Chương 1.
THUẬT TOÁN

ThS. Nguyễn Chí Hiếu

2017

NỘI DUNG

1. Giới thiệu thuật toán

2. Phương pháp biểu diễn thuật toán

3. Độ phức tạp thuật toán

Giới thiệu thuật toán

Định nghĩa

- Thuật toán *(algorithm - tên một nhà toán học người Trung Á là Abu Abd - Allah ibn Musa al'Khwarizmi, thường gọi là al'Khwarizmi)* là tập hợp hữu hạn các hướng dẫn rõ ràng để giải quyết một bài toán/vấn đề.

✍ Trong lĩnh vực máy tính, thuật toán là một dãy *hữu hạn* các bước *không mập mờ* và *thực thi được*, quá trình hành động theo các bước này *phải dừng* và cho được *kết quả như mong muốn*.

Giới thiệu thuật toán

Ví dụ 1

Cho m, n là hai số nguyên dương, ước chung lớn nhất của m và n được định nghĩa theo công thức:

$$gcd\left(m,n\right) = \left\{\forall m, n \in \mathcal{Z}, \exists g \in \mathcal{Z} : max\left\{g|m \wedge g|n\right\}\right\}.$$

trong đó, gcd *(greastest common divisor)* là ước chung lớn nhất.

- $gcd\left(3,5\right) = 1$
- $gcd\left(36,12\right) = 12$
- $gcd\left(27,15\right) = 3$

Giới thiệu thuật toán

Thuật toán **Euclid** tìm ước chung lớn nhất của hai số nguyên dương.

- Bước 1: [Kiểm tra $m \geq n$] Nếu $m < n$ thì hoán vị m và n.
- Bước 2: [Tìm số dư r_i] Chia m cho n được số dư r_i, với $0 \leq r_i < n$ và $i \geq 0$.
- Bước 3: [Kiểm tra $r_i = 0$]
  - Nếu $r_i = 0$ thì thuật toán kết thúc. Trả về kết quả ước số chung lớn nhất là $n = (r_{i-1})$
  - Ngược lại, thực hiện bước 4.
- Bước 4: [Cập nhật giá trị m và n] Gán $m \leftarrow n$ và $n \leftarrow r$. Quay lại bước 2.

Giới thiệu thuật toán

Tìm ước chung lớn nhất của 27 và 15.

- Cho $m = 27, n = 15$ và r là số dư của phép chia m cho n $(r = m \mod n)$.
- Thực hiện các bước theo thuật toán tìm ước chung lớn nhất của m và n.

	m	n	r
1			

Giới thiệu thuật toán

Tìm ước chung lớn nhất của 27 và 15.

- Cho $m = 27, n = 15$ và r là số dư của phép chia m cho n.
- Thực hiện các bước theo thuật toán tìm ước chung lớn nhất của m và n.

	m	n	r
1	27	15	12
2			

Giới thiệu thuật toán

Tìm ước chung lớn nhất của 27 và 15.

- Cho $m = 27, n = 15$ và r là số dư của phép chia m cho n.
- Thực hiện các bước theo thuật toán tìm ước chung lớn nhất của m và n.

	m	n	r
1	27	15	12
2	15	12	3
3			

Giới thiệu thuật toán

Tìm ước chung lớn nhất của 27 và 15.

- Cho $m = 27, n = 15$ và r là số dư của phép chia m cho n.
- Thực hiện các bước theo thuật toán tìm ước chung lớn nhất của m và n.

	m	n	r
1	27	15	12
2	15	12	$\boxed{3}$
3	12	$\boxed{3}$	0

- Kết luận: $gcd\,(27, 15) = 3$.

Giới thiệu thuật toán

Một thuật toán nên thỏa các tính chất sau:

- Tính xác định = không mập mờ + thực thi được
- Tính hữu hạn
- Tính chính xác
- Đầu vào và đầu ra phải rõ ràng
- Tính hiệu quả
- Tính tổng quát

Giới thiệu thuật toán

Tính xác định

- Mỗi bước của thuật toán phải được định nghĩa rõ ràng.

Ví dụ 2

Thuật toán Euclid tìm ước chung lớn nhất của hai số m và n.

- Bước 1: [Kiểm tra $m \geq n$]

Giới thiệu thuật toán

Tính hữu hạn

- Thuật toán phải kết thúc sau một số bước thực hiện.

