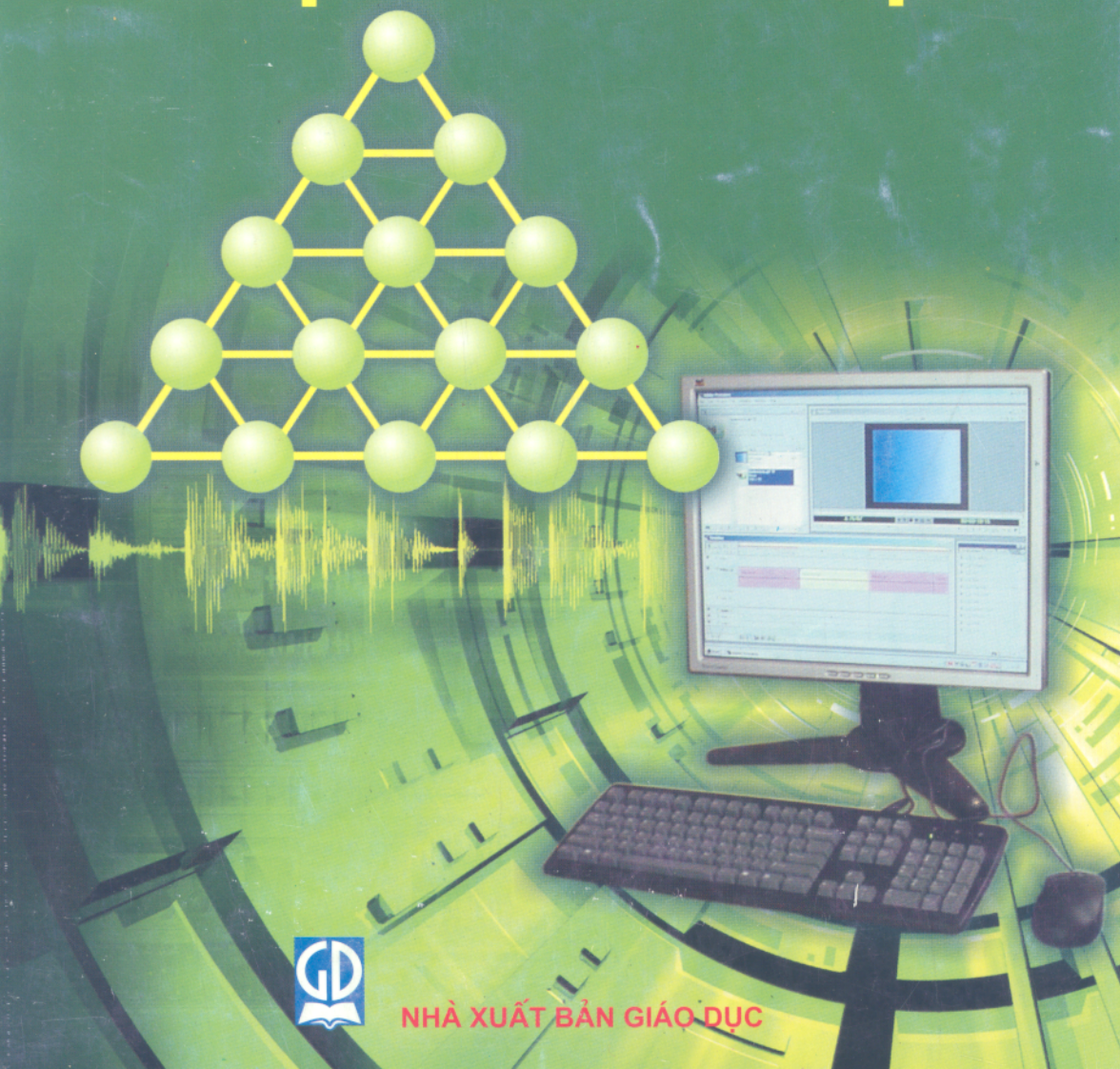
ĐỖ ĐỨC GIÁO

Toán rời rạc

ỨNG DỤNG TRONG TIN HỌC



NHÀ XUẤT BẢN GIÁO DỤC

PGS. TS. ĐỖ ĐỨC GIÁO

TOÁN RỜI RẠC
ỨNG DỤNG TRONG TIN HỌC

NHÀ XUẤT BẢN GIÁO DỤC

113–2008/CXB/59–175/GD

Mã số: 7B705Y8–DAI

Lời nói đầu

Để nâng cao chất lượng giảng dạy và học tập môn Toán rời rạc cho sinh viên và học viên ngành Công nghệ thông tin và các ngành khoa học tự nhiên, chúng tôi biên soạn cuốn ***Toán rời rạc ứng dụng trong tin học*** *gồm:*

– Phần I. Kiến thức bổ trợ;

– Phần II. Lôgic và ứng dụng;

– Phần III. Đồ thị và ứng dụng;

– Phần IV. Ngôn ngữ hình thức.

