Giới thiệu về Cấu trúc dữ liệu và Giải thuật
Độ Phức Tạp Thuật Toán

Bùi Ngọc Thăng

September 28, 2015

Outline

Mục tiêu bài học

1. Tìm hiểu các khái niệm về độ phức tạp thuật toán (thời gian, không gian)
2. Tìm hiểu khái niệm $\mathcal{O}(n)$ (Ô lớn)
3. Áp dụng để tính toán độ phức tạp của các thuật toán.

Tổng quan về độ phức tạp

Một thuật toán "tốt" cần thoả mãn 02 điều kiện sau:

1. Cho ra kết quả đúng → tính đúng đắn của thuật toán
2. Phải hiệu quả (nhanh) → **Độ phức tạp của thuật toán–Complexity of algorithm**

Độ phức tạp thuật toán là khái niệm dùng để:

1. **ước lượng tốc độ thực hiện của thuật toán** (độ phức tạp thời gian–time complexity)
2. **ước lượng thuật toán sử dụng bao nhiêu tài nguyên bộ nhớ** (độ phức tạp không gian–space complexity)

Ví dụ minh họa: Bài toán tìm kiếm

Bài toán: Tìm kiếm một phần tử trong một dãy số nguyên.
Câu hỏi: Số lượng "công việc" mà thuật toán thực hiện
Hàm: *find*(*arr*, *len*, *key*)
Đặc tả:
input: *arr* – mảng các số nguyên, *len*–số lượng phần tử trong mảng, *key*– số nguyên cần tìm
output: một số nguyên i $(0 \leq i \leq len)$ là chỉ số của phần tử trong mảng *arr* có giá trị bằng *key* hoặc giá trị -1 nếu không tồn tại phần tử *key* trong mảng

Ví dụ minh họa: Bài toán tìm kiếm (tiếp)

Câu hỏi: **Số lượng "công việc"** mà thuật toán thực hiện

Mã nguồn:

```
find(arr, len, key){
        i=0;
        while(i < len){
           if(arr[i] == key)
           return i;
          i++;
        }
        return -1;
}
```

Tổng quan về độ phức tạp- tốc độ thuật toán

Làm thế nào để ước lượng được tốc độ thực hiện (nhanh hay chậm) của thuật toán?

Có 02 vấn đề cần quan tâm khi thiết kế cách thức ước lượng:

1. Độc lập với ngôn ngữ lập trình (và phần mềm/phần cứng)
2. Giảm thiểu tối đa sự phụ thuộc vào dữ liệu vào.

$\Rightarrow$ Đếm các **phép toán cơ bản** của thuật toán.

Các thành phần chính của độ phức tạp thời gian

- Không cần thiết phải đếm tất cả các phép toán trong thuật toán– chỉ sử dụng các phép toán đại diện (hay dùng) trong thuật toán.
- Các phép toán thường dùng/phép toán đại diện(dominating operations): là các phép toán có số lượng thực hiện chiếm hầu hết "công việc" của thuật toán

Trước khi phân tích thuật toán, chúng ta cần 02 thành phần sau:

1. Xác định các **phép toán thường dùng**
2. Quan tâm tới các tham số ảnh hướng tới các phép toán thường dùng (**kích thước dữ liệu**)

Các phép toán thường dùng–dominating operations(tiếp)

Các phép toán nào sau đây là phép toán thường dùng?

Mã nguồn:

```
find(arr, len, key){
        i=0;
        while(i < len){
           if(arr[i] == key)
           return i;
          i++;
        }
        return -1;
}
```

Phép gán i=0 ?

Các phép toán thường dùng–dominating operations(tiếp)

Các phép toán nào sau đây là phép toán thường dùng?

Mã nguồn:

```
find(arr, len, key){
        i=0;
        while(i < len){
           if(arr[i] == key)
           return i;
          i++;
        }
        return -1;
}
```

Phép gán i = 0 ; return i ;

so sánh: i < len ; So sánh arr[i] == key ;

tăng chỉ số: i++

Xác định kích thước dữ liệu

Kích thước dữ liệu (data size) ảnh hưởng tới số lượng của các phép toán thường dùng?

```
find(arr, len, key){
        i=0;
        while(i < len){
           if(arr[i] == key)
           return i;
          i++;
        }
        return -1;
}
```

Cỡ của mảng *arr* là *len* ảnh hưởng tới các phép toán thường dùng
Xác định được các phép toán thường dùng (dominating operations) và cỡ dữ liệu, chúng ta xác định được **độ phức tạp thời gian**

Độ phức tạp thời gian của thuật toán

Definition

Độ phức tạp thời gian: Là số lượng các **phép toán thường dùng** được thực hiện bởi thuật toán được tính toán theo hàm với tham số là kích thước dữ liệu vào.

