Chương 1.

THUẬT TOÁN

§1. Định nghĩa.

Có thể định nghĩa thuật toán theo nhiều cách khác nhau. Ở đây chúng tôi không có ý định trình bày chặt chẽ về thuật toán như trong một giáo trình logic, mà sẽ hiểu khái niệm *thuật toán* theo một cách thông thường nhất.

Thuật toán là một qui tắc để, với những dữ liệu ban đầu đã cho, tìm được lời giải sau một khoảng thời gian hữu hạn.

Để minh hoạ cách ghi một thuật toán, cũng như tìm hiểu các yêu cầu đề ra cho thuật toán, ta xét trên các ví dụ cụ thể sau đây.

Cho n số $X[1], X[2],..., X[n]$, ta cần tìm m và j sao cho $m=X[j] = \max_{1\le k\le n} X[k]$, và j là lớn nhất có thể. Điều đó có nghĩa là cần tìm cực đại của các số đã cho, và chỉ số lớn nhất trong các số đạt cực đại.

Với mục tiêu tìm số cực đại với chỉ số lớn nhất, ta xuất phát từ giá trị $X[n]$. Bước thứ nhất, vì mới chỉ có một số, ta có thể tạm thời xem $m=X[n]$ và $j=n$. Tiếp theo , ta so sánh $X[n]$ với $X[n-1]$. Trong trường hợp n-1=0, tức n=1, thuật toán kết thúc.

Nếu $X[n-1] \le X[n]$, ta chuyển sang so sánh $X[n]$ với $X[n-2]$ .Trong trường hợp ngược lại, $X[n-1]$ chính là số cực đại trong hai số đã xét, và ta phải thay đổi m và j: đặt $m= X[n-1], j=n-1$. Với cách làm như trên, ở mỗi bước, ta luôn nhận được số cực đại trong những số đã xét. Bước tiếp theo là so sánh nó với những số đứng trước, hoặc kết thúc thuật toán trong trường hợp không còn số nào đứng trước nó.

Thuật toán mô tả trên đây được ghi lại như sau:

Thuật toán tìm cực đại.

M1. [Bước xuất phát] Đặt j←n, k←n-1, m← X[n].

M2. [Đã kiểm tra xong?] Nếu k=0, thuật toán kết thúc.

M3. [So sánh] Nếu X[k]≤m,chuyển sang M5.

M4. [Thay đổi m] Đặt j ← k, m ← X[k]. (Tạm thời m đang là cực đại)

M5. [Giảm k] Đặt k ← k-1, quay về M2.

Dấu "←" dùng để chỉ một phép toán quan trọng là phép thay chỗ (replacement).

Trên đây ta ghi một thuật toán bằng ngôn ngữ thông thường. Trong trường hợp thuật toán được viết bằng ngôn ngữ của máy tính, ta có một *chương trình*.

Trong thuật toán có những số liệu ban đầu, được cho trước khi thuật toán bắt đầu làm việc: các *đầu vào (*input). Trong thuật toán M, đầu vào là các số *X[1], X[2]*,..., *X[n]*.

Một thuật toán có thể có một hoặc nhiều *đầu ra* (ouput). Trong thuật toán M, các đầu ra là *m* và *j*.

Có thể thấy rằng thuật toán vừa mô tả thoả mãn các yêu cầu của một thuật toán nói chung, đó là:

1. *Tính hữu hạn*.Thuật toán cần phải kết thúc sau một số hữu hạn bước. Khi thuật toán ngừng làm việc, ta phải thu được câu trả lời cho vấn đề đặt ra. Thuật toán M rõ ràng thoả mãn điều kiện này, vì ở mỗi bước, ta luôn chuyển từ việc xét một số sang số đứng trước nó, và số các số là hữu hạn.

2. *Tính xác định*. Ở mỗi bước, thuật toán cần phải xác định, nghĩa là chỉ rõ việc cần làm. Nếu đối với người đọc, thuật toán M chưa thoả mãn điều kiện này thì đó là lỗi của người viết!

Ngoài những yếu tố kể trên, ta còn phải xét đến tính hiệu quả của thuật toán. Có rất nhiều thuật toán, về mặt lí thuyết là kết thúc sau hữu hạn bước, tuy nhiên thời gian "hữu hạn" đó vượt quá khả năng làm việc của chúng ta. Những thuật toán đó sẽ không được xét đến ở đây, vì chúng ta chỉ quan tâm những thuật toán có thể sử dụng thật sự trên máy tính.

Cũng do mục tiêu nói trên, ta còn phải chú ý đến *độ phức tạp* của các thuật toán. Độ phức tạp của một thuật toán có thể đo bằng *không gian*, tức là dung lượng bộ nhớ của máy tính cần thiết để thực hiện thuật toán, và bằng *thời gian*, tức là thời gian máy tính làm việc. Trong cuốn sách này, khi nói đến độ phức tạp của thuật toán, ta luôn hiểu là độ phức tạp thời gian.

§2. Độ phức tạp thuật toán.

Dĩ nhiên, thời gian làm việc của máy tính khi chạy một thuật toán nào đó không chỉ phụ thuộc vào thuật toán, mà còn phụ thuộc vào máy tính được sử dụng. Vì thế, để có một tiêu chuẩn chung, ta sẽ đo độ phức tạp của một thuật toán bằng số các phép tính phải làm khi thực hiện thuật toán. Khi tiến hành cùng một thuật toán, số các phép tính phải thực hiện còn phụ thuộc vào cỡ của bài toán, tức là độ lớn của đầu vào. Vì thế, độ phức tạp của thuật toán sẽ là một hàm số của độ lớn của đầu vào. Trong những ứng dụng thực tiễn, chúng ta không cần biết chính xác hàm này, mà chỉ cần biết " cỡ" của chúng, tức là cần có một ước lượng đủ tốt của chúng.

Khi làm việc, máy tính thường ghi các chữ số bằng những bóng đèn "sáng, tắt": bóng đèn sáng chỉ số 1, bóng đèn tắt chỉ số 0. Vì thế thuận tiện nhất là dùng hệ đếm cơ số 2, trong đó để biểu diễn một số, ta chỉ cần dùng hai kí hiệu 0 và 1. Một kí hiệu 0 hoặc 1 được gọi là một *bit* (viết tắt của chữ "binary digit"). Một số nguyên *n* biểu diễn bởi *k* chữ số 1 và 0 được gọi là một *số k-bit*. Trong chương tiếp theo, ta sẽ thấy rằng, số tự nhiên *n* sẽ là một số *k*-bit với $k=[\log_2 n]$ (dấu[ ] kí hiệu phần nguyên của một số).

Độ phức tạp của một thuật toán được đo bằng số các *phép tính bit*. Phép tính bit là một phép tính logic hay số học thực hiện trên các số 1-bit 0 và 1.

Để ước lượng độ phức tạp của thuật toán, ta dùng khái niệm *bậc O-lớn*.

Định nghĩa 1.1: Giả sử $f(n)$ và $g(n)$ là hai hàm xác định trên tập hợp các số nguyên dương. Ta nói *$f(n)$ có bậc O-lớn của $g(n)$,* và viết $f(n)=O(g(n))$ hoặc $f=O(g)$, nếu tồn tại một số $C>0$ sao cho với n đủ lớn, các hàm $f(n)$ và $g(n)$ đều dương, đồng thời $f(n) < Cg(n)$.

Ví dụ. 1) Giả sử $f(n)$ là đa thức;

$$f(n)=a_d n^d + a_{d-1}n^{d-1} + ...+a_1 n+a_0,$$

trong đó $a_d > 0$. Dễ chứng minh rằng $f(n)=O(n^d)$.

2) Nếu $f_1(n)=O(g(n)), f_2(n)=O(g(n))$ thì $f_1+f_2=O(g)$.

3) Nếu $f_1=O(g_1), f_2=O(g_2)$, thì $f_1.f_2=O(g_1.g_2)$.

4) Nếu tồn tại giới hạn hữu hạn

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

thì $f=O(g)$.

5)Với mọi số $\varepsilon>0$, $\log n=O(n^{\varepsilon})$.

Định nghĩa 1.2. Một thuật toán được gọi là có *độ phức tạp đa thức*, hoặc *có thời gian đa thức*, nếu số các phép tính cần thiết khi thực hiện thuật toán không vượt quá $O(\log^d n)$, trong đó n là độ lớn của đầu vào, và d là số nguyên dương nào đó.

Nói cách khác, nếu đầu vào là các số k-bit thì thời gian thực hiện thuật toán là $O(k^d)$, tức là tương đương với một đa thức của k.

Các thuật toán với thời gian $O(n^{\alpha})$, $\alpha>0$, được gọi là các thuật toán với *độ phức tạp mũ*, hoặc *thời gian mũ*.

Chú ý rằng, nếu một thuật toán nào đó có độ phức tạp $O(g)$, thì cũng có thể nói nó độ phức tạp $O(h)$ với mọi hàm $h > g$. Tuy nhiên, ta luôn luôn cố gắng tìm ước lượng tốt nhất có thể được để tránh hiểu sai về độ phức tạp thực sự của thuật toán.

Cũng có những thuật toán có độ phức tạp trung gian giữa đa thức và mũ. Ta thường gọi đó là thuật toán *dưới mũ*. Chẳng hạn, thuật toán nhanh nhất được biết hiện nay để phân tích một số nguyên n ra thừa số là thuật toán có độ phức tạp

$$\exp(\sqrt{\log n \log\log n}).$$

Khi giải một bài toán, không những ta chỉ cố gắng tìm ra một thuật toán nào đó, mà còn muốn tìm ra thuật toán " tốt nhất". Đánh giá độ phức tạp là một trong những cách để phân tích, so sánh và tìm ra thuật toán tối ưu. Tuy nhiên, độ phức tạp không phải là tiêu chuẩn duy nhất để đánh giá thuật toán. Có những thuật toán, về lí thuyết thì có độ phức tạp cao hơn một thuật toán khác, nhưng khi sử dụng lại có kết quả

(gần đúng) nhanh hơn nhiều. Điều này còn tuỳ thuộc những bài toán cụ thể, những mục tiêu cụ thể, và cả kinh nghiệm của người sử dụng.

Chúng ta cần lưu ý thêm một điểm sau đây. Mặc dù định nghĩa thuật toán mà chúng ta đưa ra chưa phải là chặt chẽ, nó vẫn quá "cứng nhắc " trong những ứng dụng thực tế! Bởi vậy, chúng ta còn cần đến các thuật toán "xác suất ", tức là các thuật toán phụ thuộc vào một hay nhiều tham số ngẫu nhiên. Những "thuật toán" này, về nguyên tắc không được gọi là thuật toán, vì chúng có thể, với xác suất rất bé, không bao giờ kết thúc. Tuy nhiên, thực nghiệm chỉ ra rằng, các thuật toán xác suất thường hữu hiệu hơn các thuật toán không xác suất. Thậm chí, trong rất nhiều trường hợp, chỉ có các thuật toán như thế là sử dụng được.

Khi làm việc với các thuật toán xác suất, ta thường hay phải sử dụng các số "ngẫu nhiên". Khái niệm chọn số ngẫu nhiên cũng cần được chính xác hoá. Thường thì người ta sử dụng một "máy" sản xuất số giả ngẫu nhiên nào đó. Tuy nhiên, trong cuốn sách này, chúng tôi không đề cập đến vấn đề nói trên, mà mỗi lần nói đến việc chọn số ngẫu nhiên, ta sẽ hiểu là điều đó thực hiện được trên máy.

Cũng cần lưu ý ngay rằng, đối với các thuật toán xác suất, không thể nói đến thời gian tuyệt đối, mà chỉ có thể nói đến thời gian hy vọng (expected).

Để hình dung được phần nào "độ phức tạp" của các thuật toán khi làm việc với những số lớn, ta xem bảng dưới đây cho khoảng thời gian cần thiết để phân tích một số nguyên n ra thừa số bằng thuật toán nhanh nhất được biết hiện nay (ta xem máy tính sử dụng vào việc này có tốc độ 1 triệu phép tính trong 1 giây)

Số chữ số thập phân	Số phép tính bit	Thời gian
50	$1,4.10^{10}$	3,9 giờ
75	$9,0.10^{12}$	104 ngày
100	$2,3.10^{15}$	74 năm
200	$1,2.10^{23}$	$3,8.10^{9}$ năm
300	$1,5.10^{29}$	$4,9.10^{15}$ năm
500	$1,3.10^{39}$	$4,2.10^{25}$ năm

Từ bảng trên đây, ta thấy rằng, ngay với một thuật toán dưới mũ, thời gian làm việc với các số nguyên lớn là quá lâu. Vì thế nói chung người ta luôn cố gắng tìm những thuật toán đa thức.

Lí thuyết về độ phức tạp thuật toán là một lí thuyết rất phong phú. Trong cuốn sách này, chúng tôi không lấy mục tiêu trình bày lí thuyết đó làm trọng tâm. Độc giả quan tâm đến lí thuyết thuật toán có thể tìm đọc các sách trong phần Tài liệu tham khảo.

Chương 2.

SỐ NGUYÊN

§1. Biểu diễn số nguyên và các phép tính số học

1.1 Hệ cơ số.

Mặc dù hầu hầu hết độc giả đã quen thuộc với cách biểu diễn số nguyên trong cơ số tuỳ ý, chúng tôi nhắc lại sơ qua vấn đề đó ở phần này, để thuận tiện cho việc trình bày các thuật toán về số nguyên.

