Thuật toán đánh giá và tiếp cận

Bài 1

Các vấn đề

- **Thuật toán**
  - **Khái niệm**
  - **Đặc trưng**
- **Độ phức tạp thuật toán**
  - **Cơ sở toán học**
  - **Tính toán độ phức tạp thuật toán**
- **Tiếp cận giải quyết bài toán**
  - **Các bước tiếp cận, giải quyết thuật toán**
  - **Xu hướng tiếp cận, giải quyết bài toán**

Thuật toán

- **Khái niệm thuật toán**

Định nghĩa:

Một thuật toán là một bản liệt kê các chỉ dẫn, các quy tắc cần thực hiện theo từng bước xác định nhằm giải quyết một bài toán đã cho trong một khoảng thời gian hữu hạn.

Thuật toán

- Ví dụ: 2.1 Mô tả thuật toán tìm số lớn nhất trong một dãy hữu hạn các số nguyên.
  1. Đặt giá trị cực đại tạm thời bằng số nguyên đầu tiên trong dãy;
  2. So sánh số nguyên tiếp theo với giá trị cực đại tạm thời, nếu lớn hơn giá trị cực đại tạm thời thì đặt giá trị cực đại tạm thời bằng số nguyên đó.
  3. Lặp lại bước 2) nếu còn các số nguyên trong dãy.
  4. Giá trị cực đại tạm thời ở thời điểm này chính là số nguyên lớn nhất trong dãy.

Thuật toán

Ta có thể viết lại thuật toán trên theo cách thức khác gọi là dạng giả mã:

Dữ liệu vào (input): a[1..n], a là mảng các số nguyên, *n*>0 là số các số trong mảng a;

Dữ liệu ra (output): max, số lớn nhất trong mảng a;

```
int TimMax(a: mảng các số nguyên);
max = a[1];
for  i:2 ->  n
      if (max < a[i] )
            max = a[i];
return max;
```

Thuật toán

- Như vậy, khi mô tả (hay xây dựng) một thuật toán cần chú ý tới các yếu tố sau:
- *Dữ liệu đầu vào*: Một thuật toán phải mô tả rõ các giá trị đầu vào từ một tập hợp các dữ liệu xác định. Ví dụ, dãy số nguyên a(1), a(2),...,a(n), với $n<\infty$; hai số nguyên dương a và b;...
- *Dữ liệu đầu ra*: Từ một tập các giá trị đầu vào, thuật toán sẽ tạo ra các giá trị đầu ra. Các giá trị đầu ra chính là nghiệm của bài toán. Ví dụ, số max là phần tử lớn nhất trong a(1),...,a(n); số *d* là ước chung lớn nhất của a và b;...

Thuật toán

1. *Tính xác định*: Các bước của thuật toán phải được xác định một cách chính xác, các chỉ dẫn phải rõ ràng, có thể thực hiện được.
2. *Tính hữu hạn*: Thuật toán phải kết thúc sau một số hữu hạn bước.
3. *Tính đúng đắn*: Thuật toán phải cho kết quả đúng theo yêu cầu của bài toán đặt ra.
4. *Tính tổng quát*: Thuật toán phải áp dụng được cho mọi bài toán cùng loại, với mọi dữ liệu đầu vào như đã được mô tả.

Thuật toán

Ta xét thuật toán nêu trong ví dụ trên:

Dữ liệu đầu vào: mảng các số nguyên;

Dữ liệu đầu ra: số nguyên lớn nhất của mảng đầu vào;

Tính xác định: Mỗi bước của thuật toán chỉ gồm các phép gán, mệnh đề kéo theo;

Tính hữu hạn: Thuật toán dừng sau khi tất cả các thành phần của mảng đã được kiểm tra;

Thuật toán

Tính đúng đắn: Sau mỗi bước kiểm tra và so sánh ta sẽ tìm được số lớn nhất trong các số đã được kiểm tra. Rõ ràng, sau lần kiểm tra cuối cùng thì xác định được số lớn nhất trong toàn bộ các số đã được kiểm tra, có nghĩa là toàn bộ dãy.

Tính tổng quát: Thuật toán cho phép tìm số lớn nhất của dãy số nguyên hữu hạn n bất kỳ.

Đánh giá độ phức tạp thuật toán

So sánh hai thuật toán nào tốt hơn?

Nhanh hơn? Ít tốn bộ nhớ hơn?

Dựa vào đâu để so sánh?

Đánh giá độ phức tạp thuật toán

- Độ lớn của dữ liệu đầu vào

-> Số lượng ô nhớ cần để giải quyết bài toán

-> Thời gian thực thi (số phép tính cơ bản) thực hiện

Đánh giá độ phức tạp thuật toán

Cho hai hàm số f và g, $f: R \to R$, $g: R \to R$.

