Số học - thuật toán

Tác giả: Ngô Minh Đức

Lý thuyết số, hay số học là lĩnh vực nghiên cứu về các số nguyên. Trong tài liệu này, chúng ta sẽ đề cập đến một số kiến thức và các thuật toán về số học thường gặp, bao gồm từ những vấn đề cơ bản.

1. Số nguyên tố

1.1. *Nếu p là ước nguyên tố bé nhất của n, thì* $p^2 \le n$

(n=pq, mà $p \le q$, do đó $p^2 \le pq = n$)

1.2. *Thuật toán kiểm tra tính nguyên tố*

Từ 1.1. ta có thuật toán kiểm tra tính nguyên tố của một số n, chạy trong thời gian $O(n^{1/2})$:

```
function isprime(n): boolean;
begin
     isprime:=false;
     if (n<=2) then exit;
     i:=2;
     while (i*i<=n) do
     begin
          if n mod i = 0 then
              exit;
          inc(i);
     end;
     isprime:=true;
end;
```

1.3. *Phân tích ra thừa số nguyên tố*

Cũng từ 1.1., ta có thuật toán phân tích một số n ra thừa số nguyên tố:

```
procedure factor(n);
begin
     i:=2;
     while (i*i<=n) do
     begin
          if (n mod i = 0) then
          begin
                    a:=0;
                    while (n mod i = 0) do
                    begin
                         n:=n div i;
                         inc(a);
                    end;
                    inc(m);
                    prime[m]:=i;
                    power[m]:=a;
          end;
     end;
     if (n>1) then
     begin
          inc(m);
          prime[m]:=n;
```

```
            power[m]:=1;
        end;
end;
```

Thông tin được trả về trong mảng prime và power: prime[i], power[i] tương ứng cho biết thừa số và số mũ thứ i trong phép phân tích n ra thừa số nguyên tố.

Lưu ý những dòng màu đỏ: sau khi thực hiện các phép chia, n có thể còn là một thừa số nguyên tố độc lập. Đây là phương pháp phân tích đơn giản nhất, được gọi là phép *thử chia*. Trong trường hợp xấu nhất, n là số nguyên tố, thuật toán chạy trong thời gian $O(n^{1/2})$

1.4. *Sàng số nguyên tố*

Khi cần biết các số nguyên tố đến một phạm vi nào đó, ví dụ từ 2 đến 10^8, sử dụng sàng số nguyên tố Eratosthenes sẽ hiệu qủa hơn về thời gian.

Thủ tục sau tạo sàng số nguyên tố từ 2 đến N:

```
procedure sieve(n);
begin
      fillchar(p, sizeof(p), true);
      for i:=2 to n do
            if (p[i]) then
               begin
                     j:=i+i;
                     while (j<=n) do
                     begin
                           p[j]:=false;
                           j:=j+i;
                     end;
               end;
end;
```

Thông tin trả về trong mảng p: p[i] = true nếu và chỉ nếu i là số nguyên tố.

Bạn có thể ước lượng thời gian chạy của thuật toán sàng Eratosthenes là O(nlogn), ta sẽ đề cập đến một số ước lượng cần thiết ở những phần sau.

1.5. *Ước lượng số số nguyên tố*

Kí hiệu π(n) là số số nguyên tố bé hơn n. Đây là một ước lượng tương đối tốt và ngắn gọn cho π(n):

$\pi(n) \approx n / \ln(n)$

Ví dụ, $\pi(10^6) \approx 10^6 / \ln(10^6) \approx 72382$

Con số này sẽ giúp cho việc ước lượng thời gian tính toán cho các bài toán liên quan đến số nguyên tố

1.6. *Bài tập*

Cho một dãy số nguyên dương n phần tử: $a_1, a_2, ..., a_n$

Dãy con của dãy số là dãy nhận được sau khi xóa đi một số phần tử nào đó

Yêu cầu: Tìm dãy con dài nhất, sao cho tổng của hai số liên tiếp là số nguyên tố

Input:

NT1.IN

Dòng 1: n

Dòng 2: n số nguyên dương, $a_1, a_2, ..., a_n$

Output:

NT1.OUT

Dòng 1: độ dài của dãy con tìm được ----------> f[i], O(n^2)

Giới hạn:

- $n <= 10^4$
- $a_i <= 10^4$

2. USCLN, thuật toán Euclid

2.1. *(a, b) = (b, a mod b)*

Thật vậy, từ đẳng thức

$$a = bq + r.$$

với r = a mod b

Ta thấy mọi ước chung d của a, b cũng là ước chung của b, r. Do đó (a, b) = (b, r).

2.2. *Thuật toán tìm USCLN*

Từ 2.1, ta có thuật toán Euclid tìm USCLN của a và b, viết dưới dạng đệ quy:

```
function gcd(a, b);
begin
     if (a<b) then
          gcd:=gcd(b, a)
     else if (b=0) then
          gcd:=a
     else
          gcd:=gcd(b,  a mod b);
end;
```

Với a, b ≤ n, bạn có thể ước lượng thời gian thực hiện thuật toán vào khoảng $O(\log_{10}n)$, tức là tỉ lệ với số chữ số của n.

2.3.

Nếu (a, b)=d, thì tồn tại hai số nguyên x, y sao cho

$$ax + by = d$$

2.4. *Thuật toán Euclid mở rộng*

Thuật toán Euclid mở rộng sẽ tìm USCLN d của a và b, đồng thời tìm được cả hai số nguyên x, y trong phần 2.3

Thuật toán Euclid mở rộng có thể diễn đạt bằng đệ quy như sau:

```
procedure ee(a, b, var x, var y);
var
     x2,y2;
begin
     if (a<b) then
          ee(b, a, x, y)
     else if (b=0) then
     begin
          x:=1;
          y:=0;
     end else
     begin
          ee(b, a mod b, x2, y2);
          x:=y2;
          y:=x2-(a div b)*y2;
     end;
end;
```

Giải thích:

Từ 2.1, ta đã biết (a, b) = (b, r) = d

ee(a, b, var x, var y) trả về giá trị x, y sao cho ax + by = d

Dòng lệnh màu đỏ chạy thủ tục đệ quy: tìm x_2, y_2 sao cho:

$$bx_2 + ry_2 = d$$

Mặt khác:

$$r = a - bq$$

Với

$$r = a \bmod b$$
$$q = a \operatorname{div} b$$

Do đó

$$bx_2 + (a-bq)y_2 = d$$
$$ay_2 + b(x_2 - qy_2) = d$$

Vậy

$$x = y_2$$
$$y = x_2 - qy_2$$

Cấu trúc đệ quy của thuật toán Euclid mở rộng cũng tương tự như thuật toán Euclid.

2.5. *Một số tính chất*

Giả sử

$$a = p_1^{a1} p_2^{a2} \ldots p_k^{ak}$$
$$b = p_1^{b1} p_2^{b2} \ldots p_k^{bk}$$

Định nghĩa:

$$\text{USCLN: } (a, b) = p_1^{\min(a1, b1)} p_2^{\min(a2, b2)} \ldots p_k^{\min(ak, bk)}$$
$$\text{BSCNN: } [a, b] = p_1^{\max(a1, b1)} p_2^{\max(a2, b2)} \ldots p_k^{\max(ak, bk)}$$

Tính chất:

- $(a, b) \times [a, b] = ab$
- $(a, b, c) = ((a, b), c) = (a, (b, c))$
- $[a, b, c] = [[a, b], c] = [a, [b, c]]$

Chú ý:

- Không có đẳng thức $(a, b, c)\,[a, b, c] = abc$

2.6. *Bài tập*

Cho dãy số nguyên dương n phần tử $a_1, a_2, \ldots, a_n$.