Định lí 2.1. *Giả sử b là một số nguyên lớn hơn 1. Khi đó mọi số nguyên n có thể viết duy nhất dưới dạng*

$$n=a_k b^k + a_{k-1}b^{k-1} +...+ a_1 b^1 +a_0,$$

trong đó a_j *là số nguyên,* $0 \le a_j \le k-1$, *với* $j=0,1,...,k$ *và hệ số đầu tiên* $a_k \neq 0$.

Chứng minh. Ta chỉ cần thực hiện liên tiếp phép chia *n* cho *b*:

$$n=bq_0 +a_0,\ 0 \le a_0 \le b-1.$$

Nếu $q_0 > b$, ta tiếp tục chia q_0 cho *b* để được

$$q_0=bq_1 +a_1,\ 0 \le a_1 \le b-1.$$

Tiếp tục quá trình đó, ta có:

$$q_1=bq_2 +a_2,\ 0 \le a_2 \le b-1$$

$$q_2=bq_3 +a_3,\ 0 \le a_3 \le b-1$$

$$... ... ...$$

$$q_{k-1}=b.0 +a_k,\ 0 \le a_k \le b-1.$$

Quá trình kết thúc vì ta luôn có: $n > q_0 > q_1 > ... \ge 0$.

Chúng tôi dành cho độc giả việc chứng minh *n* có dạng như trong phát biểu của định lí, và biểu diễn đó là duy nhất.

Số *b* nói trong định lí được gọi là *cơ số* của biểu diễn. Các hệ biểu diễn cơ số 10 và 2 tương ứng được gọi là hệ thập phân và nhị phân. Các hệ số a_j được gọi là các chữ số. Về sau ta dùng bit để chỉ chữ số nhị phân.

Nếu số nguyên *n* biểu diễn trong cơ số *b* có *k* chữ số, thì từ chứng minh trên, ta có :

$$b^{k-1} \le n \le b^k.$$

Như vậy số chữ số của *n* được tính theo công thức:

$$k=[\log_b n]+1=[\log n / \log b]+1,$$

trong đó, kí hiệu log dùng để chỉ logarit cơ số *e*. Trong cơ số tuỳ ý, ta có: $k=O(\log n)$.

Để phân biệt các biểu diễn của số nguyên trong những hệ cơ số khác nhau, ta thường dùng cách viết $(a_k a_{k-1}...a_1 a_0)_b$ để chỉ số $n= a_k b^k + a_{k-1} b^{k-1} +...+ a_1 b^1 + a_0$,

Ví dụ. 1). Đối với số 1994 trong hệ thập phân, ta có $(1994)_{10}=(11111001010)_2$.

2). Trong máy tính, bên cạnh hệ cơ số 2, người ta cũng thường dùng hệ cơ số 8 hoặc 16. Lí do chủ yếu là vì việc chuyển một số viết ở cơ số này sang cơ số kia trong 3 cơ số đó được thực hiện một cách dễ dàng. Ví dụ, muốn chuyển một số cho trong cơ số 2 sang cơ số 8, ta chỉ việc nhóm từ phải sang trái từng khối 3 chữ số, rồi chuyển số được viết trong khối đó sang dạng thập phân. Chẳng hạn, số $(1110010100110)_2$ được tách thành các nhóm 1,110,010,100,110. Từ đó ta được:

$$(1110010100110)_2=(16246)_8.$$

Ta có thể làm tương tự để chuyển số đã cho thành số viết trong cơ số 16, chỉ cần nhóm thành từng bộ 4 chữ số. Chú ý rằng, trong trường hợp này, cần thêm vào các kí hiệu mới để chỉ các "chữ số " từ 10 đến 15.

Ta nhắc lại rằng máy tính sử dụng cách viết nhị phân, hoặc là các "bit". Máy tính nào cũng có giới hạn về độ lớn của các số có thể đưa vào tính toán. Giới hạn đó được gọi là *cỡ từ* của máy, kí hiệu qua *w*. Cỡ từ thường là một luỹ thừa của 2, chẳng hạn 2^{35}.

Để thực hiện các phép tính số học với những số nguyên lớn hơn cỡ từ, ta làm như sau. Muốn đưa một số $n > w$ vào máy, ta viết *n* dưới dạng cơ số *w*, và khi đó *n* được biểu diễn bằng những số không vượt quá cỡ từ. Ví dụ, nếu cỡ từ của máy là 2^{35}, thì để đưa vào một số có độ lớn cỡ 2^{350}-1, ta chỉ cần dùng 10 số nhỏ nhỏ hơn cỡ từ của máy, bằng cách biểu diễn n trong cơ số 2^{35}. Như đã nói trong ví dụ ở 1, việc chuyển một số từ cơ số 2 sang cơ số 2^{35} được thực hiện dễ dàng bằng cách nhóm từng khối 35 chữ số.

Từ qui tắc của các phép tính số học, ta thấy rằng:

1) Để cộng hoặc trừ hai số nguyên *k* bit, ta cần O(*k*) phép tính bit.

2) Để nhân hoặc chia hai số *k* bit theo qui tắc thông thường, ta cần $O(k^2)$ phép tính bit.

Trong những thập kỉ gần đây, người ta tìm ra những thuật toán nhân với độ phức tạp bé hơn nhiều so với cách nhân thông thường. Điều thú vị là, nếu thoạt nhìn thì các thuật toán đó "phức tạp" hơn quy tắc nhân thông thường. Tuy nhiên, khi làm việc với những số rất lớn, các thuật toán này cho phép thực hiện việc nhân hai số với một thời gian bé hơn hẳn so với quy tắc thông thường.

1.2 Thuật toán nhân nhanh hai số.

Ta sử dụng tính chất hết sức đơn giản của phép nhân: nếu $a=a_1+a_2$, $b=b_1+b_2$, thì $ab=a_1b_1+a_2b_2+a_2b_1+a_1b_2$. Điều đáng chú ý ở đây là, thay cho việc nhân hai số nguyên *n* bit, ta thực hiện việc nhân các số có chữ số nhỏ hơn, cùng với một số phép

cộng (đòi hỏi số phép tính bit ít hơn là phép nhân). Thực ra điều này không có gì mới: ngay trong quan niệm ban đầu, phép nhân a với b đã là phép cộng b lần số a!.

Tuy nhiên để có một thuật toán nhân nhanh, ta không thể cộng b lần số a, mà phải tìm được một cách tối ưu nào đó để tách b và a thành những phần nhỏ hơn. Những thuật toán trình bày dưới đây cho chúng ta một số cách để làm việc phân chia như vậy.

Giả sử muốn nhân hai số nguyên $2n$ bit,

$$a=(a_{2n-1}a_{2n-2}...a_1a_0)_2,$$

$$b=(b_{2n-1}b_{2n-2}...b_1b_0)_2.$$

Ta viết $a=2^nA_1+A_0$, $b=2^nB_1+B_0$, trong đó

$$A_1=(a_{2n-1}a_{2n-2}...a_n)_2,\ A_0=(a_{n-1}a_{n-2}...a_1a_0)_2,$$

$$B_1=(b_{2n-1}b_{2n-2}...b_n)_2,\ B_0=(b_{n-1}b_{n-2}...b_1b_0)_2.$$

Khi đó ta có:

$$ab=(2^{2n}+2^n)A_1B_1+2^n(A_1 - A_0)+(2^n+1)A_0B_0. \qquad (1.1)$$

Như vậy, việc nhân hai số a,b $2n$ bit được đưa về việc nhân các số n bit, cùng với các phép cộng, trừ và dịch chuyển (nhân một số với một luỹ thừa bậc n của 2 được thực hiện bằng cách dịch số đó sang trái n vị trí).

Định lí 2.2. *Thuật toán 2.1 có độ phức tạp là* $O(n^{\log_2 3})$.

Chứng minh. Gọi $M(n)$ là số các phép tính bit tối đa cần thiết khi thực hiện nhân hai số nguyên n bit bằng thuật toán 2.1. Từ công thức (1.1) ta có:

$$M(2n)\le 3M(n)+Cn,$$

trong đó C là một hằng số không phụ thuộc n. Đặt $c=\max(C,M(2))$.

Bằng quy nạp, dễ chứng minh được rằng

$$M(2^k) \le c(3^k-2^k).$$

Từ đó ta có

$$M(n)=M(2^{\log_2 n}) \le M(2^{[\log_2 n]+1}) \le c(3^{[\log_2 n]+1}-2^{[\log_2 n]+1}) \le 3c.3^{[\log_2 n]} \le 3c.3^{\log_2 n}=3cn^{\log_2 3}.$$

Định lí đã được chứng minh.

Với thuật toán 2.1, ta thấy rằng, ngay chỉ với cách phân chia đơn giản số nguyên thành hai phần với số chữ số bằng nhau, ta đã nhận được một thuật toán giảm đáng kể thời gian thực hiện phép nhân. Dĩ nhiên, cách phân chia như vậy còn xa với cách phân chia tối ưu.

Ta sẽ chứng tỏ rằng cách phân chia như trên có thể tổng quát hoá để nhận được những thuật toán nhân với độ phức tạp nhỏ hơn nhiều.

Cũng như trước đây, ta sẽ kí hiệu qua $M(n)$ số các phép tính bit cần thiết để thực hiện phép nhân hai số nguyên n bit. Trước tiên, ta chứng minh công thức sau: với mọi số tự nhiên n, tồn tại thuật toán sao cho:

$$M((r+1)n) \le (2r+1)M(n)+Cn, \tag{1.2}$$

với C là một hằng số nào đó. Như vậy, Định lí 2.2 là trường hợp riêng với r=1.

Giả sử cần nhân hai số $(r+1)n$ bit:

$$a=(a_{(r+1)n-1}...a_1a_0)_2,$$

$$b=(b_{(r+1)n-1}...b_1b_0)_2.$$

Ta tách mỗi số a,b thành $r+1$ số hạng:

$$a=A_r2^{rn}+...+A_12^n+A_0$$

$$b=B_r2^{rn}+...+B_12^n+B_0,$$

trong đó A_j,B_j là các số n bit.

Ta nhận xét rằng, việc biểu diễn một số nguyên dưới dạng cơ số nào đó cũng gần giống như viết số đó dưới dạng đa thức, trong đó các chữ số chính là các hệ số của đa thức. Vì vậy việc nhân hai số có thể thực hiện tương tự như việc nhân đa thức. Ta xét các đa thức sau:

$$A(x)=A_rx^r+...+A_1x+A_0,$$

$$B(x)=B_rx^r+...+B_1x+B_0,$$

$$W(x)=A(x)B(x)=W_{2r}x^{2r}+...+W_1x+W_0.$$

Từ định nghĩa các đa thức trên ta được: $a=A(2^n),b$=B(2^n), $ab= W(2^n)$. Như vậy, ta dễ dàng tính được tích ab nếu biết được các hệ số của đa thức $W(x)$.

Công thức (1.2) sẽ được chứng minh nếu ta tìm được một thuật toán tính các hệ số của $W(x)$ mà chỉ sử dụng $2r+1$ phép nhân các số n bit và một số phép tính khác với độ phức tạp O(n). Điều đó có thể làm bằng cách tính giá trị của đa thức $W(x)$ tại $2r+1$ điểm sau đây:

$$W(0)=A(0)B(0),\ W(1)=A(1)B(1),...,\ W(2r)=A(2r)B(2r).$$

Chú ý rằng, các số A_j,B_j không nhất thiết là các số n bit, nhưng với r cố định, chúng có số chữ số nhiều nhất là r+t, với một t cố định nào đó. Dễ thấy rằng, có thể nhân hai số $(r+t)$-bit với không quá $M(n)+c_1n$ phép tính bit, trong đó c_1 là hằng số (chỉ cần tách số $(n+t)$-bit thành hai phần n-bit và t-bit, và nhận xét rằng, khi t cố định, việc nhân số t-bit với số n-bit đòi hỏi không quá cn phép tính bit).

Khi đã có các giá trị $W(j),(j$=0,1,...$2r)$, ta tìm được đa thức $W(x)$ theo công thức Lagrange:

$$W(x)=\sum_{j=0}^{2r}(-1)^jW(j)\frac{x(x-1)...(x-j+1)(x-j-1)...(x-2r)}{j!(2r-j)!}\ .$$

Như vậy, các hệ số của $W(x)$ sẽ là tổ hợp tuyến tính (với hệ số không phụ thuộc n) của các giá trị $W(j)$, và do đó, tính được bằng $O(n)$ phép tính bit.

Ta đã chứng minh được công thức sau:

$$M((r+1)n) \le (2r+1)M(n)+Cn.$$

Lập luận tương tự như trong chứng minh định lí 2.1 ta có:

$$M(n) \le C_3 n^{\log_{r+1}(2r+1)} < C_3 n^{1+\log_{r+1}2}.$$

Với mọi $\varepsilon > 0$ bé tuỳ ý, khi các thừa số có số chữ số rất lớn, ta có thể chọn r đủ lớn sao cho $\log_{r+1}2 < \varepsilon$. Ta có định lí sau:

Định lí 2.3. *Với mọi $\varepsilon > 0$, tồn tại thuật toán nhân sao cho số phép tính bit $M(n)$ cần thiết để nhân hai số n bit thoả mãn bất đẳng thức*

$$M(n) < C(\varepsilon) n^{1+\varepsilon},$$

với hằng số $C(\varepsilon)$ nào đó độc lập với n.

Nhận xét. Có thể chứng minh được rằng, với cách chọn r "đủ tốt", ta có thuật toán nhân hai số n-bit sao cho

$$M(n) = O(n \log_2 n \log\log_2 n).$$

Chứng minh định lí đó không khó, nhưng khá dài (xem [Kr]).