Ví dụ 3

Thuật toán Euclid tìm ước chung lớn nhất của hai số nguyên dương m và n.

- Nếu $r \neq 0$ thì thuật toán tiếp tục thực hiện quá trình giảm giá trị r.
- Ngược lại $r = 0$, thuật toán kết thúc.

Số bước thực hiện sẽ phụ thuộc giá trị 2 số m và n.

Giới thiệu thuật toán

Tính chính xác

- Thuật toán phải tạo những giá trị đầu ra chính xác tương ứng với mỗi tập giá trị đầu vào.

Ví dụ 4

- Số nguyên tố là số nguyên dương chỉ chia hết cho 1 và chính nó.
- Số nguyên tố là số nguyên dương n bit không chia hết cho bất kỳ số nào có số bit $\leq \frac{n}{2}$ bit và lớn hơn 1?

Giới thiệu thuật toán

Số nguyên tố là số nguyên dương n bit không chia hết cho bất kỳ số nào có số bit $\leq \frac{n}{2}$ bit và lớn hơn 1?

- Xét $m = 9 = 1001_2$.
  - Vì $n = 4$ bit, nên các số nguyên biểu diễn dưới dạng nhị phân $\leq \frac{n}{2}$ bit và lớn hơn 1 gồm: $10_2 = 2, 11_2 = 3$.
  - Ta có, m *chia hết* 3.
  - Kết luận: m *không là số nguyên tố*.

Giới thiệu thuật toán

Số nguyên tố là số nguyên dương n bit không chia hết cho bất kỳ số nào có số bit $\leq \frac{n}{2}$ bit và lớn hơn 1?

- Xét $m = 9 = 1001_2$.
  - Vì $n = 4$ bit, nên các số nguyên biểu diễn dưới dạng nhị phân $\leq \frac{n}{2}$ bit và lớn hơn 1 gồm: $10_2 = 2, 11_2 = 3$.
  - Ta có, m *chia hết* 3.
  - Kết luận: m *không là số nguyên tố*.
- Xét $m = 25 = 11001_2$.
  - Vì $n = 5$ bit, nên các số nguyên biểu diễn dưới dạng nhị phân $\leq \frac{n}{2}$ bit và lớn hơn 1 gồm: $10_2 = 2, 11_2 = 3$.
  - Ta có, m không chia hết cho 2 và 3.
  - Có thể kết luận m *là số nguyên tố* ?
  - Thực tế, m chia hết cho $101_2 = 5$.

Giới thiệu thuật toán

Tính chính xác

- Thuật toán phải tạo những giá trị đầu ra chính xác tương ứng với mỗi tập giá trị đầu vào.

Ví dụ 5

- Số nguyên tố là số nguyên dương chỉ chia hết cho 1 và chính nó.
- Số nguyên tố là số nguyên dương n bit không chia hết cho bất kỳ số nào có số bit $\leq \frac{n}{2} + 1$ bit và lớn hơn 1?

Giới thiệu thuật toán

Đầu vào và đầu ra của thuật toán (input, output)

- Đầu vào của thuật toán được lấy từ một tập xác định. Từ mỗi tập giá trị đầu vào, thuật toán sẽ tạo tập giá trị đầu ra tương ứng.

Ví dụ 6

Thuật toán Euclid tìm ước chung lớn nhất của hai số nguyên dương m và n.

- Đầu vào: hai số nguyên dương m và n.
- Đầu ra: một số nguyên dương là ước chung lớn nhất của m và n.

Giới thiệu thuật toán

Tính hiệu quả

- Một thuật toán hiệu quả phải có độ phức tạp *thời gian* và *không gian* thực hiện *nhỏ hơn* các thuật toán khác.

Ví dụ 7

Tính tổng $1.000.000$ số nguyên dương đầu tiên.

- Cách 1: thực hiện thao tác lặp $1.000.000$ lần, mỗi lần cộng với một số nguyên dương.
- Cách 2: sử dụng công thức Gauss tính tổng n số nguyên dương đầu tiên $S = \dfrac{n(n+1)}{2}$.

Giới thiệu thuật toán

Tính tổng quát

- Thuật toán phải áp dụng được cho mọi trường hợp của bài toán có dạng theo yêu cầu.

Ví dụ 8

Giải phương trình bậc hai $ax^2 + bx + c = 0, a \neq 0$ dựa vào Delta.