Trong phần này bàn đến sự so sánh độ tăng của hai hàm f(x) và g(x) khi $x \to +\infty$.

1. Định nghĩa

Định nghĩa 1.1. Ta nói rằng f(x) = o(g(x)) khi x dần tới dương vô cùng, nếu như $\lim_{x\to+\infty} f(x)/g(x) = 0$.

Khi này người ta nói rằng f(x) tăng chậm hơn so với g(x) khi x lớn dần đến $+\infty$.

Ví dụ 1.1.

$x^2 = o(x^5)$

$\sin(x) = o(x)$

$1/x = o(1)$

•

Đánh giá độ phức tạp thuật toán

Định nghĩa 1.2. Ta nói rằng f(x) là O-lớn của g(x) khi x dần tới dương vô cùng.

Kí hiệu $f(x) = O(g(x))$

hoặc đôi khi viết f(x) là O(g(x))

nếu như tồn tại hai hằng số $C > 0$ và $N > 0$ sao cho với mọi $x > N$ thì $|f(x)| \leq C.|g(x)|$.

Ví dụ 1.2.

Xét hàm số $f(x) = x^2+2x+3$.

Rõ ràng $f(x) = O(x^2)$,

vì với mọi $x>1$ ta có $f(x) \leq x^2 + 2x^2 + 3x^2 = 6x^2$. Ngược lại ta cũng có $x^2 = O(f(x))$ vì hiển nhiên là với mọi $x>0$ ta có $x^2 < f(x)$.

Đánh giá độ phức tạp thuật toán

Ví dụ 1.3.

Ta cũng dễ thấy rằng $kx^2 = O(x^3)$ với $k>0$,

vì với $x \geq k$ ta có $kx^2 \leq 1.x^3$.

Để ý rằng cặp giá trị C và N, nếu tồn tại, rõ ràng không phải là duy nhất.

Ví dụ 1.4.

$1/(1+x^2) = O(1)$

$\sin(x) = O(1)$

Đánh giá độ phức tạp thuật toán

Định nghĩa 1.3. Ta nói rằng f(x) tương đương với g(x) khi x dần tới dương vô cùng,

kí hiệu $f(x) \approx g(x)$, nếu như $\lim_{x \to +\infty} f(x)/g(x) = 1$.

Ví dụ 1.5.

$$1+x+x^2 \approx x^2,$$

$$(2x+4)^2 \approx 4x^2.$$

Đánh giá độ phức tạp thuật toán

Mệnh đề 1.1.

Cho $f(x) = a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n$, *trong đó* a_i, $i=0,1,...n$, *là các số thực.*

Đánh giá độ phức tạp thuật toán

Mệnh đề 1.1.

Cho $f(x) = a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n$, *trong đó* a_i, $i=0,1,...n$, *là các số thực.*

Khi đó $f(x) = O(x^n)$.

Chứng minh:

Kí hiệu $C = |a_0| + |a_1| + |a_2| + ... + |a_{n-1}| + |a_n|$.

Với x>1 ta có $x^k < x^n$, với $k< n$, suy ra

$|f(x)| = |a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n|$

$\leq |a_0| + |a_1x^1| + |a_2x^2| + ... + |a_{n-1}x^{n-1}| + |a_nx^n| = |a_0| + |a_1|.x + |a_2|.x^2 + ... + |a_{n-1}|.x^{n-1} + |a_n|.x^n \leq (|a_0| + |a_1| + |a_2| + ... + |a_{n-1}| + |a_n|).X^n = C.x^n$. (đpcm)

Đánh giá độ phức tạp thuật toán

Ví dụ 1.6. Đánh giá tổng n số tự nhiên đầu tiên

S(n) = 1 + 2+.... + n

Đánh giá độ phức tạp thuật toán

Ví dụ 1.6. Đánh giá tổng n số tự nhiên đầu tiên

$S(n) = 1 + 2 + ... + n < n+n+ ...+ n = n^2$.

Vậy $S(n) = O(n^2)$.

Đánh giá độ phức tạp thuật toán

Ví dụ 1.6. Đánh giá tổng n số tự nhiên đầu tiên

$S(n) = 1 + 2 + ... + n < n+n+ ...+ n = n^2$.

Vậy $S(n) = O(n^2)$.

Nhận xét:

Số mũ 2 trong $O(n^2)$ đã phải là nhỏ nhất hay chưa?

Cũng như vậy, biểu thức n^2 đã phải là nhỏ nhất hay chưa?

Việc đánh giá hàm trong O-lớn cũng như bậc của hàm càng sát càng tốt.