§2. Số nguyên tố.

Định nghĩa 2.4. Số *nguyên tố* là số nguyên lớn hơn 1, không chia hết cho số nguyên dương nào ngoài 1 và chính nó. Số nguyên lớn hơn 1 không phải là số nguyên tố được gọi là *hợp số*.

Dễ chứng minh được rằng, số các số nguyên tố là vô hạn (Bài tập 2.14).

Như ta sẽ thấy trong những chương tiếp theo, bài toán xác định một số cho trước có phải là số nguyên tố hay không có nhiều ứng dụng trong thực tiễn. Đối với những số nhỏ, bài toán đó dĩ nhiên không có gì khó. Tuy nhiên, khi làm việc với những số lớn, ta cần phải tìm ra những thuật toán hữu hiệu, nghĩa là có thể thực hiện được trên máy tính trong một khoảng thời gian chấp nhận được. Khi nói đến "những số lớn", ta thường hiểu là những số nguyên dương có khoảng 100 chữ số thập phân trở lên.

Để có thể tìm ra những thuật toán xác định nhanh một số có phải là số nguyên tố hay không, ta cần hiểu sâu sắc tính chất các số nguyên tố. Trong chương này, ta chỉ đi vào các tính chất cơ bản nhất.

Định lí sau đây cho một thuật toán đơn giản để xác định các số nguyên tố.

Định lí 2.5. *Mọi hợp số n đều có ước nguyên tố nhỏ hơn $\sqrt{n}$.*

Thật vậy, vì n là một hợp số nên ta có thể viết $n=ab$, trong đó a và b là các số nguyên với $1<a\le b<n$. Rõ ràng ta phải có a hoặc b không vượt quá $\sqrt{n}$, giả sử đó là a. Ước nguyên tố của a cũng đồng thời là ước nguyên tố của n.

Từ định lí trên, ta có thuật toán sau đây để tìm ra các số nguyên tố nhỏ hơn hoặc bằng số n cho trước.

Sàng Eratosthenes. Trước tiên, ta viết dãy các số tự nhiên từ 1 đến n. Trong dãy đó gạch đi số 1, vì nó không phải là số nguyên tố. Số nguyên tố đầu tiên của dãy là 2. Tiếp theo đó ta gạch khỏi dãy số tất cả những số chia hết cho 2. Số đầu tiên không chia hết cho 2 là 3: đó chính là số nguyên tố. Ta lại gạch khỏi dãy còn lại những số nào chia hết cho 3. Tiếp tục như thế, ta gạch khỏi dãy những số chia hết cho mọi số nguyên tố bé hơn $\sqrt{n}$. Theo định lí trên, những số còn lại của dãy là tất cả các số nguyên tố không vượt quá n. Thật vậy, các hợp số không vượt quá n, theo định lí trên, đều phải có ước nguyên tố nhỏ hơn $\sqrt{n}$, và do đó đã bị gạch khỏi dãy số trong một bước nào đó của thuật toán.

Sàng Eratosthenes, mặc dù cho ta thuật toán xác định mọi số nguyên tố không vượt quá một số cho trước, rất ít được sử dụng để xác định xem một số đã cho có phải là số nguyên tố hay không. Nguyên nhân là vì thuật toán có độ phức tạp quá lớn: để kiểm tra n, ta phải thực hiện phép chia cho tất cả các số nguyên tố không vượt quá $\sqrt{n}$.

Ta hãy xét sơ qua về độ phức tạp của thuật toán nói trên. Với mỗi số thực dương x cho trước ta kí hiệu $\pi(x)$ số các số nguyên tố không vượt quá x. Khi đó, theo định lí Hadamard-Valée-Poussin ta có:

$$\lim_{x\to\infty} \pi(x) / \frac{x}{\log x} = 1.$$

Như vậy, số các số nguyên tố không vượt quá $\sqrt{n}$ là vào khoảng $\sqrt{n}/\log\sqrt{n} = 2\sqrt{n}/\log n$. Để chia n cho m, ta cần $O(\log_2 n.\ \log_2 m)$ phép tính bit. Như vậy, số các phép tính bit cần thiết để kiểm tra n có phải là số nguyên tố hay không ít nhất là $(2\sqrt{n}/\log n)(C\log_2 n) = C\sqrt{n}$ (ở đây ta dùng ước lượng rất sơ lược $\log_2 m \ge 1$). Như vậy, nếu n vào cỡ khoảng 100 chữ số thập phân, số các phép tính bit phải dùng sẽ vào cỡ 10^{50}. Với những máy tính thực hiện một triệu phép tính trong một giây, thời gian cần thiết sẽ vào khoảng $3{,}1.10^{36}$ năm!

Ta kết thúc tiết này bằng định lý quan trọng sau đây, thường được gọi là *định lý cơ bản của số học* .

Định lí 2.6. *Mọi số nguyên tố lớn hơn 1 đều phân tích được một cách duy nhất thành tích các số nguyên tố, trong đó các thừa số được viết với thứ tự không giảm.*

Chứng minh. Giả sử tồn tại những số không viết được thành tích các số nguyên tố. Gọi n là số bé nhất trong các số đó. Như vậy, n phải là hợp số, $n=a.b$, với $a, b<n$. Do định nghĩa của n các số a và b phân tích được thành tích các số nguyên tố, nghĩa là n cũng phân tích được. Mâu thuẫn với giả thiết.

Còn phải chứng minh phân tích là duy nhất. Giả sử ta có:

$$n=p_1 p_2 ...p_s=q_1q_2...q_r,$$

trong đó p_i, q_j là các số nguyên tố. Giản ước những số nguyên tố bằng nhau có mặt trong hai vế, ta được đẳng thức

$$p_{i1}p_{i2}...p_{iu}=q_{j1}q_{j2}...q_{jv},,$$

trong đó không có số nguyên tố nào có mặt cả hai vế. Như vậy, vế trái chia hết cho q_{j1}, và do đó phải tồn tại một thừa số của tích chia hết cho q_{j1}: điều đó vô lý, vì đây là tích các số nguyên tố khác với q_{j1}.

Phân tích như trên của các số nguyên được gọi là phân tích ra thừa số nguyên tố. Khi n là một số rất lớn, việc kiểm tra xem n là số nguyên tố hay hợp số, và nếu là hợp số thì tìm phân tích của nó ra thừa số nguyên tố, là một bài toán hết sức khó khăn. Trong những phần tiếp theo của cuốn sách, ta sẽ tìm hiểu nhiều thuật toán để làm việc đó, cũng như các ứng dụng của nó trong thực tiễn.

§3. Thuật toán Euclid.

Một trong những thuật toán cơ bản và lâu đời nhất của toán học là thuật toán Euclid. Thuật toán đó cho phép xác định ước chung lớn nhất của hai số nguyên cho trước.

Khi trình bày thuật toán Euclid, ta nhắc lại sơ qua khái niệm đồng dư. Những tính chất cần dùng của đồng dư và các tính chất cơ bản của ước chung lớn nhất được cho trong các bài tập của chương này.

Giả sử m là một số nguyên dương. Ta nói hai số nguyên a và b là đồng dư với nhau modulo m nếu m chia hết hiệu a-b (ta dùng cách viết $m \mid (a-b)$). Để chỉ quan hệ đồng dư, ta dùng ký hiệu $a \equiv b \pmod m$.

Như vậy, $a \equiv b \pmod m$ khi và chỉ khi tồn tại số nguyên k sao cho $a=b+km$.

Quan hệ đồng dư là một trong những quan hệ cơ bản của số học, và ta sẽ gặp thường xuyên trong những phần tiếp theo của cuốn sách. Trong thuật toán Euclid, ta chỉ dùng quan hệ đó để diễn đạt ngắn gọn về phần dư của phép chia.

Thuật toán sau đây cho phép tính ước chung lớn nhất (ƯCLN) d của hai số nguyên không âm a và b (ký hiệu là $d=(a,b)$).

Thuật toán Euclid

E1. [Kết thúc?] Nếu b=0, in ra a và kết thúc thuật toán.

E2. [Chia Euclid] Đặt r←a mod b, a←b, b←r và quay về bước 1.

Ví dụ: tính d=(24,63) bằng thuật toán Euclid.

Ta có: d=(24,63) = (15,24)=(9,15)=(6,9)=(3,6)=(0,3)=3.

Định lý sau đây vừa cho ta một chứng minh tính đúng đắn của thuật toán Euclid, vừa cho một ước lượng về độ phức tạp của nó.

Định lí Lamé. *Số phép chia cần thiết để tìm ƯCLN của hai số nguyên dương bằng thuật toán Euclid không vượt quá 5 lần số chữ số thập phân của số bé trong hai số đã cho.*

Chứng minh. Giả sử $a>b$ là hai số nguyên dương cho trước. Bằng thuật toán Euclid, ta có: $a=r_0$, $b=r_1$ và:

$$r_0=r_1q_1+r_2,\ 0\le r_2<r_1$$

$$r_1=r_2q_2+r_3,\ 0\le r_3<r_2$$

$$\dots\dots\dots\dots$$

$$r_{n-2}=r_{n-1}q_{n-1}+r_n,\ 0\le r_n<r_{n-1}$$

$$r_{n-1}=r_nq_n$$

Như vậy, ta đã làm n phép chia. Trong các phép chia đó, ta có: $q_1, q_2, \dots, q_{n-1} \ge 1$, $q_n \ge 2$, $r_n<r_{n-1}$. Từ đó suy ra:

$$r_n \ge 1=f_2,$$

$$r_{n-1} \ge 2r_n \ge 2f_2=f_3$$

$$r_{n-2} \ge r_{n-1}+ r_n \ge f_3+f_2=f_4$$

$$r_{n-3} \ge r_{n-2}+ r_{n-1} \ge f_4+f_5=f_6$$

$$\dots\dots\dots\dots$$

$$r_2 \ge r_3+ r_4 \ge f_{n-1}+f_{n-2}=f_n$$

$$b=r_1 \ge r_2+ r_3 \ge f_n+f_{n-1}=f_{n+1}$$

Chú ý rằng, dãy số $\{f_n\}$ nhận được chính là dãy số Fibonaci quen thuộc trong số học. Đối với dãy số này, bằng quy nạp, dễ chứng minh ước lượng sau đây:

$$f_n>\left(\frac{1+\sqrt{5}}{2}\right)^{n-2}.$$

Từ bất đẳng thức $b \ge f_{n+1}$ ta có:

$$\log_{10}b \ge (n-1)\log_{10}\left(\frac{1+\sqrt{5}}{2}\right)>(n-1)/5$$

Định lí được chứng minh.

Hệ quả 2.6. *Giả sử $a<b$, khi đó số các phép tính bit cần thiết để thực hiện thuật toán Euclid là $O((\log_2 a)^3)$.*

Thật vậy, số phép chia phải làm là $O(\log_2 a)$, và mỗi phép chia cần $O((\log_2 a)^2)$ phép tính bit.

Thuật toán Euclid, mặc dù đã ra đời hàng nghìn năm, vẫn là thuật toán tốt nhất để tìm ƯCLN của hai số nguyên cho trước! Cho đến năm 1967, J.Stein xây dựng được một thuật toán khá thuận tiện để tìm ƯCLN trong trường hợp các số đã cho được viết dưới dạng nhị phân. Ưu điểm chủ yếu của thuật toán này là ta không cần làm các phép tính chia (thực ra ta có làm phép chia số chẵn cho 2, nhưng trong cơ số 2 thì đó là phép dịch chuyển số đã cho sang phải một vị trí). Thuật toán dựa trên những nhận xét đơn giản sau (xem phần bài tập cuối chương):

1) Nếu a,b là các số chẵn, thì $(a,b)=2(a/2,b/2)$.

2) Nếu a chẵn, b lẻ, thì $(a,b)=(a/2,b)$.

3) Nếu a,b đều lẻ thì $a-b$ chẵn và $|a-b|<\max(a,b)$.

4) $(a,b)=(a-b,b)$.

Thuật toán đó được mô tả như sau (chúng tôi dành phần chứng minh cho độc giả).

Thuật toán tìm ƯCLN của hai số nguyên dương a,b.

E'1. (Tìm luỹ thừa của 2) Đặt $k\leftarrow 0$ và lập liên tiếp phép tính sau cho đến khi ít nhất một trong hai số a, b lẻ: đặt $k\leftarrow k+1$, $a\leftarrow a/2$, $b\leftarrow b/2$.

E'2. (Xuất phát). (ở bước xuất phát này, a, b đều đã được chia cho 2^k, và có ít nhất một trong hai số là lẻ). Nếu a lẻ, đặt $t\leftarrow -b$ và chuyển sang E'4. Nếu ngược lại, đặt $t\leftarrow a$.

E'3. (Chia đôi t). (Tại thời điểm này, t chẵn, khác 0). Đặt $t\leftarrow t/2$.

E'4. (t có chẵn hay không?) Nếu t chẵn quay về E'3.

E'5. (Sắp xếp lại max(a,b)). Nếu $t>0$, đặt $a\leftarrow t$; nếu ngược lại, đặt $b\leftarrow -t$. Như vậy, số lớn nhất trong hai số đã được thay bởi $|t|$.

E'6. (Trừ) Đặt $t\leftarrow a-b$. Nếu $t\neq 0$, quay lại E'3. Nếu ngược lại thuật toán kết thúc và in ra $a.2^k$.