- Thuật toán này luôn giải được với giá trị hệ số a, b, c bất kỳ.

Phương pháp biểu diễn thuật toán

- Ngôn ngữ tự nhiên *(natural language)*
- Sơ đồ khối *(flowchart)*
- Mã giả *(pseudocode)*

Ngôn ngữ tự nhiên

Khái niệm

- Sử dụng ngôn ngữ bình thường để mô tả thuật toán.
- Thường viết theo dạng phân cấp 1, 1.1, 1.1.1
- Không có cấu trúc, dài dòng.

Ví dụ 9

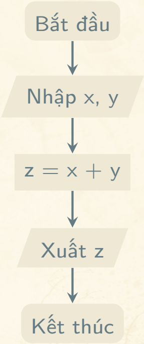
Nhập một số nguyên n. Cho biết n là số chẵn hay số lẻ.

- Bước 1: Nhập số nguyên n.
- Bước 2: Kiểm tra $n \mod 2 = 0$:
  - Bước 2.1: Nếu $n \mod 2 = 0$, thông báo n là số chẵn và chuyển qua bước 3.
  - Bước 2.2: Ngược lại, thông báo n là số lẻ.
- Bước 3: Kết thúc thuật toán.

Sơ đồ khối

Khái niệm

- Là công cụ trực quan để mô tả các thuật toán.
- Sử dụng các hình đại diện tương ứng với những thao tác trong thuật toán.



Sơ đồ khối

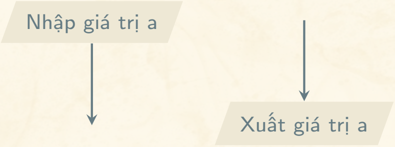
Điểm cuối

- Biểu diễn bằng hình ovan và được ghi chú *Bắt đầu* hay *Kết thúc*.
- Chỉ ra điểm bắt đầu hay kết thúc của thuật toán.

Sơ đồ khối

Đầu vào, đầu ra

- Biểu diễn bằng hình bình hành.
- Nhập, xuất các giá trị trong thuật toán.



Sơ đồ khối

Thao tác xử lý

- Biểu diễn bằng hình chữ nhật.
- Chứa nội dung các xử lý toán học, gán giá trị.

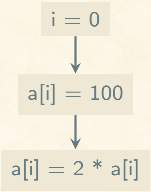
i = 0   b = 1024

a[i] = 100   c = sqrt(b)

Sơ đồ khối

Thao tác tuần tự

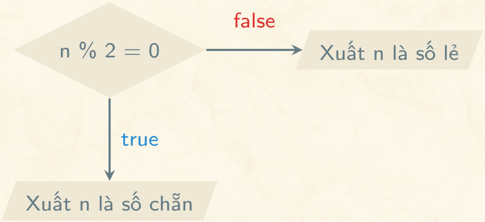
- Là một chuỗi các thao tác xử lý liên tiếp.
- Mũi tên thể hiện đường đi giữa các thao tác.



Sơ đồ khối

Thao tác chọn (rẽ nhánh)

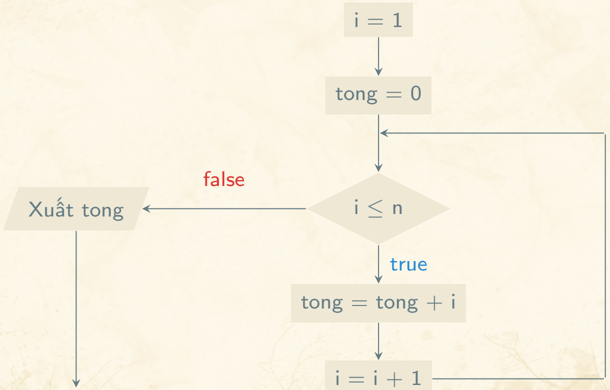
- Biểu diễn bởi hình thoi.
- Có hai đường đi tương ứng với thỏa hay không thỏa điều kiện.