Ta có nhận xét rằng nếu tồn tại các hằng số N, C_1 và C_2 sao cho bắt đầu từ $x>N$ ta có $C_1.g(x) \leq f(x) \leq C_2.g(x)$ thì rõ ràng là đánh giá $O(g(x))$ đối với $f(x)$ được coi là khá chính xác. Trong trường hợp này người ta còn nói rằng $f(x)$ và $g(x)$ là cùng bậc.

Đánh giá độ phức tạp thuật toán

- Chẳng hạn, f(x) = x^2 với g(x) = x^2+2x+3 là cùng bậc,

hoặc $f(x) = a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n$ với x^n là cùng bậc.

- Trong ví dụ 1.3 đã chỉ ra $kx^2 = O(x^3)$, nhưng rõ ràng x^3 không phải là $O(kx^2)$.

Thật vậy, với mọi C và N tuỳ ý ta chỉ cần chọn

$x > \max\{1, C.k, N\}$ khi đó $x^3 > C.kx^2$; có nghĩa là không tồn tại các số C và N như trong Định nghĩa 1.1. Như vậy, kx^2 và x^3 không phải là cùng bậc.

- Ví dụ 1.4: Đánh giá hàm giai thừa f(n) = n!.

Độ tăng của hàm

Quy ước:

0! = 1

1! =1;

3! = 1.2.3 = 6;

5! = 120;

10 ! = 362,880;

11! = 39,916,800;

20! = 2,432,902,008,176,640,000

Rõ ràng $n! < n^n$. Điều này chứng tỏ $n! = O(n^n)$.

Từ đó suy ra $\text{Log}(n!) = O(n \log n)$.

Đánh giá độ phức tạp thuật toán

Định nghĩa 1.4. Ta nói rằng $f(x) = \Omega(g(x))$ nếu như tồn tại $C>0$ và dãy $x_1, x_2, x_3, \ldots \to +\infty$, sao cho với mọi i: $|f(x_i)| > C.g(x_i)$.

Ví dụ, $x = \Omega(\log(x))$

Định nghĩa 1.5.

Ta nói rằng hàm f tăng theo hàm mũ nếu tồn tại $c>1$ và d sao cho $f(x) = \Omega(c^x)$ và $f(x) = O(d^x)$.

Ví dụ,

$f(x) = e^{2x}$

$f(n) = n!$

Độ tăng tổ hợp của hàm

Mệnh đề 1.2. Nếu $f(x) = O(u(x))$ và $g(x) = O(v(x))$ thì $(f+g)(x) = O(\max\{u(x),v(x)\})$.

Đánh giá độ phức tạp thuật toán

Mệnh đề 1.2. Nếu $f(x) = O(u(x))$ và $g(x) = O(v(x))$ thì $(f+g)(x) = O(\max\{u(x),v(x)\})$.

Chứng minh:

Từ giả thiết suy ra tồn tại C_1, k_1, C_2, k_2 sao cho với mọi $x > k_1$ thì $f(x) \leq C_1.u(x)$, với mọi $x > k_2$ thì $g(x) \leq C_2.v(x)$. Đặt $k = \max\{k_1, k_2\}$, và $C=\max\{C_1,C_2\}$. Rõ ràng là với mọi $x > k$ thì $f(x) \leq C.u(x)$ và $g(x) \leq C.v(x)$, hay $f(x)+g(x) \leq 2.C \max\{u(x), v(x)\}$. Suy ra $(f+g)(x) = O(\max\{u(x),v(x)\})$.

Hiển nhiên, nếu $u(x) = v(x)$, có nghĩa là nếu

$f(x) = O(u(x))$ và $g(x) = O(u(x))$,

thì ta có đánh giá $(f+g)(x) = O(u(x))$.

Đánh giá độ phức tạp thuật toán

Mệnh đề 1.3. Nếu $f(x) = O(u(x))$ và $g(x) = O(v(x))$ thì $(fg)(x) = O(u(x).v(x))$.

Chứng minh: (Tương tự chứng minh trên).

Đánh giá độ phức tạp thuật toán

Mệnh đề 1.3. Nếu $f(x) = O(u(x))$ và $g(x) = O(v(x))$ thì $(fg)(x) = O(u(x).v(x))$.

Chứng minh: (Tương tự chứng minh trên).

Nhận xét

Một số thuật toán bao gồm một số thủ tục con hợp lại. Với dữ liệu đầu vào xác định số các bước cần thiết để giải bài toán là tổng các bước theo các thủ tục con. Như thể ta phải tìm cách đánh giá số các bước mà các thủ tục con thực hiện sau đó tổ hợp các đánh giá đó lại.

