HK I, 2012-2013

Bài 2: Phân tích thuật toán

Giảng viên: Hoàng Thị Điệp
Khoa Công nghệ Thông tin – Đại học Công Nghệ

A principle to respect whenever you program:

Pay attention to the cost!

http://introcs.cs.princeton.edu/java/41analysis/

Mục tiêu bài học

- Thuật toán: tính đúng đắn, tính hiệu quả
- Đo thời gian chạy bằng thực nghiệm
- Thời gian chạy tốt nhất, trung bình, xấu nhất
- Vấn đề đánh đổi không gian và thời gian
- Sử dụng kí hiệu ô lớn
  - Định nghĩa hình thức
  - Các cấp độ thời gian chạy
  - Kỹ thuật đánh giá thuật toán bởi ký hiệu ô lớn
    - Thuật toán không đệ quy
    - Thuật toán đệ quy

Giải thuật nào tốt hơn?

```
int factorial (int n) {
  if (n <= 1)    return 1;
  else   return n * factorial(n-1);
}

int factorial (int n) {
  if (n<=1)    return 1;
  else {
    fact = 1;
    for (k=2; k<=n; k++)
      fact *= k;
    return fact;
  }
}
```

Thuật toán

- Thuật toán được hiểu là sự đặc tả chính xác một dãy các bước có thể thực hiện được một cách máy móc để giải quyết một vấn đề
- Biểu diễn thuật toán
  - mã, giả mã, sơ đồ khối
- Tính đúng đắn (correctness)
  - đòi hỏi trước hết
- Tính hiệu quả (efficiency)
  - quan trọng

Đánh giá thuật toán

- Một vấn đề được giải quyết bởi nhiều thuật toán khác nhau
- Đối với một thuật toán:
  - Độ phức tạp về không gian (dung lượng bộ nhớ sử dụng)
  - Độ phức tạp về thời gian chạy
- Thời gian chạy
  - Kĩ năng lập trình
  - Chương trình dịch
  - Tốc độ thực hiện các phép toán trên máy tính
  - Dữ liệu vào

Thời gian chạy của thuật toán

- Thời gian chạy 1 thuật toán phụ thuộc vào cỡ (size) của dữ liệu vào
  - Tìm xem 1 đối tượng có trong danh sách N phần tử hay không?
  - Sắp xếp tăng dần dãy số gồm N số
  - Bài toán người bán hàng cần thăm N địa điểm
- Trong các dữ liệu vào cùng một cỡ (N), thời gian chạy của thuật toán cũng thay đổi

  Ví dụ: Tìm xem 1 đối tượng có trong danh sách N phần tử hay không?
  - Đối tượng nằm ở đầu danh sách
  - Đối tượng nằm ở giữa danh sách
  - Đối tượng nằm ở cuối danh sách

Hai cách tiếp cận

1. Phân tích thực nghiệm
   - Đo thời gian chạy, vẽ đồ thị, nội suy hàm
   - Dễ tiến hành thí nghiệm,
   - Phù hợp cho dự đoán, không phù hợp để giải thích
2. Phân tích toán học
   - Phân tích để ước lượng số phép toán như một hàm của kích thước dữ liệu vào
   - Có thể cần tới kiến thức toán cao cấp
   - Phù hợp cho cả dự đoán và giải thích

- Khác biệt quan trọng
  - Kết quả phân tích toán học độc lập với máy và trình biên dịch

Thời gian chạy của thuật toán

- Thời gian chạy trong trường hợp xấu nhất (worse-case running time)
  - Thời gian chạy lớn nhất của thuật toán đó trên tất cả các dữ liệu cùng cỡ
- Thời gian chạy trung bình (average running time)
  - Là trung bình cộng thời gian chạy trên tất cả các bộ dữ liệu cùng cỡ.
    - cần biết phân phối xác suất của dữ liệu vào
- Thời gian chạy trong trường hợp tốt nhất (best-case running time)
  - Thời gian chạy ít nhất của thuật toán đó trên tất cả các dữ liệu cùng cỡ