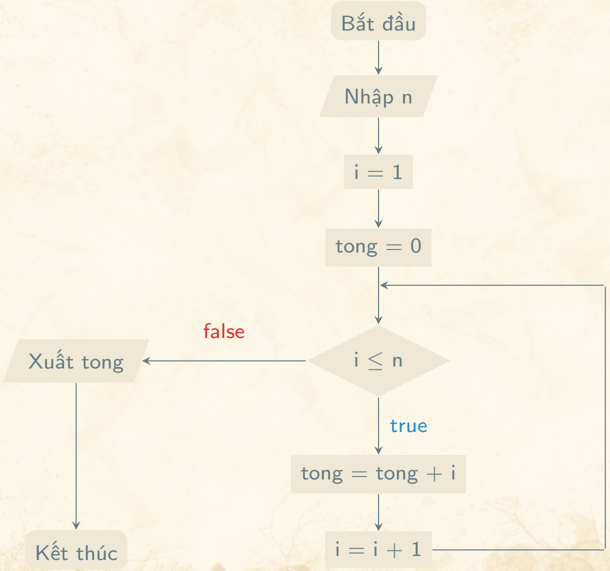
Sơ đồ khối

Thao tác lặp

- Được kết hợp từ nhiều thao tác liên tiếp nhau.
- Chia 2 loại: vòng lặp xác định và vòng lặp không xác định.



Sơ đồ khối


Bắt đầu
Nhập n
i = 1
tong = 0
i ≤ n
false
Xuất tong
true
tong = tong + i
i = i + 1
Kết thúc

Mã giả

Khái niệm

- Mã giả là phương pháp biểu diễn thuật toán có sử dụng cú pháp của một ngôn ngữ lập trình nào đó.

Ví dụ 10

Tính tổng dãy số $1 + 2 + 3 + \cdots + n$ với $n > 0$.

```
Thuật toán 1: Sum(n)
- Đầu vào: số nguyên dương n.
- Đầu ra: tổng các số nguyên dương từ 1 → n.
```

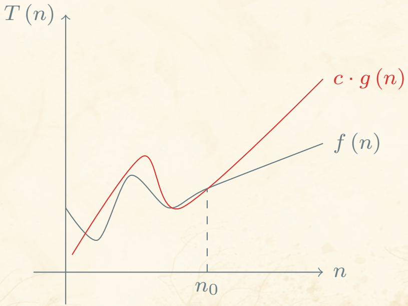
```
sum ← 0
if n > 0
  for i ← 1 to n
    sum ← sum + i
return sum
```

Các ký hiệu tiệm cận

Định nghĩa (Big-Oh)

Hàm $f(n)$ là $O(g(n))$ nếu f có tỷ lệ tăng trưởng *(growth rate)* nhiều khi đến g:

$$\exists c, n_0 \in R^+, \forall n \geq n_0 : f(n) \leq c \cdot g(n).$$

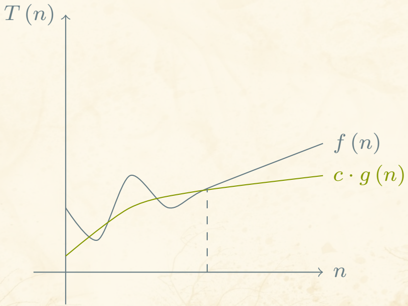


Các ký hiệu tiệm cận

Định nghĩa (Big-Omega)

Hàm $f(n)$ là $\Omega(g(n))$ nếu f có tỷ lệ tăng trưởng ít khi đến g:

$$\exists c \in R^+, \forall n \in N : f(n) \geq c \cdot g(n).$$

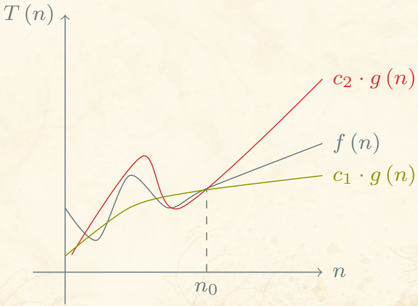


Các ký hiệu tiệm cận

Định nghĩa (Big-Theta)

Hàm $f(n)$ là $\Theta(g(n))$ nếu và chỉ nếu:

$$\exists c_1, c_2, n_0 \in R^+, \forall n \geq n_0 : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n).$$



Kích thước dữ liệu đầu vào của một thuật toán

Ví dụ 11

- Tìm kiếm, sắp xếp: $n =$ số phần tử của mảng.
- Xử lý chuỗi: $n =$ chiều dài chuỗi.
- Ma trận: $n =$ số chiều ma trận (trường hợp ma trận vuông), $n \times n$ phần tử.
- Đồ thị: $n_V =$ số đỉnh và $n_E =$ số cạnh của đồ thị.