Đánh giá độ phức tạp thuật toán

Ví dụ 1.5. Tìm đánh giá hàm

f(n)= n log (n!) + (3n^2 +2n)log n.

Đánh giá độ phức tạp thuật toán

Ví dụ 1.5. Tìm đánh giá hàm

$f(n) = n \log (n!) + (3n^2 + 2n)\log n$.

Theo các ví dụ đã nêu ở trên ta có

$\log(n!) = O(n \log n)$, từ đó dễ dàng suy ra rằng

$n \log (n!) = O(n^2 \log n)$;

Mặt khác ta có $3n^2 + 2n = O(n^2)$, hay suy ra

$(3n^2 + 2n) \log n = O(n^2 \log n)$.

Cuối cùng ta có: $f(n) = O(n^2 \log n)$

Đánh giá độ phức tạp thuật toán

Ví dụ 1.6. Tìm đánh giá đối với hàm

$$f(n) = (n+3)\log(n^2+4) + 5n^2.$$

Đánh giá độ phức tạp thuật toán

Ví dụ 1.6. Tìm đánh giá đối với hàm

$$f(n) = (n+3) \log (n^2+4) + 5n^2.$$

Ta có các đánh giá $n+3= O(n)$

và $\log (n^2+4)=O(\log n)$.

Thật vậy, đánh giá thứ nhất là hiển nhiên.

Xét đánh giá thứ hai. Rõ ràng là với $n>2$ ta có $\log(n^2+4) < \log(2n^2) < \log 2 + \log n^2 = \log 2 + 2\log n < 3 \log n$.

Từ đây suy ra $(n+3) \log (n^2+4) = O(n\log n)$.

Ngoài ra ta có $5n^2 = O(n^2)$. Từ đây suy ra

$f(n) = O(\max \{ n\log n, n^2 \}) = O(n^2)$.

Đánh giá độ phức tạp thuật toán

Ví dụ 1.7.

Tìm đánh giá tốt nhất của hàm $f(x) = 2^x + 23$.

Đánh giá độ phức tạp thuật toán

Ví dụ 1.7.

Tìm đánh giá tốt nhất của hàm $f(x) = 2^x + 23$.

Hiển nhiên là với mọi $x > 5$ ta có $f(x) < 2 \times 2^x$.

Vì vậy $f(x) = O(2^x)$.

Ta cũng dễ thấy được là $2^x < f(x)$ với mọi $x>0$.

Vậy $O(2^x)$ là đánh giá tốt nhất đối với f(x)

(hay nói cách khác 2^x là cùng bậc với f(x)).

Độ phức tạp thuật toán

- Tính hiệu quả của thuật toán thông thường được đo bởi thời gian tính (thời gian được sử dụng để tính bằng máy hoặc bằng phương pháp thủ công) khi các giá trị đầu vào có kích thước xác định. Tính hiệu quả của thuật toán cũng được xem xét theo thước đo dung lượng bộ nhớ đã sử dụng để tính toán khi kích thước đầu vào đã xác định.
- Hai thước đo đã nêu ở trên liên quan đến độ phức tạp tính toán của một thuật toán, được gọi là độ phức tạp thời gian và độ phức tạp không gian (còn gọi là độ phức tạp dung lượng nhớ).

Độ phức tạp thuật toán

- Trong phần đầu, chúng ta sẽ chỉ đề cập đến độ phức tạp thời gian của một thuật toán. Độ phức tạp thời gian của một thuật toán thường được biểu diễn thông qua số phép toán trong khi thực hiện thuật toán khi các giá trị dữ liệu đầu vào có **kích thước** xác định.
- Người ta thường dùng số phép tính làm thước đo độ phức tạp thời gian thay cho thời gian thực của máy tính là vì các máy tính khác nhau thực hiện các phép tính sơ cấp (so sánh, cộng, trừ, nhân, chia các số nguyên) trong những khoảng thời gian khác nhau.
-

Độ phức tạp thuật toán

- Chúng ta cũng sẽ không thực hiện việc phân rã các phép toán sơ cấp nêu trên thành các phép toán bit sơ cấp mà máy tính sử dụng vì việc này là khá phức tạp.
- Mặt khác cách đánh giá được sử dụng ở đây là cách đánh giá khả năng xấu nhất của thuật toán.