Cuốn ***Toán rời rạc ứng dụng trong tin học*** *trình bày các vấn đề toán học cơ bản nhất, nhưng lại hết sức thiết yếu và cần thiết đối với những ai muốn có được các kiến thức tin học vững chắc. Cuốn sách giúp người học hiểu được lý thuyết thấu đáo, rèn luyện tư duy khoa học, kỹ năng tính toán và khả năng vận dụng toán học vào giải quyết vấn đề, kích thích niềm say mê học tập và từ đó nâng cao kỹ năng thực hành, tư duy sáng tạo khi học các môn học cơ sở và chuyên ngành Công nghệ thông tin tiếp theo. Cuốn sách này cũng rất bổ ích cho việc ôn thi tuyển sinh sau đại học ngành Công nghệ thông tin được tổ chức hàng năm ở Đại học Quốc gia Hà Nội.*

Tác giả chân thành cảm ơn các bạn đồng nghiệp đã động viên tác giả biên soạn cuốn sách này.

Cuốn sách xuất bản lần đầu, nên khó tránh khỏi thiếu sót về hình thức cũng như nội dung. Vì vậy, tác giả mong nhận được sự góp ý của bạn đọc để cuốn sách ngày càng tốt hơn. Mọi góp ý xin gửi về: Công ty Cổ phần Sách Đại học và Dạy nghề: 25 Hàn Thuyên – Hà Nội.

TÁC GIẢ

MỤC LỤC

Phần II. LÔGIC VÀ ỨNG DỤNG

PHẦN I
KIẾN THỨC BỔ TRỢ

Chương 1
CÁC KHÁI NIỆM CƠ BẢN CỦA THUẬT TOÁN VÀ PHƯƠNG PHÁP ĐỆ QUY

§1. KHÁI NIỆM THUẬT TOÁN

1.1. Thuật toán là gì

Thuật toán là một khái niệm quan trọng của toán học. Nói đến thuật toán là nói đến một dãy các quy tắc, nhằm xác định một dãy các thao tác trên các đối tượng, sao cho sau một số hữu hạn bước thực hiện các thao tác, ta đạt được mục tiêu cần làm.

1.2. Các đặc trưng của thuật toán

– *Tính dừng:* Sau một số hữu hạn bước thuật toán phải dừng.

– *Tính xác định:* Ở mỗi bước, các thao tác phải rõ ràng, không gây nên sự nhập nhằng. Nói rõ hơn, trong cùng một điều kiện hai bộ xử lý cùng thực hiện một bước của thuật toán phải cho những kết quả như nhau.

– *Tính hiệu quả:* Trước hết thuật toán phải đúng đắn, nghĩa là sau khi đưa dữ liệu vào thuật toán hoạt động và đưa ra kết quả mong muốn.

– *Tính phổ dụng:* Thuật toán có thể giải bất kỳ một bài toán nào trong lớp các bài toán. Cụ thể là thuật toán có thể có các đầu vào là các bộ dữ liệu khác nhau trong miền xác định.

– *Yếu tố vào ra:* Đối với một thuật toán luôn có một đối tượng vào (input) và đối tượng ra (output).

1.3. Ngôn ngữ thuật toán

– Ngôn ngữ dùng để miêu tả thuật toán gọi là ngôn ngữ thuật toán.

– Thuật toán thường được mô tả bằng một dãy các lệnh. Bộ xử lý sẽ thực hiện các lệnh đó theo một trật tự nhất định cho đến khi gặp lệnh dừng thì kết thúc.

– Ngôn ngữ thuật toán bao gồm:

+ Ngôn ngữ liệt kê từng bước;

+ Sơ đồ khối;

+ Ngôn ngữ lập trình.

a) Ngôn ngữ liệt kê từng bước bao gồm:

– Thuật toán: Tên thuật toán và chức năng.

– Đầu vào: Các dữ liệu vào với tên, kiểu.

– Đầu ra: Các dữ liệu ra với tên, kiểu.

– Biến phụ (nếu có) gồm tên, kiểu.