Yêu cầu:

Tìm dãy con liên tiếp dài nhất có USCLN > 1

Input:

NT2.INP

Dòng 1: n

Dòng 2: n số nguyên dương, $a_1, a_2, \ldots, a_n$

Output:

NT2.OUT

Dòng 1: độ dài của dãy con tìm được

Giới hạn:

- $n <= 30000$
- $0 < a_i <= 32767$
- ---> lưu các vector a[k] gồm chỉ số các phần tử chia hết cho k
- tổng phần tử của mọi vector cùng lắm là nlogn
- như vậy với mỗi k ta duyệt để tìm dãy liên tiếp dài nhất.

3. PT, HPT đồng dư

3.1. Nghịch đảo

Trở lại với thuật toán Eulcid mở rộng:

Giả sử ta thực hiện `ee(a, m, var x, var y)`

Trong trường hợp (a, m) = 1, ta thu được giá trị x, y sao cho:

$$ax + my = 1$$

Hay

$$ax \equiv 1 \pmod m$$

$(a, m) = 1 \Leftrightarrow \exists x, ax \equiv 1 \pmod m$
x được gọi là nghịch đảo của a theo modulo m, ký hiệu a^{-1}
Để tìm x, ta sử dụng thuật toán Euclid mở rộng

3.2. *Phương trình đồng dư bậc nhất*

$$ax \equiv b \pmod m \qquad (3.2)$$

3.2.1. Trường hợp (a, m) = 1
Theo 3.1. $\exists a^{-1}$, $aa^{-1} \equiv 1 \pmod m$
Do đó $aa^{-1}b \equiv b \pmod m$
Đặt $x = (a^{-1}b)$ thì x là một nghiệm của (3.2)
Giả sử tồn tại x', sao cho
$ax' \equiv b \pmod m$
Suy ra $ax \equiv ax' \pmod m$, mà (a, m)=1
Suy ra $x \equiv x' \pmod m$
Vậy $x = (a^{-1}b)$ là nghiệm duy nhất của (3.2) theo modulo m
3.2.2. Trường hợp (a, m) = d
Nếu d không là ước của b, hiển nhiên (3.2) vô nghiệm
Nếu b | d, xét phương trình:

$$(a/d)\, y \equiv (b/d) \pmod{(m/d)}$$

Ta có (a/d, m/d) = 1, do đó theo 3.2.1.

$$y \equiv (a/d)^{-1} (b/d) \pmod{(m/d)}$$

Đặt

$$y_0 = (a/d)^{-1} (b/d)$$

(3.2) có d nghiệm:

$$x_t = y_0 + t(m/d) \qquad \text{với } t = 0, 1, ..., d-1$$

theo modulo m

3.3. Định lý phần dư Trung Hoa

Nếu

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
$$...$$
$$x \equiv a_n \pmod{m_n}$$

và $m_1, m_2, ..., m_n$ đôi một nguyên tố cùng nhau thì x được xác định duy nhất theo modulo $M = m_1m_2...m_n$:

$$x \equiv a_1b_1c_1 + a_2b_2c_2 + ... + a_nb_nc_n \pmod M \qquad (3.3)$$

Trong đó

$$c_i = M / a_i$$
$$b_i = c_i^{-1} \pmod{a_i}$$

<u>Phương pháp để tìm công thức (3.3):</u>

Xét trường hợp $a_2 = a_3 = ... = a_n = 0$
Cần tìm x_1:

$$x_1 \equiv a_1 \pmod{m_1}$$
$$x_1 \equiv 0 \pmod{m_2}$$
$$...$$

$$x_1 \equiv 0 \pmod{m_n}$$

Ta sẽ tìm được

$$x_1 \equiv a_1 b_1 c_1 \pmod{M}$$

Tương tự, lại xét trường hợp $a_1 = a_3 = ... = a_n = 0$

$$x_2 \equiv a_2 b_2 c_2 \pmod{M}$$

...

$$x_n \equiv a_n b_n c_n \pmod{M}$$

Tổ hợp các kết qủa lại ta thu được công thức (3.3), phương pháp này được gọi là phương pháp chồng.