Độ phức tạp thuật toán

- Định nghĩa 1
- *Một thuật toán được gọi là có độ phức tạp đa thức, hay còn gọi là có thời gian đa thức, nếu số các phép tính cần thiết khi thực hiện thuật toán không vượt quá* $O(n^k)$, *với k nguyên dương nào đó, còn n là kích thước của dữ liệu đầu vào.*
- *Các thuật toán với* $O(k^n)$, *trong đó n là kích thước dữ liệu đầu vào, còn k là một số nguyên dương nào đó gọi là các thuật toán có độ phức tạp hàm mũ hoặc thời gian mũ.*

Độ phức tạp thuật toán

Ví dụ 2.4. Xét thuật toán tìm kiếm phần tử lớn nhất trong dãy số nguyên cho trước

Dữ liệu vào (input): a[1..n], a là mảng các số nguyên, $n>0$ là số các số trong mảng a;

Dữ liệu ra (output): max, số lớn nhất trong mảng a;

```
int TimMax(a: mảng các số nguyên);
max = a[1];
for  i:2 ->  n
      if (max < a[i] )
            max = a[i];
return max;
```

Độ phức tạp thuật toán

- Vì các phép toán được dùng ở đây là các phép toán so sánh sơ cấp như ta đã nêu ở trên nên ta sẽ dùng số các phép toán sơ cấp này để đo độ phức tạp của thuật toán. Ta dễ dàng thấy được số các phép toán so sánh sơ cấp được sử dụng ở đây là 2(n-1). Vì vậy ta nói rằng độ phức tạp của thuật toán nói trên là O(*n*) (còn nói là độ phức tạp tuyến tính).

Độ phức tạp thuật toán

Ví dụ 2.5. Xét độ phức tạp của thuật toán tìm kiếm tuyến tính. Mảng a[1..n], tìm vị trí xuất hiện phần tử x.

```
int TK_TT(a:mảng số nguyên; x: số nguyên)
        i       =1;
        While  (i<=n  and  x≠a[i])
                i= i+1;
        if   (i<=n ) index=i
        else   index=-1;
        return index;
```

Độ phức tạp thuật toán

- Trong mỗi bước của vòng lặp có 2 phép toán so sánh được thực hiện: một để xem đã tới cuối bảng hay chưa và một để so sánh số nguyên x với một phần tử trong bảng.
- Số bước của thuật toán trong trường hợp xấu nhất là n. Sau cùng là một phép so sánh chỉ số i sau cùng của vòng lặp với số n.
- Vậy tổng số phép so sánh được sử dụng là 2n+1.
- Độ phức tạp của thuật toán so sánh tuyến tính (tuần tự) là $O(n)$.

Độ phức tạp thuật toán

- Ví dụ 2.6. Xét độ phức tạp của thuật toán tìm kiếm nhị phân.
- Mảng a[1..n] đã được sắp xếp, vị trí xuất hiện x.

int TimKiem_NP(a:mảng số nguyên; x: số nguyên)

- 1. *first* =1; *last* =*n*;
- *2. found* =*false*;
- 3. While (*first<=last* and not *found*)
  - *index*= (*first* + *last*) div 2;
  - If (x = a(index)) *found = true*
  - else if (x< a(index)) *last = index* –1
    - else *first = index* +1;
- 4. If (not *found*) *index* :=-1;
- 5. return index;

Độ phức tạp thuật toán

- Ví dụ 2.6. Xét độ phức tạp của thuật toán tìm kiếm nhị phân.
- Giả thiết rằng có $n=2^k$ phần tử.
- Ta dễ dàng thấy được số phép toán so sánh tối đa là $2k+1 = 2\log_2 n$.
- Hay độ phức tạp O(logn), độ phức tạp logarit.

Độ phức tạp thuật toán

- Ví dụ 2.7. Độ phức tạp thuật toán tìm ước số chung lớn nhất hai số nguyên dương
- void swap(int a, int b)
  - Đổi chổ giá trị hai biến a, b
- int uscln(int a, int b)

```
1.      if(b>a) swap(a,b)
2.              if(a % b == 0)
3.                      return b;
4.              return uscln(b,a % b);
```

Độ phức tạp thuật toán

- Ví dụ 2.7. Xét độ phức tạp của thuật toán
  - Số lần thực hiện của một lần đề quy
  - Số lần đệ quy

Độ phức tạp thuật toán

- Ví dụ 2.8. Đếm số phần tử lớn nhất của mảng số nguyên a[0..n-1]
- int countmax(int a[])

1. max=a[0];
2. for(i=1;<n;i++)
    1. if (a[i]>max) max=a[i]
3. c=0;
4. for(i=0;i<n;i++)
    1. if(a[i]==max) c++;
5. return max;

- Đánh giá độ phức tạp thuật toán: ?