– Hành động là các thao tác với các lệnh có nhãn là các số tự nhiên.

Ví dụ 1: Để giải phương trình bậc hai $ax^2 + bx + c = 0$ ($a \neq 0$), ta có thể mô tả thuật toán bằng ngôn ngữ liệt kê như sau:

Bước 1: Xác định các hệ số a, b, c.

Bước 2: Kiểm tra xem hệ số a có khác 0 hay không? Nếu a = 0 quay lại thực hiện bước 1.

Bước 3: Tính biểu thức $\Delta = b^2 - 4ac$.

Bước 4: Nếu $\Delta < 0$ thông báo "phương trình vô nghiệm" và chuyển đến bước 8.

Bước 5: Nếu $\Delta = 0$, tính $x_1 = x_2 = \frac{-b}{2a}$ và chuyển sang bước 7.

Bước 6: Tính $x_1 = \frac{-b - \sqrt{\Delta}}{2a}$, $x_2 = \frac{-b + \sqrt{\Delta}}{2a}$ và chuyển sang bước 7.

Bước 7: Thông báo các nghiệm x_1, x_2.

Bước 8: Kết thúc thuật toán.

b) Sơ đồ khối

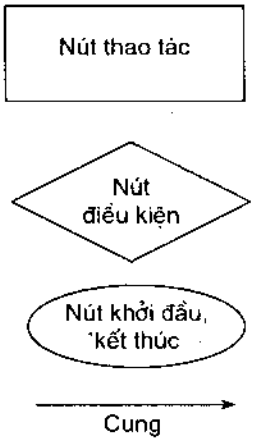
Để mô tả thuật toán bằng sơ đồ khối ta cần dựa vào các nút sau:

– *Nút thao tác:* Biểu diễn bằng hình chữ nhật, trong đó ghi câu lệnh cần thực hiện. Nếu có nhiều câu lệnh liên tiếp cần được thực hiện thì chúng có thể được viết chung trong nút thao tác;

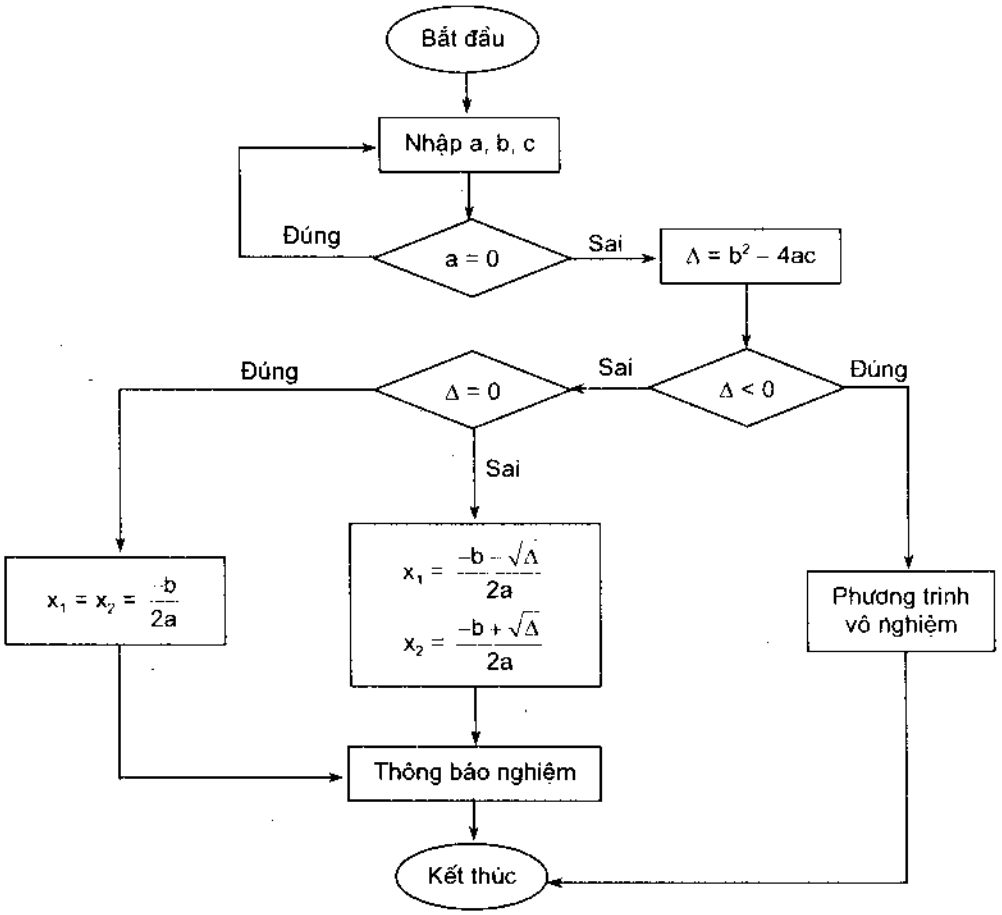
– *Nút điều kiện:* Biểu diễn bằng hình thoi, trong đó ghi điều kiện cần kiểm tra trong quá trình tính toán;

