

Chương I. Kỹ thuật phân tích giải thuật

Phạm Nguyên Khang
BM. Khoa học máy tính
Khoa CNTT – Đại học Cần Thơ
pnkhang@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 $T(n) \geq 0$, 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 $T_1(n)$ và $T_2(n)$

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 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: $T(0) = 1, T(1) = 4, T(n) = (n+1)^2$, ta có:
 $f(n) = n^2$ (với $C = 4, n_0 = 1$)

??? Chứng minh rằng:

- Tỷ suất tăng của $T(n) = 3n^3 + 2n$ là n^3
 - Chọn C , chọn n_0 , chứng minh bằng quy nạp

Độ phức tạp giải thuật

- Cho 2 giải thuật

- P_1 có $T_1(n) = 100n^2$
- P_2 có $T_2(n) = 5n^3$

??? Giải thuật nào chạy nhanh hơn

→ 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 $f(n) \rightarrow T(n)$ 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:
 $\log_2 n, n \log_2 n, n^2, n^3, 2^n, n!, n^n, \dots$

Cách tính độ phức tạp

- Cho 2 chương trình:
 - P_1 có thời gian thực hiện $T_1(n) = O(f_1(n))$
 - P_2 có thời gian thực hiện $T_2(n) = O(f_2(n))$
- Quy tắc cộng:
 - Thời gian thực thi P_1 và P_2 nối tiếp nhau sẽ là:
 - $T(n) = T_1(n) + T_2(n) = O(\max(f_1(n), f_2(n)))$
 - Thời gian thực thi P_1 và P_2 lồng nhau (vd: vòng lặp lồng nhau chẳng hạn):
 - $T(n) = T_1(n) \times T_2(n) = O(f_1(n) \times f_2(n))$

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
 - $\max(\text{lệnh 1}, \text{lệnh 2}) + \text{điều kiện}$
 - 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 $\rightarrow O(f(n))$ 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:

```
1. void sap_xep(int a[], int n) {  
2.     int tam;  
3.     for (int i = 0; i < n; i++)  
4.         for (int j = i + 1; j < n; j++)  
5.             if (a[j] < a[i]) {  
6.                 tam = a[i];  
7.                 a[i] = a[j];  
8.                 a[j] = tam;  
9.             }  
10. }
```

Cách tính độ phức tạp

- Ví dụ:

```
1. int tim_kiem(int x, int a[], int n) {  
2.     int found, i;  
3.     found = 0;  
4.     i = 0;  
5.     while (i < n && !found)  
6.         if (a[i] == x)  
7.             found = 1;  
8.         else  
9.             i = i+1;  
10.    return i;  
11. }
```

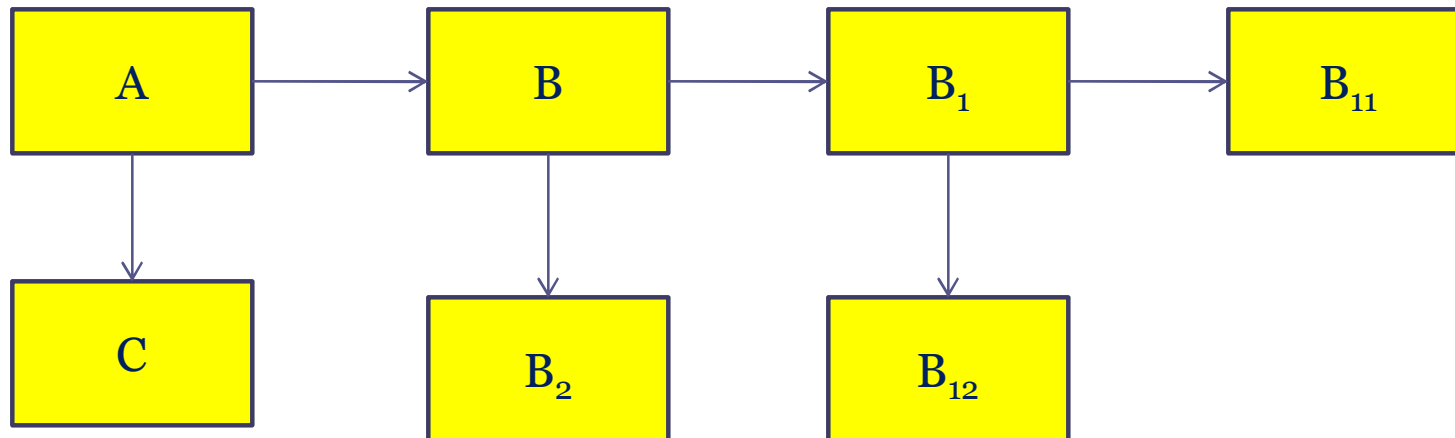
Cách tính độ phức tạp

- Ví dụ:

```
1. int tim_kiem_nhi_phan(int x, int a[], int n) {  
2.     int i = 0, j = n - 1;  
3.     while (i <= j) {  
4.         int m = (i + j)/2;  
5.         if (x == a[m])  
6.             return m;  
7.         if (x < a[m])  
8.             j = m - 1;  
9.         else  
10.            i = m + 1;  
11.     }  
12.     return -1; // khong tim thay  
13. }
```

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:

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


Giải phương trình đệ quy

- Phương pháp truy hồi
 - Triển khai $T(n)$ theo $T(n - 1)$ rồi $T(n - 2) \dots$ 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 $T(n-2) \dots$ cho đến $T(1)$
- Ví dụ: giải phương trình đệ quy:
 - $T(1) = C_1$
 - $T(n) = 2T(n-1) + C_2$
- Ta có:
 - $T(n) = 2T(n-1) + C_2$
 - $= 2(2T(n-2) + C_2) + C_2 = 2^2T(n-2) + 2C_2 + C_2$
 - $= 2^2(2T(n-3) + C_2) + 2C_2 + C_2$
 - $= 2^3T(n-3) + 2^2C_2 + 2C_2 + C_2$
 - $= \dots$
 - $= 2^kT(n-k) + 2^{k-1}C_2 + 2^{k-2}C_2 + \dots + 2C_2 + C_2$
 - Quá trình dừng lại khi $n-k = 1$ hay $k = n-1$, khi đó:
 - $T(n) = 2^{n-1}T(1) + C_2(2^{n-1} - 1)$
 - $= C_12^{n-1} + C_2(2^{n-1} - 1) = O(2^n)$

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 , $\log n$, $n \log n$, 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:
 - $T(n) = aT(n/b) + d(n)$
 - $= a[aT(n/b/b) + d(n/b)] + d(n)$
 - $= a^2T(n/b^2) + ad(n/b) + d(n)$
 - $= a^2[aT(n/b^3) + d(n/b^2)] + ad(n/b) + d(n)$
 - $= a^3T(n/b^3) + a^2d(n/b^2) + ad(n/b) + d(n)$
 - $= \dots$
 - $= a^kT(n/b^k) + \sum a^i d(n/b^i)$
 - 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:

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

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 $T(1) = 1$

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

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

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

4/

$$T(1) = 1$$

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

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 $n > 1$

Bài tập

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

$$C_n^k = \begin{cases} 1 & \text{nếu } k = 0 \text{ hoặc } k = n \\ C_{n-1}^{k-1} + C_{n-1}^k & \text{nếu } 0 < k < n \end{cases}$$

- 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 :
- 1. `int max(int a[], int n) {`
- 2. `if (n == 1)`
- 3. `return a[0];`
- 4. `if (a[n-1] > max(a, n - 1))`
- 5. `return a[n-1];`
- 6. `return max(a, n-1);`
- 7. `}`