Độ phức tạp thuật toán

- Ví dụ 2.9. In ra các phần tử, sao cho phần tử trùng lặp chỉ in một lần từ mảng a[0..n-1]
- void printone(int a[])

1. for(i=0;<n;i++)
    1. f=0;
    2. for(j=i-1;j>0;j--)
        1. if(a[i]==a[j]) f=1
    3. if(f==0)
        1. printf("%d ",a[i]);

- Đánh giá độ phức tạp thuật toán: ?
- So sánh độ phức tạp thuật toán của ví dụ 2.8 và 2.9

Độ phức tạp thuật toán

Một vài loại thường gặp

O(1)	Độ phức tạp hằng số.
O(log*n*)	Độ phức tạp logarit.
O(*n*)	Độ phức tạp tuyến tính.
O(n^k)	Độ phức tạp đa thức.
O(*n*log*n*)	Độ phức tạp *n*log*n*.
O(b^n),b>1	Độ phức tạp hàm mũ
O(n!)	Độ phức tạp giai thừa

Độ phức tạp thuật toán

- Ví dụ 2.7. Cho một số nguyên n. Hãy tìm số nguyên m lớn nhất mà khi biểu diễn nó theo cơ số 16 thì có các chữ số khác nhau đôi một và tổng các chữ số (ở cơ số 16) đúng bằng n.
- N = 5 => m = $410_{(16)}$
- N = 10 => m = $43210_{(16)}$

Độ phức tạp thuật toán

- Do một số lớn nhất có thể thành lập từ các chữ số khác nhau trong hệ đếm 16 là FEDCBA9876543210 (= 18 364 758 544 493 064 720) cho nên số n có giá trị lớn hơn 120 thì không cần kiểm tra.
- Nếu sử dụng thuật toán tìm kiếm tuần tự thì ta sẽ phải duyệt 18 364 758 544 493 064 720 trường hợp. Mỗi trường hợp phải đổi số tương ứng ra cơ số 16, tính tổng các chữ số và so sánh với n. Và cuối cùng là phải tìm số lớn nhất thoả mãn cả hai điều kiện kia. Nếu giả định mỗi giây có thể kiểm tra được 1,000,000 trường hợp thì phải mất 5,101,321,817 giờ, hay 212,555,075 ngày, hay 582,343 năm.
- .

Độ phức tạp thuật toán

- Nếu dùng một thuật toán tốt ta sẽ cần ít thời gian hơn.
- Ta so sánh hai số có cùng số chữ số với nhau. Từ trái sang phải số nào có chữ số đầu tiên lớn hơn thì số đó lớn hơn.
- Như vậy, nếu với cùng các chữ số thì việc sắp đặt sao cho các chữ số giảm dần từ trái sang phải sẽ cho ta số lớn nhất trong các số có cùng các chữ số. Từ đó dẫn tới thuật toán sau:

  InPut: Số n và mảng A còn trống.

  OutPut: Mảng A chứa các chữ số của m

Độ phức tạp thuật toán

```
Procedure  Tim_so_m(n:byte);
    For i:=0 to 15 do a[i]:=0;
    i:=0;
    While n > a[i] do
            a) Inc(i);
            b) a[i]      :=  i;
            c) n         :=  n-i;
     j:=15;
    While n>0 do
            a)  t        := min {n,            j-a[i] };
                         b)  a[i]    := a[i] + t;
                         c)  n       := n - t;
                         d)  Dec(j);           Dec(i);
```

Độ phức tạp thuật toán

- Và theo thuật toán này ta chỉ cần tốn không đầy một giây là có kết quả.
- Độ phức tạp của thuật toán O(1);

Độ phức tạp thuật toán

Ví dụ 2.8. So sánh 2 thuật toán tính giá trị của đa thức tại $x=c$:

$f(x) = a_0 + a_1x^1 + a_2x^2 + ... + a_{n-1}x^{n-1} + a_nx^n$

$= a_nx^n + a_{n-1}x^{n-1} + ... + a_2x^2 + a_1x^1 + a_0$

=

$((..((a_nx + a_{n-1}).X + a_{n-2}).X + ... + a_2)x + a_1)x + a_0$

Độ phức tạp thuật toán

Thuật toán 1:

P:=1;

Ts:=a_0;

For i:=1 to n

    P := P * c ;

    Ts:= Ts + a(i)* P ;

End For

Độ phức tạp thuật toán

Thuật toán 1:

P:=1;

Ts:=a_0;

For i:=1 to n

P := P * c ;

Ts:= Ts + a(i)* P ;

End For

Ts chính là giá trị của đa thức tại c. Số phép toán $3n$. Để ý rằng với thuật toán này việc tính các x^k, với k = 1,2,...,n một cách riêng rẽ có thể sẽ tạo ra những số lớn, hoặc gặp phải những sai số lớn.