– *Nút khởi đầu, kết thúc:* Biểu diễn bằng hình elip, thể hiện sự bắt đầu hay kết thúc quá trình;

– *Cung:* Biểu diễn bằng đoạn thẳng có hướng, dùng để chỉ đường đi của thuật toán.



Chẳng hạn, giải phương trình bậc hai $ax^2 + bx + c = 0$ $(a \neq 0)$ ta mô tả thuật toán bằng phương pháp sơ đồ khối như sau:



c) Ngôn ngữ lập trình

Để giải bài toán bằng máy tính, người ta thường sử dụng một loại ngôn ngữ, gọi là *ngôn ngữ lập trình*, chẳng hạn như ngôn ngữ lập trình Cobol, Algol, Pascal, ... *Chương trình* chính là một dãy hữu hạn các câu lệnh được viết theo một quy tắc nhất định trên một ngôn ngữ lập trình nào đó. Hay nói cách khác; để giải một bài toán trước hết cần có một thuật toán để giải bài toán đó; để máy tính hiểu được thuật toán, người ta sử dụng một ngôn ngữ lập trình cụ thể để diễn đạt thuật toán thông qua ngôn ngữ đó.

Chẳng hạn, giải phương trình bậc hai $ax^2 + bx + c = 0$ $(a \neq 0)$ ta mô tả thuật toán bằng một chương trình Pascal như sau:

```
Program  GIAI_PHUONG_TRINH_BAC_HAI;
uses crt;
var   a, b, c, delta, x1, x2: real;
BEGIN
      clrscr;
      write('Nhap he so :');
      repeat
          write('a = '); readln(a);
          write('b = '); readln(b);
          write('c = '); readln(c);
      until a <> 0;
      delta := sqr(b) – 4*a*c;
      if delta < 0 then
          begin
              write('Phuong trinh vo nghiem');
              halt;
          end
      else
          if delta = 0 then
              begin
                  write('Phuong trinh co nghiem kep x = ', –b/(2*a));
                  exit;
              end
          else
              begin
                  x1:= (–b – sqrt(delta))/(2*a);
                  x2:= (–b + sqrt(delta))/(2*a);
                  writeln('Phuong trinh co hai nghiem phan biet');
                  write('x1 = ', x1, ' x2 = ', x2);
                  exit;
              end
      readln;
END.
```

1.4. Độ phức tạp thuật toán

Khi đề xuất một thuật toán, ngoài việc quan tâm đến tính đúng đắn, thường phải quan tâm đến một số vấn đề như: ưu điểm, nhược điểm, tính phổ dụng, thời gian tính toán, ... Với các thuật toán được sử dụng có tần số cao như các thuật toán sắp xếp, thuật toán tìm kiếm, ... ta đặc biệt quan tâm đến thời gian cần thiết cho việc thực hiện thuật toán đó.

Thông thường với mỗi thuật toán, dữ liệu vào sẽ có kích thước là một số nào đó. Chẳng hạn, khi sắp xếp một dãy số thì kích thước của dữ liệu có thể xem như là số phần tử n của dãy đó. Rõ ràng với n càng lớn thì thời gian cần thiết cho việc sắp xếp sẽ càng lớn và nó là một hàm của đối số n. Ta ký hiệu hàm đó là f(n). Trước khi đưa vào khái niệm độ phức tạp của thuật toán, ta đề cập tới một số khái niệm sau:

a) Khái niệm về độ tăng của hàm

Một trong những khái niệm thường dùng để phân tích độ tăng của một hàm là khái niệm "O" được định nghĩa như sau:

Cho f(x) và g(x) là hai hàm từ tập các số nguyên dương hoặc tập các số thực vào tập các số thực. Ta nói f(x) là O(g(x)) nếu tồn tại hai hằng số C và k sao cho

$$|f(x)| \le C|g(x)| \text{ với mọi } x > k.$$

Cần lưu ý rằng, cặp các hằng số C và k thoả mãn điều kiện trên là không duy nhất, đồng thời nếu f(x) là O(g(x)) mà h(x) là hàm thoả mãn $|g(x)| < |h(x)|$ với $x > k$ thì ta cũng có f(x) là O(h(x)).