Ngoài thuật toán Euclid nói trên, trong nhiều trường hợp, ta cần đến thuật toán Euclid mở rộng. Thuật toán này không những cho ta thuật toán tìm ƯCLN của hai số a, b, mà còn cho ta biểu diễn $d=(a,b)$ dưới dạng tổ hợp tuyến tính của a, b: $d=ma+nb$, trong đó m, n là các số nguyên.

Trước hết, ta chứng minh bổ đề sau:

Bổ đề 2.7: *ƯCLN của các số nguyên a và b là số d dương nhỏ nhất biểu diễn được dưới dạng tổ hợp tuyến tính của a và b.*

Thật vậy, giả sử d là số nguyên dương nhỏ nhất biểu diễn được dưới dạng $d=ma+nb$. Ta chứng tỏ d là ước chung của a và b. Xét phép chia $a=dq+r$, trong đó $0\le r<d$. Rõ ràng r cũng là một tổ hợp tuyến tính của a và b, nên do d là số nguyên dương nhỏ nhất có tính chất đó, $r=0$. Tương tự, d là ước của b. Dễ thấy rằng, mọi ước chung khác của a và b cũng là ước của d: vậy d chính là ước chung lớn nhất.

Khi cho hai số a, b, để tìm biểu diễn của d như trong bổ đề, ta thường là như sau: viết a=$bv+q$, $0\le q<b$. Sau đó,lại viết $b=uq+r=u(a-bv)+r$, $0\le r<q$. Tiếp tục quá trình đó, do các số dư q, r giảm dần nên ta thu được biểu diễn cần thiết. Điều vừa nói được thể hiện trong thuật toán sau đây, mà chứng minh chặt chẽ được dành cho độc giả.

Thuật toán Euclid mở rộng.

Cho hai số nguyên không âm u, v, tìm (u_1,u_2,u_3) sao cho $(u,v)=u_3=uu_1+vu_2$. Trong tính toán, ta thêm vào các ẩn phụ (v_1,v_2,v_3), (t_1,t_2,t_3) và luôn có trong mọi bước các đẳng thức sau đây:

$$ut_1+vt_2=t_3,\ uv_1+vv_2=v_3,\ uu_1+vu_2=u_3.$$

Ed1.(Xuất phát). Đặt $(u_1,u_2,u_3)\leftarrow(1,0,u)$, $(v_1,v_2,v_3)\leftarrow(0,1,v)$.

Ed2. (Kiểm tra $v_3=0$?) Nếu $v_3=0$, thuật toán kết thúc.

Ed3. (Chia, trừ). Đặt $q\leftarrow[u_3/v_3]$, và sau đó đặt $(t_1,t_2,t_3)\leftarrow(u_1,u_2,u_3)-q(v_1,v_2,v_3)$, $(v_1,v_2,v_3)\leftarrow(t_1,t_2,t_3)$ và quay về bước 2.

Ví dụ. Cho $a=63$, $b=24$. Dùng thuật toán Euclid ta có:

- Bước 1. $u_1=1$, $u_2=0$, $u_3=63$, $v_1=0$, $v_2=1$, $v_3=24$.

- Bước 2. $q=2$, $u_1=0$, $u_2=1$, $u_3=24$, $v_1=1$, $v_2=-2$, $v_3=15$.

- Bước 3. $q=1$, $u_1=1$, $u_2=-2$, $u_3=15$, $v_1=-1$, $v_2=3$, $v_3=9$.

- Bước 4. $q=1$, $u_1=-1$, $u_2=3$, $u_3=9$, $v_1=2$, $v_2=-5$, $v_3=6$.

- Bước 5. $q=1$, $u_1=2$, $u_2=-5$, $u_3=6$, $v_1=-3$, $v_2=8$, $v_3=3$.

- Bước 6. $q=2$, $u_1=-3$, $u_2=8$, $u_3=3$, $v_1=8$, $v_2=-21$, $v_3=0$.

Ta có biểu diễn: 3=(-3)64+8.24.

§4. Định lí Trung Quốc về phần dư:

Giả sử $m_1,m_2,\ldots,m_r$ là các số nguyên dương nguyên tố cùng nhau từng cặp. Khi đó hệ đồng dư:

$$x_1\equiv a_1(\mathrm{mod}\ m_1),$$

$$x_2\equiv a_2(\mathrm{mod}\ m_2),$$

... ...

$$x_r \equiv a_r (\mathrm{mod}\ m_r).$$

Có nghiệm duy nhất modulo $M=m_1m_2...m_r$.

Chứng minh. Trước hết ta xây dựng một nghiệm của hệ.

Giả sử $M_k=M/m_k= m_1m_2...m_{k-1}m_{k+1}...m_r$. Ta biết rằng $(M_k,m_k)=1$ vì $(m_j,m_k)=1$ với mọi $j \neq k$. Như vậy, theo bài tập 2.18 ta có thể tìm một nghịch đảo y_k của M_k modulo m_k, tức là $M_k y_k \equiv 1 \pmod{m_k}$.

Đặt

$$x=a_1M_1y_1+ a_2M_2y_2 +...+ a_rM_ry_r .$$

Ta thấy rằng $x \equiv a_k (\mathrm{mod}\ m_k)$ với mọi k vì $m_k | M_j$ với $j \neq k$ nên $M_j \equiv 0 \pmod{m_k}$ khi $j \neq k$. Như vậy, x chính là một nghiệm của hệ đang xét.

Ta chứng tỏ rằng nghiệm vừa xây dựng là duy nhất modulo M. Giả sử x_0, x_1 là hai nghiệm của hệ. Khi đó, với mỗi k, $x_0 \equiv x_1 \equiv a_k \pmod{m_k}$, cho nên $m_k | (x_0-x_1)$. Theo bài tập 2.17, $M | (x_0-x_1)$. Định lí được chứng minh.

Định lí Trung Quốc về phần dư liên quan bài toán nổi tiếng "Hàn Tín điểm binh". Tương truyền rằng, để kiểm tra quân số, Hàn Tín thường ra lệnh cho quân sĩ xếp thành hàng 3, hàng 5, hàng 7 và thông báo cho ông các số dư. Khi biết các số dư và đã có sẵn thông tin gần đúng về số quân của mình, Hàn Tín dùng định lí trên đây để suy ra số quân chính xác.

Định lí Trung Quốc về phần dư được sử dụng trong máy tính để làm việc với những số lớn. Để đưa một số nguyên lớn tuỳ ý vào máy tính và làm các phép tính số học với chúng, ta cần có những kĩ thuật đặc biệt. Theo định lí Trung quốc về phần dư, khi cho trước các modun nguyên tố cùng nhau $m_1,m_2,...,m_r$, một số dương $n<M= m_1m_2...m_r$ được xác định duy nhất bởi các thặng dư dương bé nhất của nó theo modulo m_j với $j=1,2,...,r$. Giả sử rằng cỡ từ của máy chỉ là 100, nhưng ta cần làm các phép tính số học với những số nguyên cỡ 10^6. Trước tiên ta tìm các số nguyên nhỏ hơn 100, nguyên tố cùng nhau từng cặp, sao cho tích của chúng vượt quá 10^6. Chẳng hạn, ta có thể lấy $m_1=99$, $m_2=98$, $m_3=97$, $m_4=95$. Ta chuyển các số nguyên bé hơn 10^6 thành những bộ 4 số theo thặng dư dương bé nhất modulo m_1,m_2,m_3,m_4 (để làm được điều này, ta cũng phải làm việc với những số nguyên lớn! Tuy nhiên điều đó chỉ cần làm một lần với input, và một lần nữa với ouput). Như vậy, chẳng hạn để cộng các số nguyên, ta chỉ cần cộng các thặng dư dương bé nhất của chúng modulo m_1,m_2,m_3,m_4. Sau đó lại dùng định lí Trung Quốc về phần dư để tìm bộ 4 số tương ứng với tổng.

Ví dụ. Ta muốn tính tổng $x=123684$, $y=413456$ với một máy tính cỡ từ là 100. Ta có:

$$x \equiv 33(\mathrm{mod}\ 99), 8(\mathrm{mod}\ 98), 9(\mathrm{mod}\ 97), 89(\mathrm{mod}\ 95)$$

$$y \equiv 32(\mathrm{mod}\ 99), 92(\mathrm{mod}\ 98), 42(\mathrm{mod}\ 97), 16(\mathrm{mod}\ 95)$$

Như vậy,

$$x+y \equiv 65(\mathrm{mod}\ 99), 2(\mathrm{mod}\ 98), 51(\mathrm{mod}\ 97), 10(\mathrm{mod}\ 95)$$

Bây giời ta dùng định lí Trung Quốc về phần dư để tìm $x+y$ modulo M=99.98.97.95=89403930. Ta có: $M_1=M/99=903070$, $M_2=M/98=912288$, M_3=M/97=921690, $M_4=M/95=941094$. Ta cần tìm ngược của $M_i(\text{mod } y_i)$ với i=1,2,3,4, tức là giải hệ phương trình đồng dư sau đây (Bằng thuật chia Euclid):

$$903070y_1 \equiv 91y_1 \equiv 1(\text{mod } 99)$$

$$912285y_2 \equiv 3y_2 \equiv 1(\text{mod } 98)$$

$$921690y_3 \equiv 93y_3 \equiv 1(\text{mod } 97)$$

Ta tìm được: $y_1 \equiv 37(\text{mod } 99)$, $y_2 \equiv 38(\text{mod } 98)$, $y_3 \equiv 24(\text{mod } 97)$, $y_4 \equiv 4(\text{mod } 95)$.

Như vậy,

$$x+y=65.903070.37+2.912285.33+51.921690.24+10.941094.4=3397886480$$

$$\equiv 537140(\text{mod } 89403930)$$

Vì $0<x+y<89403930$, ta suy ra $x+y$=537140.

Rất có thể độc giả cho rằng, cách cộng hai số sử dụng định lí Trung Quốc về phần dư quá phức tạp so với cách cộng thông thường. Tuy nhiên, cần chú ý rằng, trong ví dụ trên đây, ta làm việc với các số nhỏ. Khi các số cần cộng có độ lớn vượt xa cỡ từ của máy, các quy tắc cộng "thông thường" không còn áp dụng được nữa.

Nói chung cỡ từ của máy tính là luỹ thừa rất lớn của 2, chẳng hạn 2^{35}. Như vậy, để sử dụng định lí Trung Quốc về phần dư, ta cần các số nhỏ hơn 2^{35} nguyên tố cùng nhau từng cặp. Để tìm các số nguyên như vậy, thuận tiện nhất là dùng các số dạng 2^m-1, trong đó m là số nguyên dương. Các phép tính số học với những số có dạng như vậy tương đối đơn giản dựa vào bổ đề sau.

Bổ đề 2.8. *Nếu a và b là các số nguyên dương thì thặng dư dương bé nhất modulo 2^b-1 của 2^a-1 là 2^r-1, trong đó r là thặng dư dương bé nhất của a modulo b.*

Thật vậy, nếu $a=bq+r$, trong đó r là thặng dư dương bé nhất của a modulo b, thì ta có

$$(2^a-1)=(2^{bq+r}-1)=(2^b-1)(2b(q-1)+r+...+2^{b+r}+2^r)+(2^r-1).$$

Hệ quả 2.9. *Nếu a và b là các số nguyên dương, thì ước chung lớn nhất của 2^a-1 và 2^b-1 là $2^{(a,b)}$-1.*

Hệ quả 2.10. *Các số nguyên 2^a-1 và 2^b-1 nguyên tố cùng nhau khi và chỉ khi a và b nguyên tố cùng nhau.*

Chúng tôi dành việc chứng minh hai bổ đề này cho độc giả.

Ta có thể sử dụng hệ quả trên đây để tìm các số nhỏ hơn 2^{35}, nguyên tố cùng nhau từng cặp, sao cho tích của chúng lớn hơn một số đã cho. Giả sử ta cần làm các phép tính số học với những số nguyên có cỡ 2^{184}. Ta đặt: $m_1=2^{35}$-1, $m_2=2^{34}$-1, $m_3=2^{33}$-1, $m_4=2^{31}$-1, $m_5=2^{29}$-1, $m_6=2^{23}$-1. Vì số mũ của 2 trong các số trên nguyên tố với nhau từng cặp, nên theo hệ quả trên, các số đã chọn cũng nguyên tố với nhsu từng cặp. Ta có tích $m_1\ m_2\ m_3\ m_4\ m_5\ m_6>2^{184}$. Bây giờ ta có thể làm các phép tính số học với những số cỡ đến 2^{184}.

Trong các máy tính hiện đại, việc thực hiện nhiều phép tính được tiến hành đồng thời. Vì thế việc sử dụng định lí Trung Quốc về phần dư như trên lại càng tiện lợi: thay cho việc làm các phép tính với các số nguyên lớn, ta làm nhiều phép tính đồng thời với những số nguyên bé hơn. Điều đó giảm đáng kể thời gian tính toán.

Thuật toán giải phương trình đồng dư bằng định lí Trung Quốc

Từ chứng minh định lí Trung Quốc về phần dư, ta có thuật toán sau đây để giải hệ phương trình đồng dư $x \equiv x_i \pmod{m_i}$, trong đó m_i, $1 \le i \le k$ là các số nguyên tố với nhau từng cặp, x_i là các số nguyên cho trước. Trong thuật toán trình bày sau đây, chúng ta đã tìm ra cách để tránh phải làm việc với các số lớn như M_i và a_iM_i.

Thuật toán.

1. (Xuất phát). Đặt $j \leftarrow 2$, $C_1 \leftarrow 1$. Hơn nữa ta sắp xếp lại các số m_i theo thứ tự tăng dần.