Độ phức tạp thuật toán

Thuật toán 2 (*Hoorner*):

```
Ts := a(n) ;
For  i:=1 to n
    Ts := Ts*c + a(n-i) ;
End For;
```

Ts là giá trị của đa thức tại c.

Độ phức tạp thuật toán

Thuật toán 2 (*Hoorner*):

```
Ts := a(n) ;
For  i:=1 to n
    Ts := Ts*c + a(n-i) ;
End For;
```

Ts là giá trị của đa thức tại c.

Số phép toán 2*n*.

Độ phức tạp thuật toán

Ví dụ 2.9. Đánh giá số phép chia số nguyên của thuật toán Euclid để tìm ước số chung lớn nhất của hai số nguyên a và b, a>b.

Độ phức tạp thuật toán

Ví dụ 2.9. Đánh giá số phép chia số nguyên của thuật toán Euclid để tìm ước số chung lớn nhất của hai số nguyên a và b, a>b.

```
        x:= a; y:=b;
While y>0
        r := x mod y;
        x := y;
        y := r;
End While
```

Độ phức tạp thuật toán

Ví dụ 2.9. Đánh giá số phép chia số nguyên của thuật toán Euclid để tìm ước số chung lớn nhất của hai số nguyên a và b, a>b.

```
        x:= a; y:=b;
While y>0
        r := x mod y;
        x := y;
        y := r;
End While
```

Đáp số: $O(\log_2 b)$.

Độ phức tạp thuật toán

Ví dụ 2.9. Đánh giá số phép chia số nguyên của thuật toán Euclid để tìm ước số chung lớn nhất của hai số nguyên a và b, a>b.

Định lý Lamé:

- Cho a và b là các số nguyên dương với a >= b. Số phép chia cần thiết để tìm USCLN(a,b) nhỏ hơn hoặc bằng 5 lần số chữ số của b trong hệ thập phân (hay nói cách khác thuộc $O(\log_2 b)$).

Độ phức tạp thuật toán

Khái niệm về độ phức tạp P, NP

Lớp P: P là lớp các bài toán được giải với thời gian đa thức.

Ví dụ: Tìm phần tử bé nhất trong dãy có n số nguyên cho trước.

Lớp NP: Có thể nói nôm na rằng NP là lớp các bài toán quyết định mà lời giải đối với dữ liệu vào không thể được kiểm tra nhanh chóng, chẳng hạn, với thời gian đa thức.

NP là lớp bài toán giải được bằng thuật toán bất định với thời gian đa thức (nondeterministic polynomial bouded).

Tiếp cận thuật toán

Bước 1: Xác định bài toán. Nhiệm vụ của giai đoạn này là làm rõ yêu cầu đặt ra của bài toán. Có thể đặt ra và trả lời một số câu hỏi: Bài toán đặt ra vấn đề gì phải giải quyết. Giải quyết những vấn đề này trong phạm vi nào? Xếp thứ tự về tầm quan trọng các yêu cầu đặt ra. Ví dụ, bài toán đặt ra là "Giải hệ phương trình đại số tuyến tính có số phương trình bằng số ẩn". Yêu cầu của bài toán này là "tìm nghiệm của hệ phương trình đại số tuyến tính n phương trình, n ẩn". Yêu cầu quan trọng nhất là "độ chính xác của nghiệm".

Tiếp cận thuật toán

- Bước 2: Phân tích bài toán và lựa chọn cách giải quyết. Với một bài toán đặt ra thường sẽ có nhiều cách giải quyết. Chúng có thể khác nhau về thời gian thực hiện, dung lượng bộ nhớ cần thiết, độ chính xác có thể đạt được. Tuỳ theo yêu cầu và điều kiện mà lựa chọn cách tiếp cận cho thích hợp. Ví dụ, với bài toán trên có thể lựa chọn chỉ giải quyết trường hợp hệ có một nghiệm duy nhất.

Tiếp cận thuật toán

- Bước 3: Xây dựng thuật toán (thuật giải). Trên cơ sở mô hình đã xây dựng được từ hai bước trước (chính là yêu cầu và các điều kiện) chi tiết hoá các bước thực hiện hoặc lựa chọn một trong các thuật toán đã biết phù hợp với bài toán.

Tiếp cận thuật toán

- Bước 4: Viết chương trình theo thuật toán (thuật giải) đã lựa chọn. Dựa vào thuật toán (thuật giải) đã được xây dựng, lựa chọn ngôn ngữ lập trình phù hợp với thuật toán và dữ liệu để thiết kế chương trình.