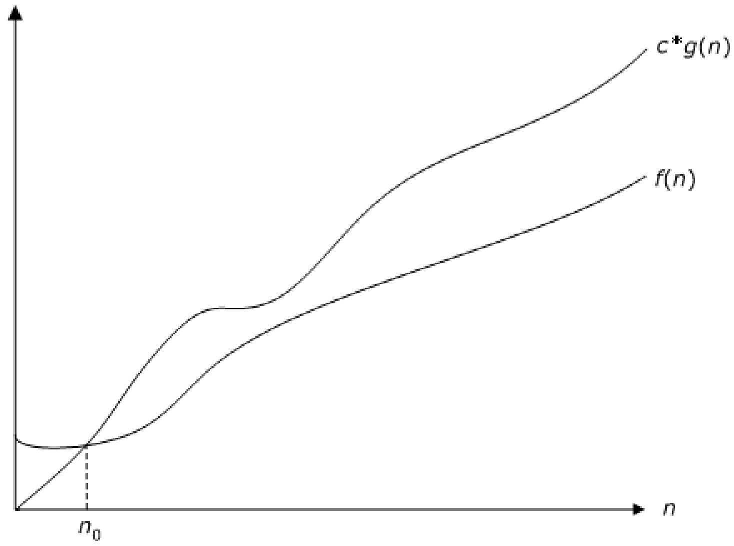
Độ phức tạp về thời gian

- Đánh giá thời gian chạy thuật toán:
  - T(n) = số lượng phép toán sơ cấp cần phải thực hiện (phép toán số học, phép toán logic, phép toán so sánh). Mỗi phép toán sơ cấp được thực hiện trong một khoảng thời gian cố định.
  - Ta chỉ quan tâm đến tốc độ tăng của hàm T(n)
  - Ví dụ:

    $T(n) = 2n^2 + 3n + 10$

Định nghĩa ký hiệu ô lớn

- Định nghĩa
  - Giả sử f(n) và g(n) là các hàm thực không âm của đối số nguyên không âm n.
  - Ta nói "f(n) là ô lớn của g(n)" và viết là f(n) = O(g(n)) nếu tồn tại các hằng số dương c và n_0 sao cho f(n) <= c*g(n) với mọi n >= n_0.



Biểu diễn thời gian chạy bởi kí hiệu O

- Ta sẽ lấy cận trên chặt (tight bound) để biểu diễn thời gian chạy của thuật toán.
- Ta nói f(n) là cận trên chặt của T(n) nếu
  – T(n) = O(f(n)), và
  – Nếu T(n) = O(g(n)) thì f(n) = O(g(n)).
- Nói cách khác
  – ta không thể tìm được một hàm g(n) là cận trên của T(n) mà lại tăng chậm hơn hàm f(n)

Biểu diễn thời gian chạy bởi kí hiệu O

- Ví dụ.
  - Giả sử $f(n) = 5n^3 + 2n^2 + 13n + 6$, ta có:

    $f(n) = 5n^3 + 2n^2 + 13n + 6 <= 5n^3 + 2n^3 + 13n^3 + 6n^3 = 26n^3$

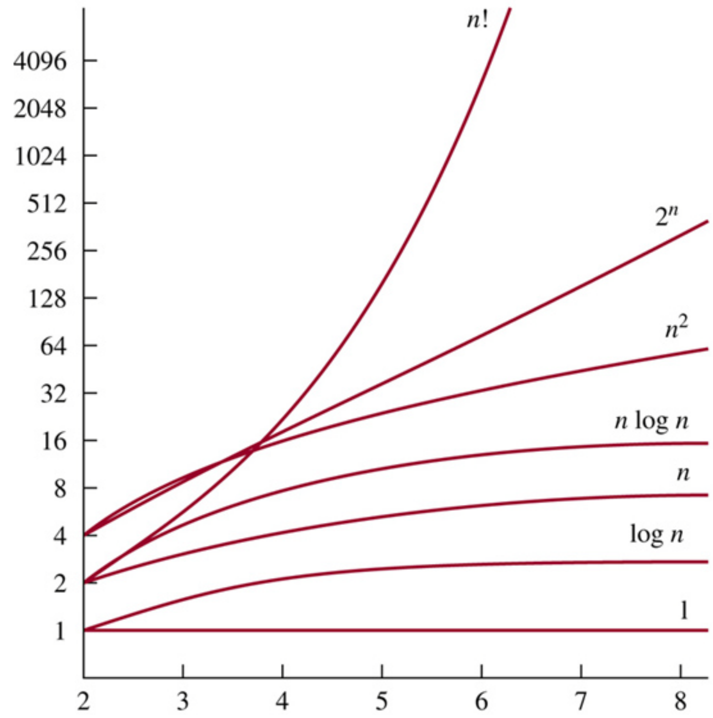
    $f(n) = O(n^3)$
  - Tổng quát nếu f(n) là một đa thức bậc k của n:

    $f(n) = a_k n^k + a_{k-1} n^{k-1} + ... + a_1 n + a_0$

    thì $f(n) = O(n^k)$

Các cấp độ thời gian chạy

Ký hiệu ô lớn	Tên gọi
$O(1)$	hằng
$O(\log n)$	logarit
$O(n)$	tuyến tính
$O(n \log n)$	nlogn
$O(n^2)$	bình phương
$O(n^3)$	lập phương
$O(2^n)$	mũ


n!
4096
2048
1024
512
256
128
64
32
16
8
4
2
1
2^n
n^2
n log n
n
log n
1
2
3
4
5
6
7
8

Các kĩ thuật đánh giá thời gian chạy

- Không đệ quy so với đệ quy
- Luật tổng
- Thời gian chạy của các lệnh
  - gán
  - lựa chọn
  - lặp

Thời gian chạy của các lệnh

- Lệnh gán

  X = <biểu thức>

  Thời gian chạy của lệnh gán bằng thời gian thực hiện biểu thức
- Lệnh lựa chọn

  if (điều kiện) $\rightarrow T_0(n)$

  lệnh 1 $\rightarrow T_1(n)$

  else

  lệnh 2 $\rightarrow T_2(n)$

  Thời gian: $T_0(n) + \max(T_1(n), T_2(n))$

Thời gian chạy của các lệnh

- Lệnh lặp: for, while, do-while

  Ví dụ:

$$\sum_{i=1}^{X(n)} \left(T_0(n) + T_i(n)\right)$$

  X(n): Số vòng lặp

  T_0(n): Điều kiện lặp

  T_i(n): Thời gian thực hiện vòng lặp thứ i

Phân tích hàm đệ quy

- Định nghĩa đệ quy (quy nạp) có 2 phần
  - Phần cơ sở: định nghĩa một (số) phần tử đầu tiên trong chuỗi
  - Phần đệ quy (quy nạp)
- Ví dụ thời gian hàm tính giai thừa đệ quy
    - $T(1) = O(1)$
    - $T(n) = T(n-1) + O(1)$ với $n > 1$

Ví dụ 1

Thuật toán tạo ra ma trận đơn vị A cấp n.

```
(1) for (i = 0 ; i < n ; i++)
(2)         for (j = 0 ; j < n ; j++)
(3)                 A[i][j] = 0;
(4) for (i = 0 ; i < n ; i++)
(5)         A[i][i] = 1;
```

Độ phức tạp:

Ví dụ 1’

Thuật toán tạo ra ma trận đơn vị A cấp n.

```
(1) for (i = 0 ; i < n ; i++)
(2)         for (j = 0 ; j < n ; j++)
(3)                 if (i == j)
(4)                         A[i][j] = 1;
(5)                 else
(6)                         A[i][j] = 0;
```

Độ phức tạp:

Ví dụ 2

```
1) sum = 0;
2) for (i = 0; i < n; i++)
3)      for (j = i + 1; j <= n; j++)
4)              for (k = 1; k < 10; k++)
5)                       sum = sum + i * j * k ;
```

Độ phức tạp:

Ví dụ 2’

```
1) sum = 0;
2) for (i = 0; i < n; i++)
3)      for (j = i + 1; j <= n; j++)
4)           for (k = 1; k < 10; k++) {
5)                      x = 2 * y;
6)                      sum = sum + i * j * k ;
7)           }
```

Độ phức tạp:

Ví dụ 2”

```
1) for (i = 0; i < n; i ++)
2)  for (j = 0; j < m; j ++) {
3)          int x = 0;
4)          for (k = 0; k < n; k++)
5)                      x = x + k;
6)          for (k = 0; k < m; k++)
7)                      x = x + k;
8)          }
```

Độ phức tạp:

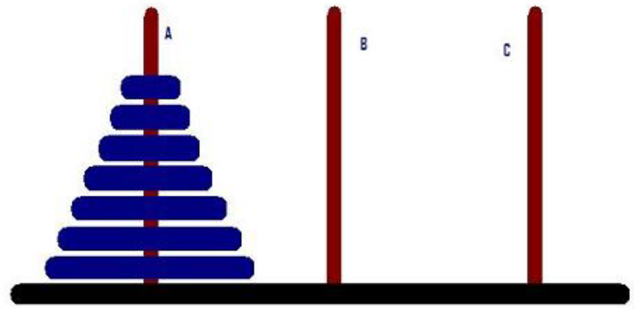
Bài tập

Giá trị trả về của hàm dưới đây biểu diễn gì? Đánh giá thời gian chạy.

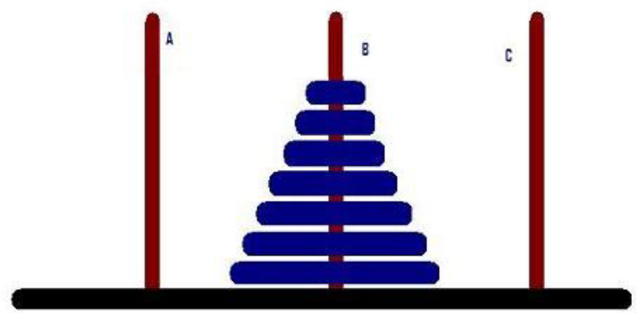
```
int algo1(int a[], unsigned int n)
{
        int sum = 0;
        int thisSum = 0;
        for(int i = 0; i < n; i++){
                thisSum += a[i];
                if(thisSum > sum)
                        sum = thisSum;
                if(thisSum < 0)
                        thisSum = 0;
        }
        return sum;
}
```

Bài tập

Đánh giá thời gian chạy của thuật toán đệ quy dưới đây cho bài toán tháp Hà Nội



// chuyển n đĩa ở A sang B
Algorithm move(n, A, B, C)
Input …
Output …
    if n = 1 **then**
        chuyển 1 đĩa ở A sang B
    else
        move(n – 1, A, C, B)
        chuyển 1 đĩa ở A sang B
        move(n – 1, C, B, A)

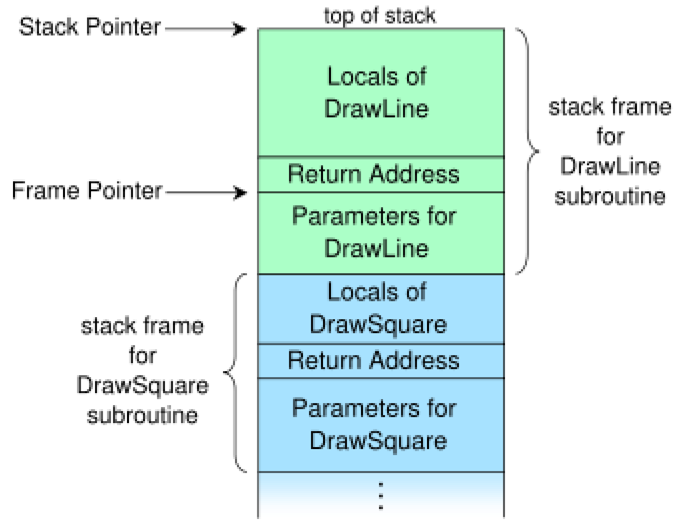


Thuật toán nào tốt hơn?

```
int factorial (int n) {
   if (n <= 1)    return 1;
   else   return n * factorial(n-1);
}

int factorial (int n) {
   if (n<=1)    return 1;
   else {
      fact = 1;
      for (k=2; k<=n; k++)
         fact *= k;
      return fact;
   }
}
```

Đệ quy hay lặp tốt hơn?



Bài tập

- Hãy đưa ra các thuật toán và phân tích độ phức tạp của từng thuật toán cho bài toán sau: Tìm dãy con liên tiếp có tổng lớn nhất của một dãy số nguyên a_1, a_2, …, a_n cho trước.

Chuẩn bị bài tới

- Đọc chương 1, chương 4 (4.1, 4.2)