ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Khái niệm cơ bản

Nội dung

- Định nghĩa và khái niệm cơ bản
- Mã giả
- Độ phức tạp tính toán
- Ký hiệu tiệm cận
- Ví dụ mở đầu

Định nghĩa và khái niệm

- Cấu trúc dữ liệu
  - Cách thức tổ chức dữ liệu trong bộ nhớ để truy cập và cập nhật thuận tiện
- Thuật toán
  - Dãy hữu hạn các bước tính toán để thu được đầu ra ứng với đầu vào
- Mục tiêu môn học
  - Trang bị kiến thức để thiết kế và cài đặt các cấu trúc dữ liệu và thuật toán hiệu quả để giải quyết các bài toán tính toán
- Ứng dụng
  - Hệ quản trị cơ sở dữ liệu
  - Tính toán tối ưu hóa
  - Trí tuệ nhân tạo, thị giác máy tính
  - Hệ điều hành
  - …

Mã giả

- Mô tả thuật toán đơn giản, gần gũi, ngắn gọn và không phụ thuộc vào cú pháp ngôn ngữ lập trình cụ thể

Assignment

```
x = <expression>;
x ← <expression>;
```

Condition

```
if a < b then {
   . . .
}
```

```
max(a[1..n]){
  ans = a[1];
  for i = 2 to n do
    if ans < a[i] then
      max = a[i];
  return ans;
}
```

For loop

```
for i = 1 to n do{
   . . .
}
```

Procedures, funtions

```
proc(a,b,x){
   . . .
   return ans;
}
```

While loop

```
while i ≠ 100 do{
   . . .
}
```

Mã giả

- Một bài toán (ví dụ sắp xếp) có thể có nhiều thuật toán giải quyết

```
selectionSort(a[1..n]){
   for k = 1 to n do{
    min = k;
    for j = k+1 to n do{
      if a[min] > a[j] then
        min = j;
    }
    swap(a[k],a[min]);
  }
}
```

```
insertionSort(a[1..n]){
   for k = 2 to n do{
     last = a[k];
     j = k;
   while(j > 1 and a[j-1] > last){
     a[j] = a[j-1];
     j--;
   }
   a[j] = last;
  }
}
```

Phân tích độ phức tạp thuật toán

- Phân tích độ phức tạp thuật toán
  - Thời gian
  - Bộ nhớ sử dụng
- Phân tích thời gian thực hiện
  - Thông qua thí nghiệm
  - Phân tích câu lệnh cơ bản

Phân tích độ phức tạp thuật toán

- Thực nghiệm
  - Viết chương trình bằng ngôn ngữ lập trình cụ thể
  - Chạy chương trình trên một máy tính với nhiều bộ dữ liệu đầu vào khác nhau
  - Vẽ biểu đồ thời gian thực hiện

- Hạn chế của phương pháp thực nghiệm
  - Cần lập trình bằng một ngôn ngữ lập trình cụ thể
  - Thời gian thực hiện phụ thuộc vào cấu hình máy tính

Phân tích độ phức tạp thuật toán

- Phân tính thời gian thực hiện bằng cách đếm số câu lệnh cơ bản (như một hàm của kích thước dữ liệu đầu vào)
- Xác định kích thước dữ liệu đầu vào
  - Số bít cần thiết để biểu diễn dữ liệu
  - Hoặc (ở mức cao hơn) là số phần tử của dãy số, số phần tử của ma trận, số đỉnh của đồ thị, …
- Xác định câu lệnh cơ bản

```
s = 0;
for i = 1 to n do
   s = s + a[i];
```

Câu lệnh cơ bản là câu lệnh gán → thời gian thực hiện là T(n) = n+1

Phân tích độ phức tạp thuật toán

```
1.  insertionSort(a[1..n]){
2.    for j = 2 to n do{
3.      key = a[j];
4.      i = j-1;
5.      while i > 0 and a[i] > key do{
6.         a[i+1] = a[i];
7.         i = i – 1;
8.      }
9.      a[i+1] = key;
10.   }
11. }
```

Ký hiệu t_j: số lần điều kiện của vòng lặp while (dòng 5) được thực hiện ứng với 1 giá trị j (vòng lặp bên ngoài)