Ví dụ 2: Cho $f(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_o$, với a_i là các số thực ($i = 0, 1, ..., n$). Khi đó $f(x) = O(x^n)$.

Thật vậy, với mọi $x > 1$:

$$\begin{aligned} |f(x)| &= |a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_o| \\ &\le |a_n|x^n + |a_{n-1}|x^{n-1} + ... + |a_1|x + |a_o| \\ &= x^n \left(|a_n| + \frac{|a_{n-1}|}{x} + ... + \frac{|a_1|}{x^{n-1}} + ... + \frac{|a_o|}{x^n} \right) \\ &\le x^n \left(|a_n| + |a_{n-1}| + ... + |a_1| + |a_o| \right). \end{aligned}$$

Đặt $C = |a_n| + |a_{n-1}| + ... + |a_1| + |a_o|$ và $k = 1$, ta có

$$|f(x)| \le Cx^n \text{ với mọi } x > k, \text{ hay } f(x) \text{ là } O(x^n).$$

Ví dụ 3: Cho $f(n) = 1 + 2 + 3 + ... + n$. Chỉ ra $f(n)$ là $O(n^2)$.

Thật vậy, $f(n) < n + n + n + ... + n = n.n = n^2$, nên $f(n)$ là $O(n^2)$ với $C = k = 1$.

Ví dụ 4: Chỉ ra hàm $n!$ là $O(n^n)$ và $\log n!$ là $O(n\log n)$.

Thật vậy, ta có $n! < n^n$ nên $n!$ là $O(n^n)$ với $C = k = 1$.

Từ $n! < n^n \Rightarrow \log n! < n\log n$, vậy $\log n!$ là $O(n\log n)$ với $C = k = 1$.

Ví dụ 5: Hãy chỉ ra hàm $\log n$ là $O(n)$.

Thật vậy, ta có $n < 2^n$ với mọi $n \geq 1$, nên $\log n < n$ (lôgarit cơ số 2) và vì vậy $\log n$ là $O(n)$ với $C = k = 1$.

b) Độ tăng của tổ hợp các hàm

Từ khái niệm về độ tăng của hàm ta có một số kết quả sau đây:

Ví dụ 6: Nếu $f_1(x)$ là $O(g_1(x))$ và $f_2(x)$ là $O(g_2(x))$ thì

$$(f_1(x) + f_2(x)) \text{ là } O(\max\{|g_1(x)|, |g_2(x)|\}).$$

Thật vậy, theo giả thiết tồn tại C_i và k_i sao cho

$$|f_i(x)| \leq C_i |g_i(x)| \text{ với mọi } x > k_i, (i = 1, 2).$$

Ta lại có
$$\begin{aligned} |f_1(x) + f_2(x)| &\leq |f_1(x)| + |f_2(x)| \\ &\leq C_1 |g_1(x)| + C_2 |g_2(x)| \\ &\leq C|g(x)| \text{ với mọi } x > k, \end{aligned}$$

ở đây $k = \max\{k_1, k_2\}$; $C = C_1 + C_2$; $g(x) = \max\{|g_1(x)|, |g_2(x)|\}$.

Ví dụ 7: Cho $f_1(x)$ và $f_2(x)$ đều là $O(g(x))$. Khi đó

$$f_1(x) + f_2(x) \text{ là } O(g(x)) \text{ (hệ quả của ví dụ 6).}$$

Ví dụ 8: Nếu $f_1(x)$ là $O(g_1(x))$ và $f_2(x)$ là $O(g_2(x))$ thì

$$f_1(x).f_2(x) \text{ là } O(g_1(x).g_2(x)).$$

Thật vậy, theo giả thiết $|f_1(x)| \leq C_1 |g_1(x)|$ với mọi $x > k_1$

và $|f_2(x)| \leq C_2 |g_2(x)|$ với mọi $x > k_2$.

Xét $|f_1(x).f_2(x)| = |f_1(x)|.|f_2(x)| \leq C_1C_2 |g_1(x)|.|g_2(x)|$

$$= C_1C_2 |g_1(x).g_2(x)|.$$

Vậy $f_1(x).f_2(x)$ là $O(g_1(x).g_2(x))$, với $C = C_1C_2$, $k = \max\{k_1, k_2\}$.