Độ phức tạp thời gian đánh giá số lượng "công việc" được thực hiện với thuật toán.
Độ phức tạp thời gian thấp thì thuật toán thực hiện càng nhanh

Độ phức tạp thời gian – ví dụ

```
find(arr, len, key){
        i=0;
        while(i < len){
           if(arr[i] == key)
           return i;
          i++;
        }
        return -1;
}
```

Giả sử:

– **phép toán thường dùng:** arr[i]==key

– **kích thước dữ liệu:** độ dài của mảng (ký hiệu là n)

$\Rightarrow$ Số lượng các phép toán thường dùng thuộc đoạn $[1, n]$

Độ phức tạp thời gian

Thông thường, độ phức tạp thời gian được ước lượng theo hai trường hợp

- Độ phức tạp trong trường hợp tồi nhất
- Độ phức tạp trung bình

Độ phức tạp tồi nhất

Giả sử một số khái niệm sau:
n – Kích thước dữ liệu
D_n – tập tất cả các tập dữ liệu vào có kích thước n
$t(d)$ – số lượng các phép toán thường dùng đối với tập dữ liệu d (có kích thước n) ($d \in D_n$)

Definition

Độ phức tạp tồi nhất: $W(n) = sup\{t(d) : d \in D_n\}$

($W(n)$– **W**orst)
Độ phức tạp tồi nhất biểu diễn số lượng các phép toán thường dùng trong **trường hợp tồi nhất** đối với tập dữ liệu đầu vào có kích thước n

Độ phức tạp tồi nhất – Ví dụ

```
find(arr, len, key){
        i=0;
        while(i < len){
            if(arr[i] == key)
            return i;
           i++;
        }
        return -1;
}
```

Độ phức tạp tồi nhất: $W(n) = n$
(n (len) là kích thước của mảng arr)

Độ phức tạp thời gian – Độ phức tạp trung bình

Giả sử một số khái niệm sau:
n – Kích thước dữ liệu
D_n – tập tất cả các tập dữ liệu vào có kích thước n
$t(d)$ – số lượng các phép toán thường dùng đối với tập dữ liệu d (có kích thước n) ($d \in D_n$)
X_n – một biến (biến ngẫu nhiên) nhận các giá trị $t(d)$ với $d \in D_n$
$P(X_n = k)$: Tính khả năng (xác suất) tập dữ liệu đầu vào (có kích thước n) sẽ thực thi k phép toán thường dùng ($0 \leq P(X_n) \leq 1$)

Definition

Độ phức tạp tồi nhất: $A(n) = \sum(P(X_n = k) \times k)$

($A(n)$– **A**verage)

Độ phức tạp trung bình – Ví dụ

```
find(arr, len, key){
        i=0;
        while(i < len){
           if(arr[i] == key)
           return i;
          i++;
        }
        return -1;
}
```

Giả sử: Giá trị cần tìm kiếm *key* chỉ xuất hiện đúng 1 lần trong mảng. Số lượng phép toán thường dùng $t(d) \in [1, n]$. $\Rightarrow$ $X_n = t(d) = k$ chỉ xuất hiện đúng một lần. Do đó, khả năng xuất hiện của $X_n = k$ là $1/n$ hay $P(X_n = k) = 1/n$.