4. Phép chia hết

4.1.

Số các số nguyên dương không vượt qúa n chia hết cho d:

hay

n div d

4.2. *Ước số*

Giả sử

$$n = p_1^{a1} p_2^{a2} ... p_k^{ak}$$

4.2.1. *Số ước*

4.2.2. *Tích các ước*

Thật vậy, viết n dưới dạng tích hai thừa số:

Do đó

hay

4.2.3. *Tổng các ước*

(4.2.3)

Thật vậy: nếu (a, b) = 1 ta cm được

Do đó

Mà

Từ đó thu được công thức (4.2.3)

5. Số Fibonacci

5.1. *Cách tính nhanh F_n*

Viết dưới dạng tích hai ma trận:

Đặt

$$A =$$

$$v =$$

Ta có công thức:

(5.1)

Mà A^{n-1} có thể tính trong thời gian O(logn), do đó công thức (5.1.) cho phép ta tính F_n trong thời gian O(logn)

5.2. *Một số kết qủa thú vị*

5.2.1. UCLN của F_m, F_n

Công thức Lucas:

$(F_m, F_n) = F_{(m, n)}$

5.2.2. Xác định một số có phải là số Fibonacci

Gessel (1972):

n là số Fibonacci nếu và chỉ nếu $5n^2 + 4$ hoặc $5n^2 – 4$ là số chính phương

5.3. *Biểu diễn Zeckendorf*

5.3.1. Định lý Zeckendorf:

Mọi số nguyên dương đều được biểu diễn duy nhất dưới dạng tổng các số Fibonacci, trong đó không có hai số Fibonacci liên tiếp, nghĩa là dưới dạng:

với

$$a_k = 0 \text{ hoặc } a_k = 1$$

và

$$a_k a_{k+1} = 0$$

5.3.2. Ví dụ:

$$100 = 89 + 8 + 3$$

5.3.3. Thuật toán

Tìm biểu diễn Zeckendorf, hay còn gọi là biểu diễn dưới dạng cơ số Fibonacci của n:

```
while (n>0) do
begin
      f là số fibonnaci lớn nhất không vượt qúa n;
      chọn f vào biểu diễn;
      n:=n-f;
end;
```

hay ta có thể cài đặt như sau:

```
for i:=max downto 1 do
      while (Fi <= n) do
```

```
begin
        chọn Fi;
        n:=n-Fi;
end;
```

với max là chỉ số lớn nhất của số Fibonacci trong giới hạn làm việc

Tính đúng đắn:

Thuật toán tham 5.3.3. sẽ không bao giờ chọn hai số Fibonacci liên tiếp, thật vậy, giả sử thuật toán chọn F_{n-1}, F_{n-2} vào tổng, thì do ta đi qua danh sách số Fibonacci theo thứ tự giảm dần, thuật toán ắt đã chọn $F_n = F_{n-1} + F_{n-2}$ thay vì F_{n-1}, F_{n-2}

5.4. *Bài tập:*

5.4.1. http://acm.uva.es/p/v9/948.html

6. Tham khảo

Trên đây chỉ là một số vấn đề về số học - thuật toán thôi, các bạn nên tìm hiểu thêm, từ bất kỳ nguồn nào, internet, sách vở. Nếu có những vấn đề, những bài tập hay, hãy đóng góp lại cho mọi người!

Một số nguồn để các bạn tham khảo thêm:

Concerte Mathematics – A Foundation for Computer Science

Mathworld

Wikipedia

Thuật ngữ, ghi chú

1.

Số nguyên tố	PRIME
Kiểm tra tính nguyên tố	PRIMALITY TEST
Phân tích ra thừa số nguyên tố	PRIME FACTORIZATION
Sàng	SIEVE

Có rất nhiều thuật toán kiểm tra & phân tích ra thừa số nguyên tố hiệu qủa hơn, tuy nhiên những gì chúng ta trình bày là những thuật toán đơn giản nhất, sẽ sử dụng trong các bài tập và kì thi.