Ví dụ 9: Cho $f(n) \triangleq n\log(n!) + n^2\log n$, với n = 1, 2, 3, ... Khi đó f(n) là $O(n^2\log n)$.

Thật vậy, ta có nlog(n!) là $O(n^2\log n)$, còn $n^2\log n$ là $O(n^2\log n)$. Từ ví dụ 7 ta có f(n) là $O(n^2\log n)$.

Ví dụ 10: Chứng minh rằng, nếu f(x) là O(g(x)) thì $f^n(x)$ là $O(g^n(x))$, ở đây $f^n(x) = \underbrace{|f(x).f(x)...f(x)|}_{n \text{ lần}} = \underbrace{|f(x)|.|f(x)|...|f(x)|}_{n \text{ lần}} \le C^n |g^n(x)|$

với mọi x > k.

Hay $|f^n(x)| \le C|g^n(x)|$ với mọi x > k, $C := C^n$.

Vậy $f^n(x)$ là $O(g^n(x))$.

Ví dụ 11: Giả sử f(x) là O(g(x)) với f(x), g(x) là các hàm đơn điệu tăng và không giới nội. Chứng minh log(f(x)) là O(log(g(x))).

Thật vậy, theo giả thiết ta có $|f(x)| \le C_1|g(x)|$ với mọi $x > k_1$. Ta có thể giả thiết f(x) > 1, g(x) > 1 với x đủ lớn, hay $f(x) \le Cg(x)$ với x đủ lớn. Lôgarit cơ số 2 bất đẳng thức trên ta có

$$\log(f(x)) \le \log(C_1 g(x)) = \log C_1 + \log(g(x)) \le 2\log(g(x)) \text{ với x đủ lớn.}$$

Hay log(f(x)) là O(log(g(x)).

c) Độ phức tạp thuật toán

Ở đây chúng ta chỉ đề cập tới độ phức tạp của thuật toán về thời gian tính toán mà không đề cập tới độ phức tạp về không gian của thuật toán.

Các phép toán được dùng để đo độ phức tạp thời gian của thuật toán là phép so sánh số nguyên; các phép cộng, trừ, nhân, chia các số nguyên; hoặc bất kỳ một phép tính sơ cấp nào khác xuất hiện trong quá trình tính toán.

Cần lưu ý các thuật ngữ dùng trong độ phức tạp thuật toán thường gặp:

– Độ phức tạp O(1) gọi là *độ phức tạp hằng số.*

– Độ phức tạp O(logn) gọi là *độ phức tạp lôgarit.*

– Độ phức tạp O(n) gọi là *độ phức tạp tuyến tính.*

– Độ phức tạp O(nlogn) gọi là *độ phức tạp nlogn.*

– Độ phức tạp $O(n^b)$ gọi là *độ phức tạp đa thức.*

– Độ phức tạp $O(b^n)$ (b > 1) gọi là *độ phức tạp hàm mũ.*

– Độ phức tạp O(n!) gọi là *độ phức tạp giai thừa.*

Để minh hoạ về độ phức tạp của thuật toán ta xét một số ví dụ sau:

Ví dụ 12: Mô tả thuật toán tìm phần tử lớn nhất của dãy hữu hạn các số nguyên và tìm độ phức tạp của thuật toán đó.

Bước 1: Đặt giá trị cực đại tạm thời bằng số nguyên đầu tiên của dãy.

Bước 2: So sánh số nguyên tiếp theo với giá trị tạm thời, nếu nó lớn hơn giá trị cực đại tạm thời thì đặt cực đại tạm thời bằng số nguyên đó.

Bước 3: Lặp lại bước 2 (nếu còn các số nguyên trong dãy).

Bước 4: Dừng khi không còn số nguyên nào trong dãy. Khi đó cực đại tạm thời chính là số nguyên lớn nhất trong dãy.

Mô tả thuật toán tìm phần tử lớn nhất trong dãy hữu hạn:

```
Procedure MAX(a1, a2, ..., an : integer);
    max := a1;
    for i := 2 to n do
        if max < ai then max := ai
    {max là phần tử lớn nhất}
```

Lưu ý: Mỗi số hạng của dãy dùng hai phép so sánh, một để xác định chưa đạt đến cuối dãy, một để xác định có phải nó là giá trị lớn nhất tạm thời hay không. Việc so sánh này được dùng cho mỗi phần tử a_i trong dãy từ phần tử thứ hai trở đi (i = 2, 3, ..., n). Sau đó là phép so sánh để ra khỏi vòng lặp, nên số phép so sánh cần dùng tất cả là 2(n − 1) + 1 đối với thuật toán trên. Vậy thuật toán trên có độ phức tạp thời gian là O(n) (độ phức tạp tuyến tính).