Độ phức tạp thời gian

Các bước phân tích độ phức tạp thời gian

1. Xác định phép toán cơ sở.
2. Tính số lần thực hiện phép toán cơ sở của một hàm.

Công thức tính độ phức tạp thời gian

$$T(n) \approx c \cdot g(n).$$

Trong đó,

- $T(n)$: thời gian chạy của hàm,
- c: thời gian chạy của phép toán cơ sở,
- $g(n)$: số lần thực hiện các phép toán cơ sở của một hàm với n là kích thước dữ liệu đầu vào.

Độ phức tạp thời gian

Phân lớp độ phức tạp thời gian

Bảng 1: Một số lớp độ phức tạp thời gian của thuật toán.

Độ phức tạp	Thuật ngữ
$O(1)$	Độ phức tạp hằng số
$O(\log n)$	Độ phức tạp logarit
$O(n)$	Độ phức tạp tuyến tính
$O(n \log n)$	Độ phức tạp $n \log n$
$O\left(n^b\right), b > 1$	Độ phức tạp đa thức
$O(b^n)$	Độ phức tạp hàm mũ
$O(n!)$	Độ phức tạp giai thừa

Độ phức tạp thời gian

QUY TẮC CỘNG: nếu $T_1(n) = O(g_1(n))$ và $T_2(n) = O(g_2(n))$ là thời gian thực hiện của 2 đoạn chương trình P_1 và P_2 thì thời gian thực hiện của 2 đoạn chương trình đó *nối tiếp nhau* là

$$T(n) = O(g_1(n) + g_2(n)).$$

Hay lấy giá trị của $g_i(n)$ lớn nhất

$$T(n) = O(max(g_1(n), g_2(n))).$$

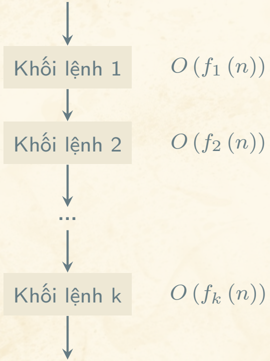
QUY TẮC NHÂN: nếu $T_1(n) = O(g_1(n))$ và $T_2(n) = O(g_2(n))$ là thời gian thực hiện của 2 đoạn chương trình P_1 và P_2 thì thời gian thực hiện của 2 đoạn chương trình đó *lồng vào nhau* là

$$T(n) = O(g_1(n) \cdot g_2(n)).$$

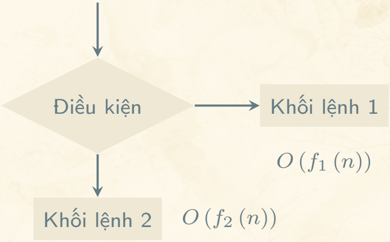
Độ phức tạp thời gian

- Phép toán cơ sở (so sánh, gán, ...): $O(1)$.
- Các phép toán nối tiếp nhau: quy tắc cộng.
- Cấu trúc điều kiện (`if`): là thời gian lớn nhất sau `if` hay `else` và thời gian kiểm tra điều kiện (thường thời gian kiểm tra điều kiện là $O(1)$).
- Cấu trúc lặp (`for, while`): quy tắc nhân.

Độ phức tạp thời gian

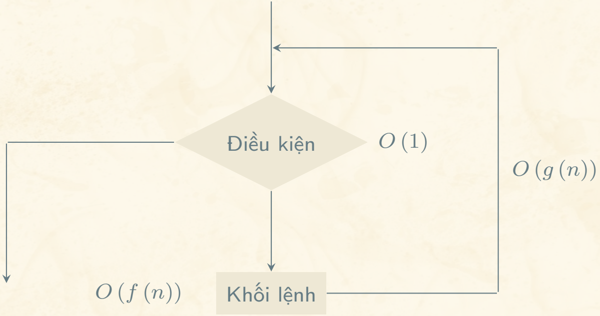


$$O\left(max\left\{f_1\left(n\right), f_2\left(n\right), \ldots, f_k\left(n\right)\right\}\right)$$



$$O\left(max\left\{f_1\left(n\right), f_2\left(n\right)\right\}\right)$$

Độ phức tạp thời gian



$$O(f(n) \cdot g(n))$$

Độ phức tạp thuật toán không đệ quy

Ví dụ 12

Tìm số lớn nhất trong mảng a gồm n phần tử cho trước.

```
Thuật toán 2: Max(a[], n)
- Đầu vào: mảng a gồm n phần tử.
- Đầu ra: trả về phần tử lớn nhất trong mảng a.
```

```
max ← a[0]
for i ← 1 to n - 1
  if max < a[i]
    max ← a[i]
return max
```

- Phép tính cơ sở: phép so sánh `max < a[i]`.
- Thời gian thực hiện: $T(n) = \sum_{i=1}^{n-1} 1 = n - 1 = O(n)$.

