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

Ví dụ:

- Hàm $f(n) = n^k$ là một hàm nhân, vì $f(m.n) = (m.n)^k = m^k.n^k = f(m).f(n)$.
- Hàm $f(n) = \log n$ không phải là một hàm nhân, vì $f(n.m) = \log(n.m) = \log n + \log m \neq \log n \cdot \log m = f(n).f(m)$

Nguyễn Thái Dư - AGU

52

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

◆ Khi $d(n)$ là hàm nhân, ta có:

$$◆ d(b^{k-j}) = d(b \cdot b \cdot b \dots b) = d(b) \cdot d(b) \dots d(b) = [d(b)]^{k-j}$$

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

$$\text{Hay } NR = \frac{a^k - [d(b)]^k}{\frac{a}{d(b)} - 1}$$

Ba trường hợp (tt)

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

Trường hợp 3: $a = d(b)$

Công thức trên không xác định nên ta phải tính trực tiếp nghiệm riêng:

$$NR = [d(b)]^k \sum_{j=0}^{k-1} \left[\frac{a}{d(b)} \right]^j = a^k \sum_{j=0}^{k-1} 1 = a^k k \quad (\text{do } a = d(b))$$

Do $n = b^k$ nên $k = \log_b n$ và $a^k = n^{\log_b a}$.

Vậy NR là $n^{\log_b a} \log_b n > NTN$.

Do đó $T(n)$ là $O(n^{\log_b a} \log_b n)$.

Ba trường hợp

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

◆ **Trường hợp 1:** $a > d(b)$

Trong công thức trên ta có $a^k > [d(b)]^k$, theo quy tắc lấy độ phức tạp ta có NR là $O(a^k) = O(n^{\log_b a}) = NTN$.

Do đó $T(n)$ là $O(n^{\log_b a})$.

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

◆ **Trường hợp 2:** $a < d(b)$

Trong công thức trên ta có $[d(b)]^k > a^k$, theo quy tắc lấy độ phức tạp ta có NR là $O([d(b)]^k) = O(n^{\log_b d(b)}) > NTN$.

Do đó $T(n)$ là $O(n^{\log_b d(b)})$.

Bài tập

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

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

$$4/ \quad T(1) = 1$$

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

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

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

Nghiệm của phương trình đệ quy tổng quát khi $d(n)$ không phải là hàm nhân

- Trong trường hợp hàm tiến triển không phải là một hàm nhân thì chúng ta không thể áp dụng các công thức ứng với ba trường hợp nói trên mà chúng ta phải tính trực tiếp NR, sau đó lấy MAX(NR, NTN).

Ví dụ: GPT với $T(1) = 1$ và

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

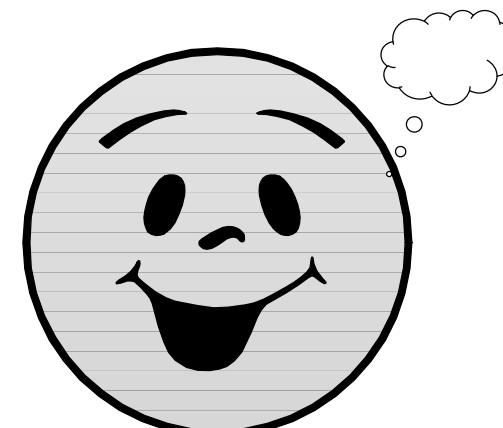
- PT thuộc dạng phương trình tổng quát nhưng $d(n) = n\log n$ không phải là một hàm nhân.
- $NTN = n^{\log_b a} = n^{\log 2} = n$
- Do $d(n) = n\log n$ không phải là hàm nhân nên ta phải tính nghiệm riêng bằng cách xét trực tiếp

Ví dụ (tt)

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

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

- 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 $k = \log n$,
- $NR = O(n\log^2 n) > NTN$
- $T(n) = O(n\log^2 n)$.



Cảm ơn !