Ví dụ 13: Mô tả thuật toán xác định vị trí của một phần tử trong một bảng liệt kê sắp thứ tự. Bài toán tìm kiếm được phát biểu như sau:

Xác định vị trí của x trong bảng liệt kê các phần tử phân biệt: $a_1, a_2, ..., a_n$.

Dưới đây ta đề cập tới thuật toán tìm kiếm tuyến tính:

Mô tả thuật toán: Trước hết ta so sánh x với a_1, nếu $x = a_1$ thì vị trí là 1, còn trường hợp $x \neq a_1$ ta so sánh tiếp x với a_2. Nếu $x = a_2$ thì vị trí của x là 2, còn trường hợp $x \neq a_2$ thì so sánh tiếp x với a_3. Quá trình trên được tiếp tục cho đến khi nếu $x = a_i$ ($i \leq n$) thì vị trí là i, trường hợp $x \neq a_i$ ($\forall i \leq n$) thì x không có mặt trong bảng liệt kê.

Mô tả thuật toán tìm kiếm tuyến tính:

```
Procedure TÌM_KIẾM_TT (x: integer, a1, a2; ..., an: các số nguyên phân biệt);
    i := 1
    while (i ≤ n) and (x ≠ ai) do
        i := i + 1;
    if i ≤ n then location := i
    else location := 0
    {location là chỉ số của số hạng bằng x (location = 0 nếu không tìm được x)}.
```

Lưu ý: Ở mỗi bước của vòng lặp trong thuật toán trên có hai phép so sánh: một để xem đã tới cuối dãy chưa; một để xem phần tử x có trùng với một phần tử của bảng liệt kê hay không. Cuối cùng là một phép so sánh ngoài vòng lặp.

Nếu $x = a_i$ thì có $2i + 1$ phép so sánh được sử dụng. Số phép so sánh lớn nhất là $2n + 1$ (khi $x \neq a_i$, $i = 1, 2, ..., n$). Sau đó cần một phép so sánh để thoát khỏi vòng lặp. Vậy, khi x không có mặt trong bảng thì tổng số phép so sánh trong thuật toán này là $2n + 2$. Vậy độ phức tạp của thuật toán là $O(n)$.

Xét bài toán trên trong trường hợp các phần tử trong bảng được sắp theo thứ tự tăng dần. Ví dụ, tìm số 19 trong bảng liệt kê theo thứ tự tăng dần: 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 13, 15, 16, 17, 18, 19, 20, 22. Tổng quát: Tìm x trong bảng liệt kê $a_1, a_2, ..., a_n$ với $a_1 < a_2 < ... < a_n$.

Để giải bài toán trên ta dùng thuật toán nhị phân, được mô tả như sau: So sánh x với số hạng a_m ở giữa dãy ($m = [(n + 1)/2]$, ở đây [x] là phần nguyên lớn nhất không vượt quá x). Nếu $x > a_m$ thì việc tìm kiếm x trên dãy gồm các số hạng $a_{m+1}, a_{m+2}, ..., a_n$; còn nếu $x \leq a_m$ thì việc tìm kiếm sẽ thực hiện trên dãy gồm các số $a_1, a_2, ..., a_m$. Cả hai trường hợp đều đưa đến bài toán tìm kiếm trên bảng không lớn hơn $[n/2]$ phần tử. Quá trình trên được thực hiện cho tới khi bảng liệt kê chỉ còn một phần tử và so sánh x với chính số hạng này.

Mô tả thuật toán tìm kiếm nhị phân:

```
Procedure TÌM_KIẾM_NHỊ_PHÂN (x : integer, a1, a2, ..., an: integer - tăng dần)
    i := 1 {i là điểm mút trái của khoảng tìm kiếm}
    j := n {j là điểm mút phải của khoảng tìm kiếm}
    while i < j
        begin
            m := [(i + j)/2]
            if x > am then i := m + 1
            else j := m
        end
```

```
if x = a_i then location := i
else location := 0
{location là chỉ số của số hạng bằng x (location = 0 nếu không tìm thấy x)}.
```

Lưu ý: Độ phức tạp của thuật toán tìm kiếm nhị phân trên là O(logn) vì nó phải dùng tới 2[logn] + 2 phép toán so sánh. Như vậy, trong trường hợp bảng liệt kê được sắp thứ tự thì thuật toán tìm kiếm nhị phân tốt hơn thuật toán tìm kiếm tuyến tính.