Độ phức tạp thuật toán không đệ quy

Ví dụ 13

Kiểm tra các phần tử trong mảng A có trùng nhau hay không?

Thuật toán 3: `IsDuplicate(a[], n)`
- Đầu vào: mảng a gồm n phần tử.
- Đầu ra: trả về true/false.

```
for i ← 0 to n - 2
  for j ← i + 1 to n - 1
    if a[i] = a[j]
      return true
return false
```

- Phép tính cơ sở: phép so sánh `a[i] = a[j]`.
- Thời gian thực hiện: $T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} n - 1 - i = \frac{n(n-1)}{2} = O(n^2)$.

Độ phức tạp thuật toán đệ quy

Phương pháp lặp (iteration method)

- Mở rộng quá trình đệ quy k lần. $k = ?$
- Thực hiện tính toán để tìm được công thức tính tổng.
- Ước lượng công thức vừa tìm được để đưa về một lớp độ phức tạp thời gian của thuật toán.

Phương pháp lặp còn được gọi là phương pháp thay thế.

Định lý chủ (master theorem)

- Thường sử dụng đối với các thuật toán sử dụng kỹ thuật chia để trị *(divide and conquer)*.

Cấp số nhân

Định ngĩa

Cấp số nhân là một dãy số (*hữu hạn* hay *vô hạn*) với mỗi số hạng *(trừ số hạng đầu tiên)* đều bằng tích của số hạng đứng ngay trước nó và một số r không đổi.

- Số hạng thứ i của cấp số nhân:

$$a_i = ar^{i-1}, \quad i \geq 1.$$

- Số r được gọi là công bội của cấp số nhân.

Ví dụ 14

Các dãy số sau là một cấp số nhân.

- $1, 3, 9, 27$
- $1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}$

Cấp số nhân

Tổng n số hạng đầu tiên

Giả sử có cấp số nhân với công bội r. Khi đó, tổng n số hạng đầu tiên:

$$S_n = a + ar^1 + ar^2 + \cdots + ar^n. \tag{1}$$

- Dãy hữu hạn: nếu $r > 1$, thì

$$S_n = a\sum_{i=0}^{n} r^i = \frac{a\left(1 - r^n\right)}{1 - r}. \tag{2}$$

- Dãy vô hạn: với $0 < r < 1$, thì

$$S = a\sum_{i=0}^{\infty} r^i = \frac{1}{1 - r} \tag{3}$$

Phương pháp lặp

Công thức 1

$$T(n) = \begin{cases} c_0 & , n = 0 \\ T(n-1) + cn & , n > 0 \end{cases}$$

Chứng minh

$$\begin{aligned} T(n) &= T(n-1) + cn \\ &= T(n-2) + c(n-1) + cn \\ &= T(n-3) + c(n-2) + c(n-1) + cn \\ &\dots \\ &= T(n-k) + c(n-k+1) + \dots + c(n-2) + c(n-1) + cn \\ &= T(n-k) + c \cdot \sum_{i=n-k+1}^{n} i \qquad , n \geq k \end{aligned}$$

Phương pháp lặp

Chứng minh

- *Giả sử* $n = k$*, thuật toán dừng đệ quy.*

$$T(n) = T(0) + c \cdot \sum_{i=1}^{n} i = c_0 + c \cdot \frac{n(n+1)}{2}$$

- *Do đó,*

$$T(n) = \frac{n(n+1)}{2} \approx \frac{n^2}{2} = O(n^2).$$

✍ CÔNG THỨC 1 *thường dùng cho chương trình đệ quy có vòng lặp duyệt qua dữ liệu nhập để bỏ một phần tử.*

Phương pháp lặp

Công thức 2

$$T(n) = \begin{cases} c_0 & , n = 1 \\ T\left(\frac{n}{2}\right) + c & , n > 1 \end{cases}$$