2. (Tính toán sơ bộ). Đặt $p \leftarrow m_1m_2...m_{j-1} \pmod{m_j}$. Tính (u,v,d) sao cho $up+vm_j=d=UCLN(p,m_j)$ bằng thuật toán Euclid mở rộng.

Ed. Nếu d>0, in ra thông báo: các m_i không nguyên tố cùng nhau từng cặp. Nếu ngược lại, đặt $C_j \leftarrow u$, $j \leftarrow j+1$ và chuyển sang bước 3 nếu $j \le k$.

3. (Tính các hằng số phụ). Đặt $y_1 \leftarrow x_1 \bmod m_1$, và mỗi $j=2,...,k$ tính:

$y_j \leftarrow (x_j-(y_1+m_1(y_2+m_2(y_3+...+m_{j-2}y_{j-1})...))C_j \bmod m_j$.

4. (Kết thúc). In ra

$x \leftarrow y_1+m_1(y_2+m_2(y_3+...+m_{k-1}y_k)...)$, và kết thúc thuật toán.

§5. Một số đồng dư đặc biệt.

Định lí Wilson. *p là số nguyên tố khi và chỉ khi* $(p-1)! \equiv -1 \pmod{p}$.

Chứng minh. Trước tiên, giả sử *p* là số nguyên tố. Khi p=2, ta có $(p-1)! \equiv 1 \equiv -1 \pmod 2$. Bây giờ giả sử *p* là số nguyên tố lớn hơn 2. Theo bài tập 2.18, với mỗi số nguyên *a* với $1 \le a \le p-1$, tồn tại nghịch đảo $\overline{a}$, $1 \le \overline{a} \le p-1$, với $a\overline{a} \equiv 1 \pmod p$. Theo bài tập 2.13, trong số các số nguyên dương nhỏ hơn *p*, chỉ có 1 và *p*-1 là nghịch đảo với chính nó. Như vậy ta có thể nhóm các số nguyên từ 2 đến *p*-2 thành (*p*-3)/2 cặp số nguyên, tích của mỗi cặp đồng dư với 1 modulo *p*. Như vậy ta có:

$$2.3.....(p-3)(p-2) \equiv 1 \pmod p$$

Nhân hai vế với 1 và *p*-1 ta được:

$$(p-1)! \equiv 1.2.3...(p-2)(p-1) \equiv 1(p-1) \equiv -1 \pmod p$$

Ngược lại giả sử *p* thoả mãn đồng dư phát biểu trong định lí và *a* là một ước số của *p*, $a<p$. Khi đó, $a \mid (p-1)!$. Nhưng theo giả thiết, $p \mid (p-1)!+1$, từ đó suy ra $a=1$, vì là ước chung của *p* và (*p*-1)!. Vậy *p* là số nguyên tố, định lí được chứng minh.

Định lí Wilson có thể được dùng để kiểm tra một số có phải là số nguyên tố hay không. Tuy nhiên , dễ thấy rằng, thuật toán dựa theo định lí Wilson khó có thể sử dụng với những số nguyên lớn, bởi vì số các phép tính bit đòi hỏi quá cao.

Để đơn giản, ta gọi công việc xem xét một số đã cho có phải là số nguyên tố hay không là *kiểm tra nguyên tố*. Định lí sau đây có nhiều ứng dụng trong kiểm tra nguyên tố.

Định lí Fermat bé. *Nếu p là số nguyên tố và a là số không chia hết cho p thì* $a^{p-1} \equiv 1 (\mathrm{mod}\ p)$.

Chứng minh. Xét p-1 số nguyên a, $2a,..., (p-1)a$. Các số đó đều không chia hết cho p và không có hai số nào đồng dư modulo p. Như vậy, các thặng dư dương bé nhất của chúng phải là 1, 2,... p-1, xếp theo thứ tự nào đó. Từ đó ta có:

$$a.2a.....(p-1) \equiv 1...(p-1) \equiv (p-1)!(\mathrm{mod}\ p)$$

tức là

$$a^{p-1}(p-1)! \equiv 1(\mathrm{mod}\ p)$$

Vì $((p-1)!,p)=1$ nên ta có $a^{p-1} \equiv 1(\mathrm{mod}\ p)$.

Hệ quả 2.11. *Nếu p là số nguyên tố và a là số nguyên dương thì* $a^p \equiv a (\mathrm{mod}\ p)$.

Hệ quả 2.12. *Nếu p là số nguyên tố và a là số nguyên không chia hết cho p thì* a^{p-2} *là nghịch đảo của a modulo p*.

Hệ quả 2.13. *Nếu a và b là các số nguyên dương, p nguyên tố, p/a thì các nghiệm của đồng dư thức tuyến tính* $ax \equiv b(mod\ p)$ *là các số nguyên x sao cho* $x \equiv a^{p-2}b(mod\ p)$.

§6. Số giả nguyên tố.

Theo định lí Fermat, nếu n là số nguyên tố và b là số nguyên tuỳ ý, thì $b^n \equiv b(\mathrm{mod}\ n)$. Do đó nếu tồn tại số b sao cho $b^n \not\equiv b(\mathrm{mod}\ n)$ thì n phải là hợp số. Trong nhiều ứng dụng , chúng ta lại cần đến các thuật toán để chỉ ra một số n là số nguyên tố. Trong trường hợp này, ta không thể dùng định lí Fermat bé, vì định lí ngược của nó không đúng. Tuy nhiên, nếu một số nguyên thoả mãn các giả thiết của định lí Fermat bé thì "có nhiều khả năng" nó là một số nguyên tố! Ta có định nghĩa sau đây.

Định nghĩa 2.14. Giả sử b là một số nguyên dương. Nếu n là hợp số nguyên dương và $b^n \equiv b(\mathrm{mod}\ n)$ thì n được gọi là *số giả nguyên tố cơ sở b*.

Trong trường hợp $(n,b)=1$, ta thường dùng định nghĩa tương đương: $b^{n-1} \equiv b(\mathrm{mod}\ n)$.

Ví dụ. Số nguyên 561=3.11.17 là số giả nguyên tố cơ sở 2. Thật vậy, áp dụng định lí Fermat bé, ta có $2^{560}=(2^2)^{280} \equiv 1(\mathrm{mod}\ 3)$, $2^{560}=(2^{10})^{56} \equiv 1(\mathrm{mod}\ 11)$, $2^{560}=(2^{16})^{35} \equiv 1(\mathrm{mod}\ 17)$. Từ đó suy ra (bài tập 2.12) $2^{560} \equiv 1(\mathrm{mod}\ 561)$.

Nói chung các số giả nguyên tố ít hơn nhiều so với các số nguyên tố. Chẳng hạn, có tất cả 4550525112 số nguyên tố bé hơn 10^{10}, nhưng chỉ có 14884 số giả nguyên tố cơ sở 2 trong khoảng đó. Sự kiện này giải thích cách nói ở trên: Các số thoả mãn định lí Fermat bé có nhiều khả năng là số nguyên tố. Tuy nhiên đối với mọi cơ sở tuỳ ý, số các số giả nguyên tố là vô hạn. Chẳng hạn, ta chứng minh điều đó đối với cơ sở 2.

Định lí 2.15. *Có vô số số giả nguyên tố cơ sở 2.*

Chứng minh. Giả sử n là một số giả nguyên tố cơ sở 2, ta sẽ chứng tỏ rằng, $m=2^n-1$ cũng là số giả nguyên tố cơ sở 2. Theo giả thiết, n là hợp số, chẳng hạn $n=dt$ $(1<d,t<n)$, và $2^{n-1}\equiv 1 (\mathrm{mod}\ n)$. Dễ thấy rằng m là hợp số, vì $(2^d-1) \mid (2^n-1)=m$. Do n là giả nguyên tố, tồn tại k sao cho $2^n-2=kn$. Ta có $2^{m-1}=2^{kn}$, và do đó, $m=(2^n-1)\mid(2^{nk}-1)=2^{m-1}-1$, tức là $2^{m-1}\equiv 1 (\mathrm{mod}\ m)$. Vậy số m là giả nguyên tố cơ sở 2.

Như vậy, để kiểm tra một số có phải là số nguyên tố hay không, trước tiên ta xem nó có là giả nguyên tố cơ sở 2 hay không, sau đó có thể tiếp tục kiểm tra đối với các cơ sở khác. Tuy nhiên, tồn tại các số giả nguyên tố với mọi cơ sở, đó là các số Carmichael.

Định nghĩa 2.16. Hợp số nguyên n thoả mãn $b^{n-1}\equiv 1 (\mathrm{mod}\ n)$ với mọi số nguyên dương b sao cho $(n,b)=1$ được gọi là *số Carmichael*.

Ví dụ. Số nguyên $561=3.11.17$ là một số Carmichael. Thật vậy, nếu $(b,561)=1$ thì $(b,3)=(b,11)=(b,17)=1$. Theo định lí Fermat bé, ta có $b^2\equiv 1 (\mathrm{mod}\ 3)$, $b^{10}\equiv 1 (\mathrm{mod}\ 11)$, $b^{16}\equiv 1 (\mathrm{mod}\ 17)$. Do đó, viết $560=2.280=10.56=16.35$ ta được:

$$b^{560}=(b^2)^{280}\equiv 1 (\mathrm{mod}\ 3),$$

$$b^{560}=(b^{10})^{56}\equiv 1 (\mathrm{mod}\ 11),$$

$$b^{560}=(b^{16})^{35}\equiv 1 (\mathrm{mod}\ 17).$$

Từ đó suy ra (bài tập 2.12): $b^{560}\equiv 1 (\mathrm{mod}\ 561)$.

Giả thuyết sau đây mới được chứng minh rất gần đây ([AGP]): tồn tại vô hạn số Carmichael.

Định lí sau đây cho một cách tìm số Carmichael.

Định lí 2.17. Nếu $n=q_1q_2...q_k$, *trong đó* q_j *là các số nguyên tố khác nhau thoả mãn* $(q_j-1) \mid (n-1)$, *thì n là số Carmichael.*

Thật vậy, giả sử b là số nguyên dương, $(b,n)=1$. Khi đó, $(b,q_j)=1$ với mọi j, và $b^{q_j-1}\equiv 1(\mathrm{mod}\, q_j)$. Vì $(q_j-1) \mid (n-1)$ nên $b^{n-1}\equiv 1 (\mathrm{mod}\ q_j)$, và do đó, $b^{n-1}\equiv 1 (\mathrm{mod}\ n)$.

Phần đảo của định lí trên đây cũng đúng, tuy nhiên được chứng minh hơi dài nên ta sẽ bỏ qua. Độc giả nào quan tâm có thể tìm đọc trong [Ro].

Như vậy, việc kiểm tra nguyên tố sẽ khó khăn khi gặp phải các số Carmicheal. Tuy nhiên, ta có thể khắc phục bằng cách sau đây. Nếu gặp đồng dư $b^{n-1}\equiv 1 (\mathrm{mod}\ n)$, ta chuyển sang xét đồng dư $b^{(n-1)/2}\equiv x (\mathrm{mod}\ n)$. Nếu n là số nguyên tố thì $x\equiv 1$ hoặc $x\equiv -1 (\mathrm{mod}\ n)$, ngược lại thì n phải là hợp số (bài tập 2.22).

Ví dụ, với số Carmicheal bé nhất 561 ta có: $5^{(561-1)/2}=5^{280} \equiv 67(\text{mod } 561)$. Vậy, 561 là hợp số.

Về sau, ta sẽ đề cập đến những thuật toán kiểm tra nguyên tố hiện đại. Trong phần này, để thấy thêm ứng dụng của các định lí đồng dư vừa trình bày, ta tìm hiểu vài thuật toán đơn giản.

Định nghĩa 2.18. Giả sử n là số nguyên dương lẻ, $n-1=2^s t$, trong đó s là số nguyên không âm, t là số nguyên dương lẻ. Ta nói *n trải qua được kiểm tra Miller cơ sở b*, nếu hoặc $b^t \equiv 1(\text{mod } n)$, hoặc $b^{2^j t} \equiv -1(\text{mod } n)$, với j nào đó, $0 \le j \le s-1$.

Ta chứng tỏ rằng, nếu n là số nguyên tố thì *n trải qua được kiểm tra Miller cơ sở b với mọi số b sao cho $n|b$*. Thật vậy, giả sử $n-1=2^s t$. Đặt $x_k=b^{(n-1)/2^k}=b^{2^{s-k_t}}$, với $k=0,1,...,s$. Vì n là số nguyên tố nên $x_0 \equiv 1(\text{mod } n)$. Do đó $x_1^2 \equiv 1(\text{mod } n)$, tức là $x_1 \equiv 1(\text{mod } n)$ hoặc $x_1 \equiv -1(\text{mod } n)$. Tiếp tục quá trình như vậy ta sẽ đi đến kết luận rằng, hoặc $x_k \equiv 1(\text{mod } n)$ với $k=0,1,...,s$, hoặc $x_k \equiv -1(\text{mod } n)$ với một số nguyên k nào đó. Như vậy n trải qua được kiểm tra Miller cơ sở b.

Dễ thấy rằng, nếu n trải qua được kiểm tra Miller cơ sở b thì n sẽ là số giả nguyên tố cơ sở b. Ta có định nghĩa sau.

Định nghĩa 2.19. n được gọi là *số giả nguyên tố mạnh cơ sở b* nếu nó là hợp số và trải qua được kiểm tra Miller cơ sở b.

Như vậy các số giả nguyên tố mạnh lại còn ít hơn các số giả nguyên tố. Tuy nhiên, ta có định lí sau.