Dòng	Thời gian	Số lần
2	c_2	n
3	c_3	n-1
4	C_4	n-1
5	C_5	$\sum_{j=2}^{n} t_j$
6	C_6	$\sum_{j=2}^{n} (tj - 1)$
7	C_7	$\sum_{j=2}^{n} (tj - 1)$
9	c_9	n-1

Thời gian thực hiện $T(n) = c_2n + c_3(n\text{-}1) + c_4(n\text{-}1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(tj - 1) + c_9(n\text{-}1)$

Phân tích độ phức tạp thuật toán

Thời gian tính $T(n) = c_2n + c_3(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(tj - 1) + c_9(n-1)$

- Tình huống tốt nhất: dãy đã được sắp xếp, $t_j = 1$ (j = 2,…,n)

→ $T(n)$ có dạng $an + b$ (tuyến tính)

- Tình huống tồi nhất: dãy được sắp xếp theo thứ tự ngược lại, $t_j = j$ (j = 2,…, n)

→ $T(n)$ có dạng $an^2 + bn + c$ (bình phương)

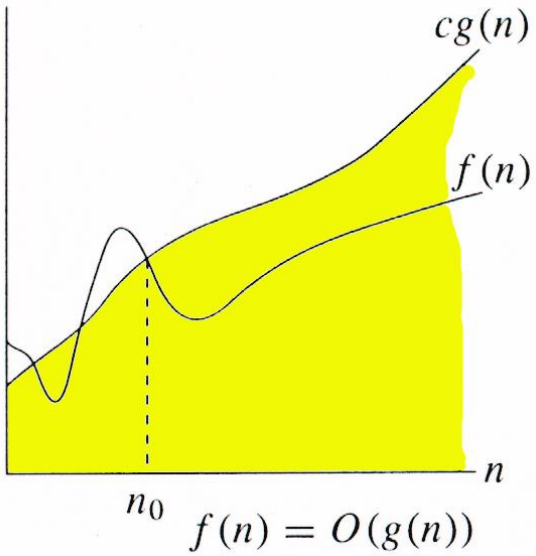
Phân tích độ phức tạp thuật toán

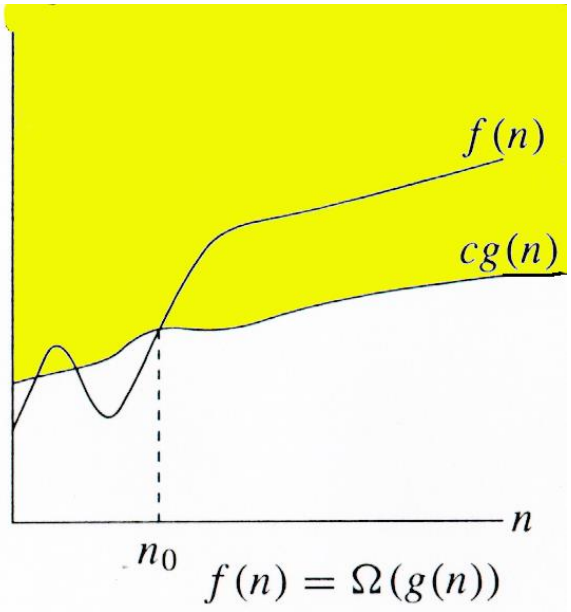
- **Độ tăng**: số hạng có số mũ cao nhất (ví dụ, n^2 trong $an^2 + bn + c$) sẽ đại diện cho độ tăng của hàm
- Độ tăng là chỉ số xác định tốc độ tăng của thời gian tính khi kích thước dữ liệu đầu vào tăng
- Một số độ tăng điển hình thường gặp

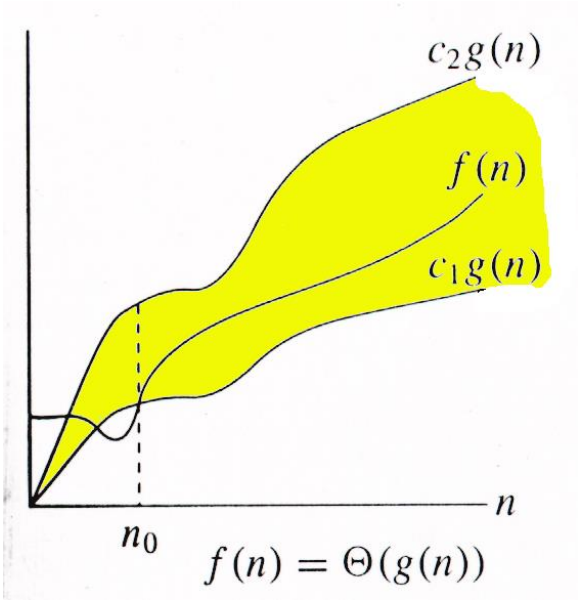
Logarithmic algorithms	Log n
Linear algorithms	n
Quadratic algorithms	n^2
Polynomial algorithms	n^k
Exponential algorithms	c^n