Chứng minh

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + c = T\left(\frac{n}{4}\right) + 2c = T\left(\frac{n}{8}\right) + 3c \\ &\dots \\ &= T\left(\frac{n}{2^k}\right) + k \cdot c \qquad , n \geq 2^k \end{aligned}$$

Phương pháp lặp

Chứng minh

- *Giả sử* $n = 2^k$, *thuật toán dừng đệ quy.*
  $T(n) = T(1) + k \cdot c = c_0 + k \cdot c = k = \log n$
- *Do đó,*
$$T(n) = O(\log n).$$

✍ Công thức 2 *thường dùng cho chương trình đệ quy mà dữ liệu nhập được chia thành hai phần mỗi bước thực hiện.*

Phương pháp lặp

Công thức 3

$$T(n) = \begin{cases} c_0 & , n = 1 \\ T\left(\dfrac{n}{2}\right) + n & , n > 1 \end{cases}$$

Chứng minh

$$\begin{aligned} T(n) &= T\left(\tfrac{n}{2}\right) + n \\ &= T\left(\tfrac{n}{4}\right) + \tfrac{n}{2} + n \\ &= T\left(\tfrac{n}{8}\right) + \tfrac{n}{4} + \tfrac{n}{2} + n \\ &\ldots \\ &= T\left(\tfrac{n}{2^k}\right) + \tfrac{n}{2^{k-1}} + \ldots + \tfrac{n}{4} + \tfrac{n}{2} + n \qquad , n \geq 2^k \\ &= T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} \frac{n}{2^i} \end{aligned}$$

Phương pháp lặp

Chứng minh

- *Giả sử* $n = 2^k$, *thuật toán dừng đệ quy.*
  $$T(n) = T(1) + 2^k \sum_{i=0}^{k-1} \frac{1}{2^i} = c_0 + 2^k \sum_{i=0}^{k-1} \frac{1}{2^i} = 2^k \sum_{i=0}^{k-1} \frac{1}{2^i}$$
- *Tính tổng số hạng của cấp số nhân của dãy số có dạng* $\sum_{i=0}^{m} \frac{1}{2^i}$
  - *Ta có, phần tử đầu tiên của chuỗi là* $a = 1$ *và công bội* $r = \frac{1}{2}$.
  - *Áp dụng công thức tính tổng của dãy số vô hạn* $\sum_{i=0}^{\infty} \frac{1}{2^i} = \frac{1}{1-r} = 2$
- *Do đó,* $\sum_{i=0}^{k-1} \frac{1}{2^i} = \frac{1}{1-r} = 2$

Phương pháp lặp

Chứng minh

- *Do đó,*

$$T\left(n\right) = 2^k \cdot 2 = 2n = O\left(n\right).$$

✍ Công thức 3 *thường dùng cho chương trình đệ quy mà dữ liệu nhập được chia thành hai phần nhưng có thể kiểm tra mỗi phần tử của dữ liệu nhập.*

Công thức 4

$$T(n) = \begin{cases} c_0 & , n = 1 \\ 2T\left(\frac{n}{2}\right) + n & , n > 1 \end{cases}$$

Chứng minh

$$\begin{aligned} T(n) &= 2T\left(\tfrac{n}{2}\right) + n \\ &= 4T\left(\tfrac{n}{4}\right) + n + n \\ &= 8T\left(\tfrac{n}{8}\right) + n + n + n \\ &\ldots \\ &= nT\left(\frac{n}{2^k}\right) + k \cdot n \qquad , n \geq 2^k \end{aligned}$$

Phương pháp lặp

Chứng minh

- *Giả sử* $n = 2^k$*, thuật toán dừng đệ quy.*
$$\begin{aligned} T(n) &= 2^k \cdot T(1) + k \cdot 2^k \\ &= 2^k \cdot c_0 + k \cdot 2^k \\ &= n \cdot c_0 + n \log n \end{aligned}$$
- *Do đó,*
$$T(n) = O(n \log n).$$

✍ CÔNG THỨC 4 *thường dùng cho chương trình đệ quy mà duyệt tuyến tính dữ liệu nhập trước, trong hay sau khi được chia thành hai phần.*

Phương pháp lặp

Ví dụ 15

Công thức 5

$$T(n) = \begin{cases} c_0 & , n = 1 \\ 2T\left(\frac{n}{2}\right) + c & , n > 1 \end{cases}$$