Định lí 2.20. *Tồn tại vô số số giả nguyên tố mạnh cơ sở 2*.

Thật vậy, giả sử n là một số giả nguyên tố cơ sở 2. Khi đó, $2^{n-1}=nk$ với số nguyên lẻ k nào đó. Đặt $N=2^n-1$, ta có

$$N-1=2^n-2=2(2^{n-1}-1)=2nk;$$

nghĩa là n là hợp số. Mặt khác,

$$2^{(N-1)/2}=2^{nk}=(2^n)^k \equiv 1(\text{mod } N).$$

Vậy với mỗi số giả nguyên tố n, ta xây dựng được số giả nguyên tố mạnh N và các số n khác nhau cho ta các số N khác nhau: định lí được chứng minh, bởi vì có vô số giả nguyên tố cơ sở 2.

Ta có thể dùng kiểm tra Miller để kiểm tra nguyên tố những số không lớn lắm. Ta biết rằng, số giả nguyên tố mạnh lẻ có sở 2 bé nhất là 2047. Như vậy, nếu n lẻ và $n<2047$, thì n là nguyên tố nếu nó trải qua kiểm tra Miller. Tương tự như vậy, số 1373653, là số giả nguyên tố mạnh lẻ bé nhất cơ sở 2 và 3, được dùng để kiểm tra nguyên tố những số bé hơn nó. Đối với cơ sở 2,3 và 5, số giả nguyên tố mạnh lẻ bé nhất là 25326001, trong trường hợp cơ sở 2,3,5,7,số tương ứng là 3215031751. Trong những số nhỏ hơn 25.10^9, chỉ có một số giả nguyên tố lẻ với cơ sở 2,3,5,7, đó là 3251031751. Như vậy, nếu $n<25.10^9$ là số lẻ trải qua kiểm tra Miller, thì n là số nguyên tố nếu nó khác với 3251031751.

Cách làm trên đây chỉ áp dụng được khi cần kiểm tra nguyên tố những số không lớn. Đối với những số lớn, ta có thể dùng thuật toán xác suất dựa trên định lí sau đây:

Định lí 2.21. *Nếu n là một hợp số dương lẻ thì tồn tại không quá (n-1)/4 cơ sở b, $1 \le b \le n-1$, sao cho n trải qua được kiểm tra Miller đối với các cơ sở đó.*

Định lí trên đây được chứng minh dựa vào khái niệm chỉ số mà ta không trình bày ở đây. Độc giả nào quan tâm có thể tìm đọc trong[Ro]. Nhờ định lí 2.21, ta có thể kết luận *n* là một hợp số nếu thấy nó trải qua kiểm tra Miller với hơn (n-1)/4 cơ sở. Tuy nhiên, việc kiểm tra như thế đòi hỏi quá nhiều thời gian.

Từ định lí 2.21 suy ra rằng, nếu số *b* được chọn ngẫu nhiên trong khoảng $1 \le b \le n-1$ thì *n* trải qua kiểm tra Miller cơ sở *b* với xác suất bé hơn 1/4. Như vậy, nếu ta chọn *k* số ngẫu nhiên thì xác suất để *n* trải qua kiểm tra Miller đối với *k* cơ sở đó sẽ bé hơn $1/4^k$. Khi *k* đủlớn, ví dụ *k*=20, xác suất đó quá nhỏ, nên với *n* trải qua với 20 cơ sở ngẫu nhiên thì có thể tin "hầu chắc chắn" rằng *n* là số nguyên tố. Từ đó ta có thuật toán xác suất sau đây.

Thuật toán Rabin-Miller (1980)

Cho $N \ge 3$ lẻ, thuật toán sau đây xác định rằng *N* là một hợp số, hoặc in ra thông báo *N* là số nguyên tố với xác suất lớn hơn $1-1/4^{20}$.

RM1. (Xuất phát). Đặt $q \leftarrow N-1$, $t \leftarrow 0$, và nếu q chẵn đặt $q \leftarrow q/2$, $t \leftarrow t+1$ (bây giờ ta có $N-1=2^t q$, với q lẻ). Sau đó đặt $c \leftarrow 20$.

RM2. (Chọn a mới). Chọn ngẫu nhiên số a trong khoảng $1<a<N$. Đặt $e \leftarrow 0$, $b \leftarrow a^q \bmod N$. Nếu $b=1$, chuyển sang RM4.

RM3.(Bình phương). Nếu $b \not\equiv \pm 1 \pmod N$ và $e<t-2$, ta đặt $b \leftarrow b^2 \bmod N$, $e \leftarrow e+1$. Nếu $b \ne N-1$, in ra thông báo "n là hợp số" và kết thúc thuật toán.

RM4. Đặt $c \leftarrow c-1$. Nếu $c>0$, chuyển sang RM2. Nếu $c=0$, in ra thông báo "N là số nguyên tố".

§7. Phân số liên tục.

Giả sử *a,b* là các số nguyên dương, *a>b*. Khi đó, phân số *a*/*b* có thể viết dưới dạng:

$$\frac{a}{b} = a_0 + \frac{c_0}{b} = a_0 + \frac{1}{\dfrac{b}{c_0}}.$$

Phân số b/c_0 lại có thể biểu diễn dưới dạng tương tự như vậy, và cuối cùng ta nhận được:

$$\frac{a}{b} = a_o + \cfrac{1}{a_1 + \cfrac{1}{\ldots a_{n-1} + \cfrac{1}{a_n}}}.$$

Cách viết như trên được gọi là biểu diễn số hữu tỷ a/b dưới dạng *phân số liên tục*.

Để đơn giản kí hiệu, ta thường dùng cách viết $a/b=[a_0;a_1,a_2,...,a_n]$. Phân số liên tục $[a_0;a_1,a_2,...,a_n]$ được gọi là *phân số liên tục hữu hạn*.

Dùng thuật toán Euclid, có thể biểu diễn mọi số hữu tỷ dưới dạng phân số liên tục hữu hạn. Thật vậy, ta có $a=a_0b+c_0$, $b=a_1c_0+c_1$,.... Ngược lại, rõ ràng mỗi phân số hữu hạn liên tục là một số hữu tỷ.

Ta cũng có thể biểu diễn một số thực tuỳ ý dưới dạng phân số liên tục. Tuy nhiên trong trường hợp này, phân số liên tục có thể không hữu hạn. Cách làm cũng hoàn toàn tương tự như khi làm với các số hữu tỷ.

Giả sử x là số thực tuỳ ý. Đặt $a_0=[x]$, phần nguyên của x, và $x_0=x-a_0$ là phần lẻ của x. Tiếp theo đó, ta đặt $a_1=[1/x_0]$, $x_1=1/x_0-a_1$. Tóm lại đối với mỗi số $i>1$, đặt $a_i=[1/x_{i-1}]$, $x_i=1/x_{i-1}-a_i$. Nếu ở bước thứ i nào đó, $x_i=0$ thì quá trình kết thúc (Điều này xảy ra khi và chỉ khi x là số hữu tỷ). Ngược lại, ta có biểu diễn x dưới dạng phân số liên tục vô hạn: $[a_0;a_1,a_2,...,a_n,...]$.

Nhiều khi để thuận tiện, ta dùng cách viết sau đây:

$$x= a_0 + \frac{1}{a_1 +} + \frac{1}{a_2 +} + \ldots + \frac{1}{a_n +} + \ldots$$

Các phân số liên tục định nghĩa như trên với các số a_i nguyên còn được gọi là các *phân số liên tục đơn giản*. Khi không đòi hỏi a_i là các số nguyên, mà có thể là các số thực tuỳ ý, ta cũng dùng cách viết

$$x=[a_0;a_1,a_2,...,a_n]= \quad a_0 + \frac{1}{a_1 +} + \frac{1}{a_2 +} + \ldots + \frac{1}{a_n}.$$

Khi có một phân số liên tục $x=[a_0;a_1,a_2,...,a_n,...]$, ta gọi các số sau đây là các *phân số hội tụ riêng của x:*

$$C_k= [a_0;a_1,a_2,...,a_k].$$

Định lí 2.22. *Giả sử $a_0,a_1,...,a_n$ là các số thực, trong đó $a_0,a_1,...,a_n>0$. Đặt $p_0=a_0$, $q_0=1$, $p_1=a_0a_1+1$, $q_1=a_1$, và với mỗi $k\geq 2$, $p_k=a_kp_{k-1}+p_{k-2}$, $q_k=a_kq_{k-1}+q_{k-2}$. Khi đó đối với các phân số hội tụ riêng C_k ta có:*

$$C_k= [a_0;a_1,a_2,...,a_k]= p_k/q_k.$$

Chứng minh. Ta chứng minh bằng qui nạp. Với $k=0$, $C_0=a_0=p_0/q_0$. Với $k=1$,

$$C_1=[a_0;a_1]=a_0+\frac{1}{a_1}=p_1/q_1.$$

Ta có:

$$C_{k+1}=[a_0;a_1,a_2,...,a_{k+1}]=a_0+\frac{1}{a_1+}+\frac{1}{a_2+}+\ldots+\frac{1}{a_{k+1}}$$

$$=[a_0;a_1,a_2,...,a_{k-1},a_k+\frac{1}{a_{k+1}}]=\frac{(a_k+\frac{1}{a_{k+1}})p_{k-1}+p_{k-2}}{(a_k+\frac{1}{a_{k+1}})q_{k-1}+q_{k-2}}.$$

theo giả thiết qui nạp. Tính toán đơn giản dựa vào định nghĩa các số p_k,q_k, ta được:

$$C_{k+1}=p_{k+1}/q_{k+1.}$$

Định lí được chứng minh.

Định lí 2.23. *Với mọi* $k\geq 1$, *ta có:*

$$p_kq_{k+1}-p_{k-1}q_k=(-1)^{k-1}.$$

Từ đó ta suy ra ngay rằng, các số p_k,q_k nguyên tố cùng nhau.

Định lí 2.24. *Ta có:*

$$C_1>C_3>C_5>...$$

$$c_0<C_2<C_4>...$$

$$C_{2j+1}>C_{2k}, \text{ với mọi } j,k$$

$$\lim C_k=x.$$

Chứng minh các định lí trên (bằng quy nạp) được dành cho độc giả. Có thể thấy rằng, tên gọi "phân số liên tục riêng " được giải thích bằng định lí trên đây.

Định lí 2.25. *Giả sử n là một số tự nhiên không chính phương và* p_k,q_k *là các phân số hội tụ riêng của* $\sqrt{n}$. *Ta đặt* $\alpha_0=\sqrt{n}$, *và các số* α_k, Q_k, P_k *được định nghĩa theo công thức sau:*

$$\alpha_k=(P_k+\sqrt{n})/Q_k,$$

$$a_k=[\alpha_k],$$

$$P_{k+1}=a_kQ_k-P_k$$

$$Q_{k+1}=(n-P_{n+1}^2)Q_k$$

Khi đó ta có:

$$p_k^2-n\,q_k^2=(-1)^{k-1}Q_{k+1}.$$

Chứng minh. Áp dụng định lí vừa chứng minh, ta có:

$$\sqrt{n} = \alpha_0 = [a_0; a_1, a_2, \ldots, a_{k+1}] = \frac{a_{k+1}p_k + p_{k-1}}{a_{k+1}q_k + q_{k-1}}.$$

Từ đó, vì $a_{k+1} = (P_{k+1} + \sqrt{n})/ Q_{k+1}$, ta được:

$$\sqrt{n} = \frac{(P_{k+1} + \sqrt{n})p_k + Q_{k+1}p_{k-1}}{(P_{k+1} + \sqrt{n})q_k + Q_{k+1}q_{k-1}}.$$

Vậy,

$$nq_k + (P_{k+1}q_k + Q_{k+1}q_{k-1})\sqrt{n} = (P_{k+1}p_k + Q_{k+1}p_{k-1}) + p_k\sqrt{n}.$$

Từ đó suy ra:

$$nq_k = P_{k-1}p_k + Q_{k+1}p_{k-1},$$

$$p_k = P_{k+1}q_k + Q_{k+1}q_{k-1}.$$

Nhân đẳng thức thứ nhất với q_k, đẳng thức thứ hai với p_k và trừ đẳng thức thứ hai cho đẳng thức thứ nhất, ta thu được kết quả cần chứng minh.

Sau đây ta sẽ áp dụng phân số liên tục để tìm một thuật toán phân tích số nguyên ra thừa số nguyên tố. Nói chính xác hơn, ta sẽ xây dựng một thuật toán để với số tự nhiên n cho trước, tìm ước số không tầm thường (khác 1 và n).

Ta xuất phát từ nhận xét đơn giản sau đây: Nếu ta tìm được các số x, y sao cho $x-y \neq 1$ và $x^2-y^2=n$ thì ta tìm được số ước không tầm thường của n, vì $n = x^2-y^2=(x-y)(x+y)$.

Bây giờ, giả sử ta có kết quả yếu hơn, chẳng hạn tìm được x,y sao cho $x^2 \equiv y^2 \pmod n$ và $0<x<y<n$, $x+y \neq n$.

Khi đó n là một ước của tích $(x-y)(x+y)$, và rõ ràng n không là ước của $x+y$ cũng như $x-y$. Như vậy các ước số chung $d_1=(n,x-y)$ và $d_2=(n,x+y)$ là các ước số không tầm thường của n. Các ước số này tìm được một cách nhanh chóng nhờ thuật toán Euclid. Định lí 2.25 cho ta phương pháp để tìm các số x,y cần thiết.