Ký hiệu tiệm cận Big O

- Giả sử g(n) là một hàm từ *N* đến *R*
  - $O(g(n)) = \{f(n) \mid \exists\, c > 0$ và n_0 sao cho $0 \le f(n) \le cg(n)\ \forall n \ge n_0\}$
  - $\Omega(g(n)) = \{f(n) \mid \exists\, c > 0$ và n_0 sao cho $0 \le cg(n) \le f(n)\ \forall n \ge n_0\}$
  - $\Theta(g(n)) = \{f(n) \mid \exists\, c_1, c_2 > 0$ và n_0 sao cho $c_1 g(n) \le f(n) \le c_2 g(n)\ \forall n \ge n_0\}$







Ký hiệu tiệm cận Big O

- Ví dụ
  - $10^3n^2 + 2n + 10^6 \in O(n^2)$
  - $10^3n^2 + 2n + 10^6 \in O(n^3)$
  - $10^3n^2 + 2n + 10^6 \in \Theta(n^2)$
  - $10^3n^2 + 2n + 10^6 \in \Omega(n)$
  - $10^3n^2 + 2n + 10^6 \in \Omega(n\log n)$

Ký hiệu tiệm cận Big O

- Giả sử *f* và *g* là các hàm không âm từ *N* đến *R*
  - Nếu $f(n) \in \Theta(g(n))$, thì $f(n) \in \Omega(g(n))$ và $f(n) \in O(g(n))$
  - Nếu $0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$, thì $f(n) \in \Theta(g(n))$
  - Nếu $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, thì $f(n) \in O(g(n))$

Ví dụ mở đầu

- Cho dãy $a = (a_1, a_2, \ldots, a_n)$. Một dãy con của a được định nghĩa là dãy gồm một số phần tử liên tiếp $a_i, a_{i+1},\ldots,a_j$. Trọng số của dãy con là tổng các phần tử của nó. Tìm dãy con có trọng số lớn nhất
- Ví dụ: a = 2, -10, **11, -4, 13**, -5, 2 khi đó dãy **11, -4, 13** là dãy con lớn nhất

Ví dụ mở đầu

```
maxSubSeq3(a[1..n]){
  ans = -∞;
  for i = 1 to n do{
    for j = i to n do{
       s = 0;
       for k = i to j do
          s = s + a[k];
       if s > ans then
         ans = s;
    }
  }
  return ans;
}
```

Thuật toán trực tiếp: thời gian $O(n^3)$

```
maxSubSeq2(a[1..n]){
  ans = -∞;
  for i = 1 to n do{
    s = 0;
    for j = i to n do{
       s = s + a[k];
       if s > ans then
         ans = s;
    }
  }
  return ans;
}
```

Cải tiến: Thời gian $O(n^2)$

Ví dụ mở đầu

- Quy hoạch động
  - Ký hiệu s[i]: trọng số của dãy con lớn nhất của dãy $a_1, \ldots, a_i$ trong đó phần tử cuối cùng là a_i.

- s[1] = a_1
- s[i] = $\begin{cases} s[i-1] + a_i, \text{ if } s[i-1] > 0 \\ a_i, \text{ ngược lại} \end{cases}$

```
maxSubSeq1(a[1..n]){
  s[1] = a[1];
  ans = s[1];
  for i = 2 to n do{
     if s[i-1] > 0 then
        s[i] = s[i-1] + a[i];
     else s[i] = a[i];
     if ans < s[i] then
       ans = s[i];
  }
  return ans;
}
```

Thuật toán tốt nhất: Thời gian O(n)

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

soict.hust.edu.vn/ fb.com/groups/soict