Độ phức tạp trung bình – Ví dụ (tiếp)

```
find(arr, len, key){
        i=0;
        while(i < len){
            if(arr[i] == key)
            return i;
           i++;
        }
        return -1;
}
```

Độ phức tạp trung bình của thuật toán:

$A(n) = \sum(P(X_n = k) \times k) = \sum_{0 \leq k < n} \frac{1}{n} \times k = \frac{n+1}{2}$

Ô lớn: định nghĩa

Definition

Định nghĩa: Cho hai hàm số $f(n)$ và $g(n)$. Ta gọi $f(n)$ bằng Ô lớn của $g(n)$ nếu và chỉ nếu tồn tại hai số dương $c > 0$ và n_0 sao cho với mọi $n > n_0$ ta có: $f(n) \leq c \times g(n)$

$$f(n) = \mathcal{O}(g(n)) \iff \exists c\ \exists n_0\ \forall n \geq n_0 :\ f(n) \leq c \times g(n)$$

Ví dụ:

$$f(n) = \frac{n+1}{2} = \mathcal{O}(n)$$
$$f(n) = \frac{n^2+1}{2} = \mathcal{O}(n^2)$$
$$f(n) = \frac{n^2+n}{3} = \mathcal{O}(n^2)$$

Khái niệm Ô lớn: Các độ phức tạp thường dùng

Độ phức tạp	**Tên gọi**
$\mathcal{O}(1)$	Độ phức tạp hằng số
$\mathcal{O}(log(n))$	Độ phức tạp lôgarit
$\mathcal{O}(n)$	Độ phức tạp tuyến tính
$\mathcal{O}(nlog(n))$	Độ phức tạp *nlogn*
$\mathcal{O}(n^2)$	Độ phức tạp bình phương
$\mathcal{O}(n^3)$	Độ phức tạp lập phương
$\mathcal{O}(n^k)$	Độ phức đa thức (k là hằng số)
$\mathcal{O}(2^n)$	Độ phức tạp hàm mũ

Khái niệm Ô lớn: Quy tắc tổng

Definition

Định nghĩa: Giả sử $f_1(n) = \mathcal{O}(g_1(n))$ và $f_2(n) = \mathcal{O}(g_2(n))$ thì $f(n) = \mathcal{O}(\max\{g_1(n), g_2(n)\})$

Ví dụ:

$$f_1(n) = \mathcal{O}(n)$$
$$f_2(n) = \mathcal{O}(n^2)$$
$$f_1(n) + f_2(n) = \mathcal{O}(\max\{n, n^2\}) = \mathcal{O}(n^2)$$

Khái niệm Ô lớn: Tính độ phức tạp cho dòng lệnh

1. Cho các câu lệnh gán, đọc, ghi $\Rightarrow$ $\mathcal{O}(1)$
2. Cấu trúc if–else và switch
3. Cấu trúc lặp (for, while, do–while)

Tính độ phức tạp cho cấu trúc if –else và switch

if (condition)	$\mathcal{O}(g_1(n))$
S_1	$\mathcal{O}(g_2(n))$
else	
S_1	$\mathcal{O}(g_3(n))$

Độ phức tạp: $\mathcal{O}(g_1(n)) + \mathcal{O}(\max\{g_2(n), g_3(n)\})$

Thực hiện tương tự cho câu lệnh **switch**.

Tính độ phức tạp cho cấu trúc if –else: ví dụ

if (a<b)	$\mathcal{O}(1)$
$min = a$	$\mathcal{O}(1)$
else	
$min = b$	$\mathcal{O}(1)$

Độ phức tạp: $\mathcal{O}(1) + \mathcal{O}(\max\{1, 1\}) = \mathcal{O}(1)$

Tính độ phức tạp cho vòng lặp

for (Khởi tạo; điều kiện lặp; biểu thức tăng/giảm)
S_1 (Khối lệnh)

Khởi tạo	$\mathcal{O}(1)$
Điều kiện lặp	$\mathcal{O}(g_1(n))$
Biểu thức tăng/giảm	$\mathcal{O}(g_2(n))$
Khối lệnh	$\mathcal{O}(g_3(n))$

Độ phức tạp: $\mathcal{O}(g_1(n) \times \max\{g_2(n), g_3(n)\})$

Tương tự cho việc tính độ phức tạp thời gian đối với vòng lặp **while** và **do–while**

Tính độ phức tạp cho vòng lặp: ví dụ

for (i=1,sum=0; i<=n;i++) sum=sum+i;

Khởi tạo	$\mathcal{O}(1)$
Điều kiện lặp	$\mathcal{O}(n)$
Biểu thức tăng/giảm	$\mathcal{O}(1)$
Khối lệnh	$\mathcal{O}(1)$

Độ phức tạp: $\mathcal{O}(n \times \max\{1, 1\}) = \mathcal{O}(n)$

Câu hỏi và trả lời