Theo định lí 2.25 ta có:

$$p_k^2 \equiv (-1)^{k-1}Q_{k-1} \pmod n.$$

Như vậy, vấn đề là phải tìm được các Q_k với chỉ số chẵn, và là một số chính phương. Mỗi lần tìm được một số Q_k như vậy, ta tìm được một ước của n (cũng có thể xảy ra trường hợp ước đó là tầm thường: các p_k, Q_k không nhất thiết bé hơn n nên điều kiện n không phải là ước của x+y và x-y có thể không thoả mãn).

Ví dụ. 1). Phân tích số 1037 ra thừa số bằng cách sử dụng phân số liên tục.

ta có $\alpha = \sqrt{1037} = 32,2...$, $Q_1=1, Q_2=49$, $p_1=129$. Như vậy, $129^2 \equiv 49 \pmod{1037}$. Do đó, $129^2-7^2=(129-7)(129+7) \equiv 0 \pmod{1037}$. Tính các ước chung lớn nhất, ta được: $(129-7,1037)=61$, $(129+7,1037)=17$. Ta có hai ước của 1037, và trong trường hợp này có khai triển $1037=61.17$.

2) Phân tích 1000009. Ta tính được $Q_1=9$, $Q_2=445$, $Q_3=873$, $Q_4=81$. Như vậy, $p_3^2 \equiv 9^2 (\mathrm{mod}\ 100009)$: ta không thu được ước không tầm thường. Tính toán tiếp tục, ta có: $Q_{18}=16$ là một số chính phương, $p_{17}=494881$. Bằng thuật toán đã mô tả, ta tìm được các ước số 293, 3413.

Bài tập và tính toán thực hành chương 2

I. Bài tập.

2.1. Chuyển số (1999) từ cơ số 10 sang cơ số 7, số (6105) từ cơ số 7 sang cơ số 10.

2.2. Chuyển các số 10001110101 và 11101001110 từ cơ số 2 sang cơ số 16 (kí hiệu các chữ số của cơ số 16 bởi 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E).

2.3. Chứng minh rằng mọi vật nặng không quá 2^k-1 (với trọng lượng là số nguyên) đều có thể cân bằng một cái cân hai đĩa, sử dụng các quả cân 1, 2, 2^2,...,2^{k-1}.

2.4. Chứng minh rằng mọi số nguyên đều có thể biểu diễn duy nhất dưới dạng

$$e_k 3^k+e_{k-1}3^{k-1}+...+e_1 3+e_0,$$

trong đó e_j=-1, 0 hoặc 1; j=0,1,...,k.

2.5. Chứng minh rằng, mọi số thực $\alpha \in$R, $0 \le \alpha <1$ đều có thể biểu diễn duy nhất dưới dạng cơ số *b*

$$\alpha = \sum_{j=1}^{\infty} c_j / b^j \ , 0 \le c_j \le b\text{-}1,$$

thoả mãn điều kiện: với mọi N, tồn tại $n \ge N$ để $c_n \ne b$-1.

2.6. Áp dụng bài 2.5, viết π trong cơ số 2 với 10 chữ số sau dấu phẩy.

2.7. a) Chứng minh rằng mọi số nguyên dương *n* đều có biểu diễn Cantor duy nhất dưới dạng sau:

$$n=a_m m!+a_{m-1}(m-1)!+...+a_2 2!+a_1 1!.$$

b) Tìm khai triển Cantor của 14, 56, 384.

2.8. Giả sử *a* là số nguyên (trong cơ số 10) với bốn chữ số sao cho không phải mọi chữ số là như nhau. *a*' là số nhận được từ *a* bằng cách viết các chữ số theo thứ tự giảm dần, *a*'' là số nhận được bằng cách viết các chữ số theo thứ tự tăng dần. Đặt T(a)=a'-a''. Ví dụ: T(1998)=9981-1899.

a) Chứng minh rằng số nguyên duy nhất (không phải 4 chữ số đều như nhau) sao cho T(a)=a là a=6174.

b) Chứng minh rằng nếu *a* là số nguyên dương 4 chữ số, không phải mọi chữ số đều như nhau, thì dãy *a, T(a), T(T(a)),...* nhận được bằng cách lặp phép toán T, sẽ dừng ở số 6174 (được gọi là *hằng số Kapreka*)

2.9. Ước lượng thời gian cần thiết để tính *n*!.

2.10. Ước lượng thời gian cần thiết để chuyển một số *k*-bit sang hệ thập phân.

2.11. a) Chứng minh rằng, nếu A, B là các ma trận vuông cấp *n* thì để tìm tích AB (theo quy tắc nhân ma trận thông thường) ta cần n^3 phép nhân.

b) Chứng minh rằng có thể nhân hai ma trận vuông cấp hai mà chỉ cần 7 phép nhân, nếu sử dụng đồng nhất thức sau đây:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}=$$

$$\begin{pmatrix} a_{11}\ b_{11}+a_{12}b_{21} & x+(a_{21}+a_{22})(b_{12}-b_{11})+(a_{11}+a_{12}-a_{21}-a_{22})b_{22} \\ x+(a_{11}-a_{21})(b_{22}-b_{12})-a_{22}(b_{11}-b_{21}-b_{12}+b_{22}) & x+(a_{11}-a_{21})(b_{22}-b_{12})+(a_{21}+a_{22})(b_{12}-b_{11}) \end{pmatrix},$$

trong đó $x=a_{11}b_{11}-(a_{11}-a_{21}-a_{22})(b_{11}-b_{12}+b_{22})$.

c) Bằng quy nạp và tách ma trận $2n \times 2n$ thành 4 ma trận $n \times n$, chứng minh rằng có thể nhân hai ma trận $2^k \times 2^k$ chỉ với 7^k phép nhân và không ít hơn 7^{k+1} phép cộng.

d) Từ c) suy ra rằng có thể nhân hai ma trận vuông cấp n với $O(n^{\log}c^7)$ phép tính bit nếu mọi phần tử của ma trận có dưới c bit, với hằng số c nào đó.

2.12. Dùng sàng Eratosthenes để tìm mọi số nguyên tố bé hơn 1998.

2.13. Cho $Q_n=p_1p_2...p_n+1$, trong đó $p_1,p_2,...,p_n$ là n số nguyên tố đầu tiên. Tìm ước nguyên tố bé nhất của Q_n, với n=1,2,3,4,5,6.

Trong dãy Q_n có vô hạn hay hữu hạn số nguyên tố?

2.14. Chứng minh rằng tồn tại vô hạn số nguyên tố.

2.15. Chứng minh rằng nếu ước nguyên tố bé nhất p của một số nguyên dương n vượt quá $\sqrt[3]{n}$ thì n/p là số nguyên tố.

2.16. Chứng minh rằng không tồn tại một "bộ ba nguyên tố" nào p, p+2, p+4 ngoài 3,5,7.

2.17. Chứng minh rằng nếu a|x, b|x và a, b nguyên tố cùng nhau thì a.b|x.

2.18. Chứng minh rằng nếu a,m nguyên tố cùng nhau thì tồn tại nghịch đảo m mod b.

2.19. Cho a,b,c,m là các số nguyên, m dương. Giả sử d là ƯCLN của c và m. Khi đó, nếu $ac \equiv bc \pmod m$ thì $a \equiv b \pmod{m/d}$.

2.20. Giả sử $r_1, r_2,...,r_m$ là một hệ thặng dư đầy đủ modulo m, a là một số nguyên, nguyên tố cùng nhau với m, b là số nguyên tuỳ ý. Chứng minh rằng $ar_1+b, ar_2+b, ..., ar_m+b$ cũng là một hệ đầy đủ các thặng dư modulo m

2.21. Giả sử $a \equiv b \pmod{m_j}$, j=1,2,..., k, trong đó m_j là các số nguyên tố cùng nhau từng cặp. Chứng minh rằng $a \equiv b \pmod{m_1m_2...m_k}$.

2.22. Cho p là số nguyên tố. Chứng minh rằng $a^2 \equiv 1 \pmod p$ khi và chỉ khi $a \equiv \pm 1 \pmod p$.

2.23. Chứng minh rằng với mọi số nguyên không âm m,n và mọi số nguyên $a>1$, ta có

$$(a^m-1, a^n-1)=a^{(m,n)}-1.$$

2.24. a) Chứng minh rằng có thể tìm ước chung lớn nhất của hai số nguyên dương bằng thuật toán sau

$$(a,b)=\begin{cases} a & \text{nếu } a=b \\ 2(a/2,b/2) & \text{nếu } a,b \text{ chẵn} \\ (a/2,b) & \text{nếu } a \text{ chẵn}, b \text{ lẻ} \\ (a-b,b) & \text{nếu } a,b \text{ lẻ} \end{cases}$$

b) Dùng thuật toán trên để tìm (2106, 8318).

2.25. Chứng minh rằng, với mọi n, tìm được n số tự nhiên liên tiếp sao cho mỗi số đều có ước là số chính phương.

2.26. Giả sử $n=p_1p_2...p_k$, trong đó p_j là các số nguyên tố, và n là một số Carmichael. Chứng minh rằng $k\geq 3$. Áp dụng kết quả để tìm ra số Carmichael nhỏ nhất.

2.27. Chứng minh rằng, nếu 6m+1, 12m+1, 18m+1 đều là số nguyên tố thì (6m+1)(12m+1)(18m+1) là số Carmichael.

Chứng minh các số sau đây là số Carmichael:

1729, 294409, 56052361, 118901521, 172947529.

2.28. Chứng minh rằng 6601 là một số Carmichael.

2.29. Chứng minh rằng n=2047=23.89 là số giả nguyên tố mạnh cơ sở 2.

2.30. Cho b,m là các số nguyên nguyên tố cùng nhau, a,c là các số nguyên dương. Chứng minh rằng, nếu $b^a\equiv 1(\text{mod } m)$, $b^c\equiv 1(\text{mod } m)$ và $d=(a,c)$ thì $b^d\equiv 1(\text{mod } m)$.

2.31. Cho p là số nguyên tố, $p|b^m-1$. Chứng minh rằng, hoặc $p|b^d-1$ với d nào đó là ước thực sự của m (khác m), hoặc $d\equiv 1(\text{mod } m)$. Nếu $p>2$, m lẻ thì trong trường hợp sau, ta có $p\equiv 1(\text{mod } 2n)$.

2.32. Áp dụng bài tập trên để phân tích ra thừa số các số $2^{11}-1=2047$, $2^{13}-1=8191$, $3^{12}-1=531440$, $2^{35}-1=34355738367$.

2.33. Tìm phân số liên tục của các số $\sqrt{2}$, $\sqrt{3}$, $\sqrt{5}$, $(1+\sqrt{5})/2$.

2.34. Biết phân số liên tục của e là

e=[2;1,2,1,1,4,1,1,6,1,1,8,...]

a) Tìm 8 phân số hội tụ riêng đầu tiên của e.

b) Tìm xấp xỉ hữu tỷ tốt nhất của e có mẫu số bé hơn 100.

2.35. Cho α là một số vô tỷ. Chứng minh rằng, hoặc $|\alpha-p_k/q_k|<1/2q_k^2$, hoặc $|\alpha-p_{k+1}/q_{k+1}|<1/2q_{k+1}^2$.

2.36. Cho $f(x)$ là một đa thức tuỳ ý với hệ số nguyên. Chứng minh rằng tồn tại vô hạn số nguyên k sao cho $f(k)$ là hợp số.

II. Thực hành tính toán trên máy

Đối với tất cả các chương, tính toán thực hành trên máy tính với chương trình Maple được bắt đầu bằng dòng lệnh:

```
[>with(numtheory);
```

Các phép toán số học (phép cộng [+], phép trừ [-], phép nhân [*], phép chia [/], phép luỹ thừa [^], khai căn bậc hai [sqrt(.)],...) được viết và thực hiện theo thứ tự quen biết.

Luôn luôn ghi nhớ rằng cuối dòng lệnh phải là dấu chấm phẩy (;) hoặc dấu (:). Muốn thực hiện dòng lệnh nào thì phải đưa con trỏ về dòng lệnh đó (sau dấu chấm phẩy) và nhấn phím [Enter]. Hãy thực hiện các dòng lệnh theo đúng trình tự trước sau, vì một số tính toán trong các bước sau có thể yêu cầu kết quả từ các bước trước.

II. 1. Thực hành kiểm tra một số là số nguyên tố

Để kiểm tra một số *n* có phải là số nguyên tố hay không ta thực hiện lệnh như sau:

```
[>isprime(n);
```

Sau dấu (;) ấn phím "Enter". Nếu trên màn hình hiện ra chữ "true" thì *n* là số nguyên tố, nếu trên màn hình hiện ra chữ "false" thì *n* là hợp số.

Thí dụ: Số 2546789 có phải là số nguyên tố hay không?

```
[>isprime(n);
                                  False
```

Vậy 2546789 không phải là số nguyên tố.

II. 2. Thực hành tìm ước chung lớn nhất

Để thực hành tìm ước chung lớn nhất của hai số a và b, hãy vào dòng lệnh có cú pháp như sau:

```
[>gcd(a,b);
```

Sau dấu (;) ấn phím "Enter" thì việc tìm ước chung lớn nhất sẽ được thực hiện và sẽ có ngay kết quả.

Thí dụ: Tìm ước số chung lớn nhất của 2 số 157940 và 78864.

Thực hiện bằng câu lệnh sau:

```
[> gcd(157940,78800);
                                    20
```

Vậy ước chung lớn nhất của 157940 và 78864 là 20.