Để kết thúc phần này ta xét thêm thuật toán Euclid.

Trong lý thuyết số người ta đã chứng minh được rằng:

Nếu a = bq + r, trong đó a, b, q, r là các số nguyên thì:

$$\text{ƯCLN}(a, b) = \text{ƯCLN}(b, r) \qquad (*)$$

Giả sử a, b là các số nguyên dương, với a ≥ b. Đặt $r_o = a$, $r_1 = b$. Bằng cách áp dụng kết quả trên ta có:

$$r_o = r_1q_1 + r_2 \qquad (0 \le r_2 < r_1)$$
$$r_1 = r_2q_2 + r_3 \qquad (0 \le r_3 < r_2)$$
$$\ldots$$
$$r_{n-2} = r_{n-1}q_{n-1} + r_n \qquad (0 \le r_n < r_{n-1})$$
$$r_{n-1} = r_nq_n.$$

Số dư xuất hiện trong dãy các phép chia liên tiếp, vì dãy các số $a = r_o > r_1 > r_2 > \ldots \ge 0$ không thể chứa quá a số hạng. Từ (*) ta suy ra:

$$\text{ƯCLN}(a, b) = \text{ƯCLN}(r_o, r_1) = \text{ƯCLN}(r_1, r_2) = \ldots$$
$$= \text{ƯCLN}(r_{n-2}, r_{n-1}) = \text{ƯCLN}(r_{n-1}, r_n) = \text{ƯCLN}(r_n, 0) = r_n.$$

Vậy, ước chung lớn nhất là số dư khác 0 cuối cùng trong dãy các phép chia.

Ví dụ 14: Tìm ƯCLN(414, 662).

Áp dụng thuật toán Euclid ta được:

$$662 = 414.1 + 248$$
$$414 = 248.2 + 166$$
$$248 = 166.1 + 82$$
$$166 = 82.2 + 2$$
$$82 = 2.41$$

Do đó ƯCLN(414, 662) = 2 (2 là số dư cuối cùng khác không).

Thuật toán Euclid được mô tả như sau:

```
Procedure UCLN(a, b : integer);
    x := a
    y := b
    while y ≠ 0
        begin
            r := x mod y
            x := y
            y := r
    end {UCLN(a, b) là x}
```

Lưu ý: Giá trị ban đầu của x, y tương ứng là a và b. Ở mỗi giai đoạn của thủ tục x được thay bằng y và y được thay bởi x mod y (là số dư r trong phép chia x cho y). Quá trình này được lặp lại nếu $y \neq 0$ và thuật toán sẽ dừng khi $y = 0$. Giá trị của x ở thời điểm này là số dư khác 0 cuối cùng trong thủ tục và chính là ước số chung lớn nhất cần tìm.

§2. PHƯƠNG PHÁP ĐỆ QUY

Đệ quy là một khái niệm tồn tại trong cuộc sống, trong toán học, trong lập trình. Đệ quy cho một phương pháp ngắn gọn và sáng sủa để mô tả các đối tượng cũng như một số quá trình. Như vậy, đệ quy là một phương pháp xác định tập hợp các đối tượng thoả mãn một yêu cầu nào đó. Nó bao gồm các quy tắc, trong đó một số quy tắc dùng để xác định các đối tượng ban đầu, còn quy tắc khác dùng để xác định các đối tượng tiếp theo nhờ các đối tượng ban đầu đã được xác định.

Ví dụ 1: Một dãy số $a_o, a_1, a_2, ..., a_n, ...$ có tính chất: $a_o = 0$, $a_n = 2a_{n-1} + 3$ với mọi $n \geq 1$. Khi đó ta có thể xác định được số hạng a_n tổng quát của dãy số trên như sau:

Trước hết chọn các số α_1, α_2 sao cho

$$(a_n - \alpha_1) = \alpha_2(a_{n-1} - \alpha_1) \quad (1)$$

Từ (1) suy ra $a_n = \alpha_2 a_{n-1} - \alpha_1.\alpha_2 + \alpha_1$.

Rõ ràng các số α_1, α_2 phải thoả mãn hệ phương trình