Tiếp cận thuật toán

- Bước 5: Xây dựng chương trình, thử nghiệm, triển khai

Tiếp cận thuật toán

- Ví dụ: tìm số có giá trị lớn thứ 2 trong mang a[0..n-1]. Ví dụ 1, 2, 5, 3, 7, 8, 3, 5, 8: thì số cần tìm là 7.
- Giải quyết:
  - Yêu cầu bài toán: Tìm số có giá trị lớn thứ 2. Đầu vào mảng có n phần tử. (Xác định chính xác tham số đầu ra)
  - Số có giá trị thứ 2:
    - Sắp xếp, sau đó tìm phần tử có giá trị lớn thứ 2
    - Dùng vòng lặp thứ nhất tìm phần tử lớn thứ nhất, vòng lặp thứ 2 tìm phần tử thứ 2
    - Dùng một vòng lặp, số lớn nhất sẽ là số đầu tiền, tìm số thứ khác số thứ nhất so sánh để lấy số thứ 2. duyệt đến cuối, nếu số lớn hơn số lớn nhất thì số lớn nhất hiện tại là số lớn thứ 2, ngược lại và lớn hơn số thứ 2 thì cập nhật số thứ 2.

Tiếp cận thuật toán

- Giải quyết (t):
  - int secondmax(int a[])
  - M1=a[0];
  - For(i=1;a[i]==m1 && i<n;i++)
  - If(i<n)
    - If(a[i]>m1)
      - M2=m1; m1=a[i]
    - Else
      - M2=a[i];
  - For(j=i+1;j<n;j++)
    - If(a[j]>m1
      - M2=m1; m1=a[j]
    - Else
      - If(a[j]>m2)
        - m[2=a[j]
  - Return m2

Tiếp cận thuật toán

- Trường hợp thông tin đầu ra với a[n]=1, 1,1,1,1
- => thông tin về số:
- => sinh viên thực hiện lại tham số đầu ra và viết lại hàm

Các vấn đề

- **Thuật toán**
  - **Khái niệm**
  - **Đặc trưng**
- **Độ phức tạp thuật toán**
  - **Cơ sở toán học**
  - **Tính toán độ phức tạp thuật toán**
- **Tiếp cận giải quyết bài toán**
  - **Các bước tiếp cận, giải quyết thuật toán**
  - **Xu hướng tiếp cận, giải quyết bài toán**

Bài tập

- Các phép tương quan ứng với các kí hiệu O, o, Ω, $\approx$ có tính bắc cầu hay không (ví dụ, f(x) = O(g(x)), g(x) = O(h(x)) thì suy ra f(x) = O(h(x)))?
- Chứng tỏ rằng nếu f(n) = $O(\log_a n)$, với a>1, thì f(n) = $O(\log_b n)$, với mọi b>1.
- Chứng minh rằng n logn = $O(n^2)$.
- Chứng minh rằng $\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$
- Tìm hàm f(x) sao cho f(x) = $O(x^{1+\varepsilon})$, với mọi $\varepsilon > 0$, nhưng f(x) $\neq O(x)$.

Bài tập

- Chứng minh rằng mệnh đề $f(n) = O((2+\varepsilon)^n)$ tương đương với mệnh đề $f(n) = o((2+\varepsilon)^n)$, với mọi $\varepsilon > 0$.
- Chứng tỏ rằng $T(n) = n^{o(1)}$ khi và chỉ khi tồn tại hằng số $k > 0$ sao cho $T(n) = O(n^k)$.
- Chứng minh rằng
- $\lg(n!) = O(n \lg n)$
- $n! = o(n^n)$.
- Chứng minh rằng, nếu $f(n)$ và $g(n)$ là các hàm đơn điệu tăng, thì $f(n) + g(n)$ và $f(g(n))$ cũng là các hàm đơn điệu tăng.
- Xây dựng các thuật toán tìm số bit 1 trong một số xâu bit s, số nguyên n. So sánh các thuật toán này.

Bài tập

- Mô tả thuật toán chèn một số nguyên vào một mảng đã được sắp xếp tăng dần.
- Mô tả thuật toán tìm các số cực đại trong một dãy hữu hạn các số thực.
- Mô tả thuật toán tìm từ dài nhất trong một xâu kí tự (ta hiểu từ là một xâu các chữ cái liên tiếp).
- Mô tả thuật toán tìm kiếm tam phân trên một mảng được sắp xếp tăng dần các số nguyên.
- Mô tả thuật toán tìm một dãy con liên tiếp không giảm từ một dãy số nguyên cho trước (ở đây ta hiểu dãy con liên tiếp là dãy con gồm các phần tử liên tiếp của dãy ban đầu).
- Xây dựng thuật toán nhân hai số nguyên, biết rằng mỗi số có thể có tới 100 chữ số.