II. 3. Phân tích ra thừa số nguyên tố

Để phân tích số *n* ra thừa số nguyên tố ta thực hiện lệnh sau:

```
[>ifactor(n);
```

Sau dấu (;) ấn phím "Enter" thì việc phân tích *n* ra thừa số nguyên tố sẽ được thực hiện và sẽ có ngay kết quả.

Thí dụ: Phân tích số 122333444455555666666777777788888888999999999 ra thừa số nguyên tố.

Ta thực hiện như sau:

```
[>
ifactor(122333444455555666666777777788888888999999999);
```

```
   (3)(1224191378520521031389750603311206734714 3)(3331)
```

Ta cũng có thể dùng lệnh trên để kiểm tra xem một số *n* có phải là số nguyên tố hay không

II. 4. Thực hành kiểm tra một số là số Carmichael

Ta nhớ lại Định lí 2. 17 như sau:

Định lí 2.17. *Nếu* $n=q_1q_2...q_k$, *trong đó* q_j *là các số nguyên tố khác nhau thoả mãn* $(q_j-1) | (n-1)$, *thì n là số Carmichael*.

Do đó để kiểm tra xem một số *n* có phải là số Carmichael hay không ta thực hiện theo các bước sau:

Bước 1: Phân tích *n* thành tích các thừa số nguyên tố, ta thực hiện bằng dòng lệnh:

```
[>ifactor(n);
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ hiện ra kết quả phân tích *n* ra thừa số nguyên tố. Nếu *n* là hợp số và có dạng $n=q_1q_2...q_k$, trong đó q_j là các số nguyên tố khác nhau thì thực hiện tiếp bước kiểm tra thứ 2. Nếu không thì có thể khẳng định *n* không phải là số Carmichael.

Bước 2:. Thực hiện các phép tính chia $(n-1):(q_j-1)$, ta thực hiện bằng dòng lệnh sau:

```
[>(n-1)/(q_j-1);
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ hiện ra kết quả thương của phép chia**.** Nếu với mọi $j=1,2, ..., k$ các thương tìm được là các số nguyên thì ta khẳng định *n* là số Carmichael, nếu không thì trả lời không phải.

Thí dụ 1: Số 6601 có phải là số Carmichael hay không?

Thực hiện kiểm tra như sau:

```
[>ifactor(6601);
```

```
                          (7)(23)(41)
```

6601 được phân tích thành các thừa số nguyên tố khác nhau, vậy có thể nghi ngờ nó là số Carmichel. Để kiểm tra xem nó có thực sự là số Carmichel hay không, ta thực hiện các lệnh sau:

```
[>(6601-1)/(7-1);
                              1100
[>(6601-1)/(23-1);
                               300
[>(6601-1)/(41-1);
                               165
```

Vậy 6601 là số Carmichael.

Thí dụ 2: Số 6 có phải là số Carmichael hay không?

Thực hiện kiểm tra như sau:

```
[>ifactor(6);
                             (2)(3)
[>(6-1)/(2-1);
                               5
[>(6-1)/(3-1);
                              5/2
```

Vậy 6 không phải là số Carmichael.

Thí dụ 3: Số 45 có phải là số Carmichael hay không?

Thực hiện kiểm tra như sau:

```
[>ifactor(45);
                            (3)^2(5)
```

Số 45 không thoả mãn bước thứ nhất.

Vậy 45 không phải là số Carmichael.

II. 5. Thực hành kiểm tra một số là giả nguyên tố

Cho hai số nguyên dương *n, b*. Để kiểm tra xem *n* có phải là số giả nguyên tố cơ sở *b* hay không ta thực hiện các bước như sau:

Bước 1: Kiểm tra *n* là hợp số, ta thực hiện dòng lệnh:

```
[>isprime(n);
```

Sau dấu (;) ấn phím "Enter". Nếu trên màn hình hiện ra chữ "true" thì *n* là số nguyên tố, nếu trên màn hình hiện ra chữ "false" thì *n* là hợp số. Nếu *n* là số nguyên tố thì *n* không phải là số giả nguyên tố cơ sở *b*. Nếu ngược lại thực hiện tiếp bước 2.

Bước 2: Kiểm tra đồng dư thức $b^n-b \equiv 0(mod\ n)$, thực hiện bằng dòng lệnh:

```
[>b&^n-b mod n;
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ hiện ra kết quả. Nếu đó là số 0 thì *n* là số giả nguyên tố cơ sở *b*.

Thí dụ1: Số 561 có phải là số giả nguyên tố cơ sở 2 hay không?

Ta thực hiện các lệnh sau:

```
[>isprime(561);
                              false
[>2&^561-2 mod 561;
                                0
```

Vậy 561 là số giả nguyên tố cơ sở 2.

Thí dụ 2: Số 1224191378520521031389750603311206734714 3 có phải là số giả nguyên tố cơ sở 8 hay không?

Ta thực hiện các lệnh sau:

```
[>ispime(1224191378520521031389750603311206734714 3);
                              true
```

Số 1224191378520521031389750603311206734714 3 là một số nguyên tố. Do đó 1224191378520521031389750603311206734714 3 không phải là số giả nguyên tố cơ sở 8.

Thí dụ 3: Số 326 có phải là số giả nguyên tố cơ sở 3 hay không?

Ta thực hiện các lệnh sau:

```
[>isprime(326);
                              false
[>3&^326-3 mod 326;
                                6
```

Vậy 326 là không phải là số giả nguyên tố cơ sở 3.

II. 6. Thực hành kiểm tra một số là số giả nguyên tố mạnh

Cho *n* là số nguyên dương lẻ, *b* là số nguyên dương. Để kiểm tra *n* có phải là số giả nguyên tố mạnh cơ sở *b* hay không ta thực hiện theo các bước sau:

Bước 1: Kiểm tra *n* là hợp số, ta thực hiện bằng dòng lệnh:

```
[>isprime(n);
```

Sau dấu (;) ấn phím "Enter". Nếu trên màn hình hiện ra chữ "true" thì n là số nguyên tố, nếu trên màn hình hiện ra chữ "false" thì n là hợp số. Nếu n là số nguyên tố thì n không phải là số giả nguyên tố mạnh cơ sở b. Nếu ngược lại thực hiện tiếp bước 2.

Bước 2: Phân tích n-1 ra thừa số nguyên tố, ta thực hiện bằng dòng lệnh:

```
[>ifactor(n-1);
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ hiện ra sự phân tích của n-1 và ta thu được kết quả có dạng $n-1=2^s t$, trong đó s là số nguyên dương, t là số nguyên dương lẻ.

Bước 3: Kiểm tra đồng dư thức $b^t-1 \equiv 0 (mod\ n)$. Vào lệnh

```
[>b&^t-1 mod n;
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ hiện ra kết quả. Nếu đó là số 0 thì n là số giả nguyên tố mạnh cơ sở b, nếu kết quả là một số khác 0 ta thực hiện tiếp bước 4.

Bước 4: Kiểm tra các đồng dư thức $(b^{2^j t}+1) \equiv 0 (mod\ n)$ với j=0,...s-1, ta thực hiện dòng lệnh:

```
[>seq (b&^((2^j)t)+1 mod n, j=0..s-1);
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ hiện ra dãy kết quả. Nếu trong dãy kết quả có một số là số 0 thì n là số giả nguyên tố mạnh cơ sở b.

Thí dụ: Số 2047 có phải là số giả nguyên tố mạnh cơ sở 2 hay không?

Thực hiện kiểm tra như sau:

```
[>isprime(2047);
                              false
```

Do đó n là hợp số. Tiếp tục thực hiện lệnh

```
[>ifactor(n-1);
                          (2)(3)(11)(31)
```

Tiếp tục thực hiện lệnh

```
[>2&^(3*11*31)-1 mod 2047;
                                 0
```

Vậy 2047 là số giả nguyên tố mạnh cơ sở 2.

II. 7. Thực hành biểu diễn một số dưới dạng phân số liên tục

1. Biểu diễn số n dưới dạng phân số liên tục theo cách thông thường với số thương trong biểu diễn là k, ta dùng lệnh:

```
[>cfrac(n,k);
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ xuất hiện kết quả.

Thí dụ: Biểu diễn π dưới dạng phân số liên tục theo cách thông thường với 6 thương.

Ta thực hiện lệnh:

```
[> cfrac (Pi,6);
```

$$3+\cfrac{1}{7+\cfrac{1}{15+\cfrac{1}{1+\cfrac{1}{292+\cfrac{1}{1+\cfrac{1}{1+\ldots}}}}}}$$

2. Biểu diễn số *n* dưới dạng phân số liên tục theo cách đơn giản với số chữ số trong biểu diễn là *k*, ta dùng lệnh:

```
[>cfrac(n,k,'quotients');
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ xuất hiện kết quả.

Thí dụ: Biểu diễn π dưới dạng phân số liên tục theo cách viết đơn giản với 100 chữ số biểu diễn.

Ta thực hiện lệnh:

```
[> cfrac (Pi,100,'quotients');

[3,7,15,1,292,1,1,1,2,1,3,1,14,2,1,1,2,2,2,2,1,84,2,1,1,
15,3,13,1,4,2,6,6,99,1,2,2,6,3,5,1,1,6,8,1,7,1,2,3,7,1,
2,1,1,12,1,1,1,3,1,1,8,1,1,2,1,6,1,1,5,2,2,3,1,2,4,4,16,
1,161,45,1,22,1,2,2,1,4,1,2,24,1,2,1,3,1,2,1,1,10,2,...]
```

3. Biểu diễn số *n* dưới dạng phân số liên tục theo chu kỳ tuần hoàn, ta dùng lệnh:

```
[>cfrac(n,'periodic');
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ xuất hiện kết quả.

Thí dụ: Biểu diễn $3^{1/2}$ dưới dạng phân số liên tục theo chu kỳ tuần hoàn.

Ta thực hiện lệnh:

```
[>cfrac (3^(1/2),'periodic');
```

$$1+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{1+\cfrac{1}{2+\ldots}}}}$$

4. Biểu diễn số *n* dưới dạng phân số liên tục theo chu kỳ tuần hoàn đơn giản, ta dùng lệnh:

```
[>cfrac (n,'periodic','quotients');
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ xuất hiện kết quả.

Thí dụ: Biểu diễn $3^{1/2}$ dưới dạng phân số liên tục theo chu kỳ tuần hoàn đơn giản.

Ta thực hiện lệnh:

```
[> cfrac (3^(1/2),'periodic','quotients');
                                [[1], [1, 2]]
```

II. 8. Thực hành tìm phân số hội tụ thứ *k* của một số

Để thực hành tìm phân số hội tụ thứ *k* của một số *n*, ta thực hiện theo các lệnh sau:

Bước 1: Biểu diễn *n* dưới dạng phân số liên tục

```
[> cf:= cfrac(n);
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ xuất hiện sự biểu diễn

Bước 2: Tính phân số hội tụ thứ *k*

```
[> nthconver(cf,k);
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ xuất hiện ra kết quả.

Trong quá trình thực hiện ta không cần biết kết quả hiện thị ở bước 1, do đó có thể thay dấu (;) bằng dấu (:) ở dòng lệnh đầu tiên (`[>cf:=cfrac(n):`). Khi đó trên màn hình sẽ hiện ra dấu nhắc ([>) để thực hiện tiếp lệnh thứ 2.

Thí dụ: Tính phân số hội tụ thứ 5 của *e*.

Ta thực hiện như sau:

```
[> cf:= cfrac(exp(1));
```

$$cf := 2 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{4 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{6 + \cfrac{1}{1+\ldots}}}}}}}}}$$

```
[> nthconver(cf,5);
```

$$\frac{87}{32}$$

Như vậy, phân số hội tụ thứ 5 của e là $\frac{87}{32}$.

II. 8. Thực hành đổi cơ số

1. Để thực hành đổi một số n từ cơ 10 sang cơ số b ta dùng dòng lệnh sau:

```
[>convert(n,base,b);
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ hiện lên một dòng kết quả. Chú ý rằng kết quả đưa ra trên màn hình được viết theo thứ tự ngược lại.

Thí dụ 1: Đổi số 24564 từ cơ số 10 sang cơ số 6.

Ta thực hành như sau:

```
[>convert(24564,base,6);

                              [0, 2, 4, 5, 0, 3]
```

Vậy ta được số là $(305420)_6$.

Chú ý: Trong trường hợp cơ số b >10, ta vẫn thực hiện dòng lệnh đổi cơ số như bình thường. Tuy nhiên, sau khi nhận được kết quả, để tránh nhầm lẫn ta thực hiện việc đặt tương ứng các số lớn hơn 10 với các kí hiệu nào đó. Ta xem ví dụ sau:

Thí dụ 2: Đổi số 45676 từ cơ số 10 sang cơ số 15, trong đó đặt 10=A, 11=B,12=C,13=D,14=E.

Ta thực hành như sau:

```
[>L:=convert(45676,base,6):
[>subs(10=A,11=B,12=C,13=D,14=E,L);

                                 [1, 0, 8, D]
```

Vậy ta được số là $(D801)_{15}$.

2. Để thực hành đổi một số n từ cơ số a sang cơ số b ta dùng dòng lệnh sau:

```
[> convert(n,base,a,b);
```

Sau dấu (;) ấn phím "Enter" trên màn hình sẽ hiện lên một dòng kết quả. Chú ý rằng kết quả đưa ra trên màn hình được viết theo thứ tự ngược lại.

Thí dụ: Đổi số 305420 trong cơ số 6 sang cơ số 10.

Ta thực hiện dòng lệnh

```
[> convert([0,2,4,5,0,3],base,6,10);

                                [4, 6, 5, 4, 2]
```

Vậy ta có kết quả là $(24564)_{10}$