Chứng minh

$$\begin{aligned} T(n) &= 2T\left(\tfrac{n}{2}\right) + c \\ &= 4T\left(\tfrac{n}{4}\right) + 2c + c \\ &= 8T\left(\tfrac{n}{8}\right) + 4c + 2c + c \\ &\ldots \\ &= nT\left(\frac{n}{2^k}\right) + c \cdot \sum_{i=0}^{k-1} 2^i \qquad , n \geq 2^k \end{aligned}$$

Phương pháp lặp

Chứng minh

- *Giả sử* $n = 2^k$, *thuật toán dừng đệ quy.*

$$T(n) = 2^k \cdot T(1) + c \cdot \sum_{i=0}^{k-1} 2^i = 2^k \cdot c_0 + c \cdot \sum_{i=0}^{k-1} 2^i.$$

Phương pháp lặp

Chứng minh

- *Giả sử* $n = 2^k$*, thuật toán dừng đệ quy.*

$$T(n) = 2^k \cdot T(1) + c \cdot \sum_{i=0}^{k-1} 2^i = 2^k \cdot c_0 + c \cdot \sum_{i=0}^{k-1} 2^i.$$

- *Tính tổng số hạng của cấp số nhân hữu hạn có dạng* $\sum_{i=0}^{m} 2^i$
  - *Ta có, phần tử đầu tiên là* $a = 1$ *và công bội* $r = 2$.
  - *Áp dụng công thức tính tổng của dãy số hữu hạn* $\sum_{i=0}^{m} 2^i = \frac{a(1-r^m)}{1-m} = 2^m - 1$.
- *Do đó,* $\sum_{i=0}^{k-1} 2^i = 2^{k-1} - 1$.

Phương pháp lặp

Chứng minh

- *Do đó,*

$$T\left(n\right) = 2^k \cdot c_0 + 2^{k-1} \cdot c - c = n \cdot c_0 + \frac{n}{2} \cdot c - c = O\left(n\right).$$

✍ Công thức 5 *thường dùng cho chương trình đệ quy mà mỗi bước thực hiện dữ liệu được chia thành hai phần.*

Định lý chủ

Cho $a \geq 1, b > 1$ và $T(n)$ là độ phức tạp thời gian của thuật toán

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \tag{4}$$

với $f(n) \in O\left(n^d\right), d \geq 0$

- b: số bài toán con (thường là 2).
- a: số bài toán con cần được xử lý.
- $f(n)$: chi phí chia bài toán con và chi phí tổng hợp kết quả.

- Nếu $a < b^d \Rightarrow T(n) = O\left(n^d\right)$.
- Nếu $a = b^d \Rightarrow T(n) = O\left(n^d \log n\right)$.
- Nếu $a > b^d \Rightarrow T(n) = O\left(n^{\log_a b}\right)$.

Định lý chủ

Ví dụ 16

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

  $\begin{cases} a = 4, b = 2, d = 3 \\ a < b^d \end{cases} \Rightarrow \mathsf{T}(n) = O\left(n^3\right).$

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

  $\begin{cases} a = 4, b = 2, d = 2 \\ a = b^d \end{cases} \Rightarrow \mathsf{T}(n) = O\left(n^2 \log n\right).$

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

  $\begin{cases} a = 2, b = 2, d = 0 \\ a > b^d \end{cases} \Rightarrow \mathsf{T}(n) = O\left(n^{\log_2 2}\right) = O(n).$

Bài tập

Tính độ phức tạp thời gian các thuật toán

1. Thuật toán giải bài toán Tháp Hà Nội *(Towers of Hanoi)*

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

2. Thuật toán tính n!

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

Bài tập

3. Thuật toán sắp xếp trộn *(MergeSort)*

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

4. Thuật toán Strassen nhân hai ma trận

$$T(n) = \begin{cases} 1 & , n = 1 \\ 7T\left(\frac{n}{2}\right) + n^2 & , n > 1 \end{cases}$$

Tài liệu tham khảo

Dương Anh Đức, Trần Hạnh Nhi.
Nhập môn Cấu trúc dữ liệu và Thuật toán.
Đại học Khoa học tự nhiên TP Hồ Chí Minh, 2003.

Donald E. Knuth.
The Art of Computer Programming, Volume 3.
Addison-Wesley, 1998.

Niklaus Wirth.
Algorithms + Data Structures = Programs.
Prentice-Hall, 1976.

Robert Sedgewick.
Algorithms in C.
Addison-Wesley, 1990.