1.5. Xem PRIME NUMBER THEOREM

2.

USCLN	Greatest Common Divisor (GCD)
Thuật toán Euclid	Euclidean Algorithm
Thuật toán Euclid mở rộng	Extended Euclidean Algorithm
BSCNN	Least Common Multiple (LCM)

2.1. Về thời gian chạy của thuật toán Euclid, xem thêm LAMÉ'S THEOREM

2.3. Xem thêm BÉZOUT'S LEMMA

3.

Phương trình đồng dư	Congruence Equation
Định lý phần dư Trung Hoa	Chinese Remainder Theorem

5.

Biểu diễn Zeckendorf	Zeckendorf Representation

Bài 1 : Amount of Degrees

Một số A gọi là có bậc K đối với cơ số B nếu như :

- $A = B^{x1} + B^{x2} + \ldots + B^{xk}$

(trong đó $x1 \neq x2 \neq x3 \ldots \neq xk$)

Ví dụ :

- 17 có bậc 2 đối với cơ số 2 vì $17 = 2^4 + 2^0$.
- 151 có bậc 3 đối với cơ số 5 vì $151 = 5^3 + 5^2 + 5^0$.

Yêu cầu : Cho trước 1 đoạn [X,Y] . Hãy xác định xem trong đoạn này có bao nhiêu số có bậc K đối với cơ số B.

Giới hạn : + $1 \leq X \leq Y \leq 10^9$.

+ $1 \leq K \leq 20$, $2 \leq B \leq 9$.

+ Bộ nhớ : 200KB.

+ Time limit : 1 s.

Input

X Y K B

Output

Gồm 1 số duy nhất là kết quả tìm được .

Ví dụ

Input

15 20 2 2

Output

3

(Giải thích : Đó là các số $17 = 2^4 + 2^0$.

$18 = 2^4 + 2^1$.

$20 = 2^4 + 2^2$.)

---> Cách giải của tôi: Đưa A về hệ cơ số B rồi QHĐ từ điển f[i][j][0/1].

Thuật giải :

Ta sẽ đưa bài toán này về một bài toán nhỏ hơn đó là cho biết trong đoạn [1,A] có bao nhiêu số có bậc K đối với cơ số B.

Với mỗi giá trị A ta đều xác định được B^n sao cho $B^{n+1} > A$. Áp dụng tính chất: $B^n > B^{n-1} + B^{n-2} + \ldots B^0 \rightarrow A > B^{n-1} + B^{n-2} + \ldots B^0$.

$\rightarrow$ Tổng bất kỳ của 1 tổ hợp K phần tử của (n-1) phần tử này cũng đều < A (tức là : $A > B^{x1} + B^{x2} + \ldots + B^{xi} + \ldots + B^{xk}$ (với mọi $xi \leq n-1$)).

$\rightarrow$ Như vậy ta đã giải quyết xong vấn đề với các số mà không chứa B^n, nếu như không chứa B^n thì số lượng số thoả mãn sẽ = Tổ hợp chập K của (n-1) . Ta sẽ tiếp tục giải quyết vấn đề với các số mà có dạng = $B^n + B^{x2} + \ldots + B^{xk}$.Dễ thấy vấn đề ở đây lại quay về là tìm số lượng các số có bậc (K-1) đối với cơ số B trong đoạn $[1,A-B^n]$ và không chứa B^n, vậy thì ở đây ta chỉ cần xác định lại n mà thôi, n(mới) $\leq$ n (cũ) – 1.

Như vậy bài toán đã được giải quyết. Nói tóm lại đây là một bài khá đặc trưng cho QHĐ = Đệ quy với công thức truy hồi đơn giản :

Cal(A,K,B) = Tổ hợp chập K của (N-1) + Cal($A-B^n$,K-1,B).

Bài 2 : Nikifor – 2

Nikifor có một số X nhưng đó không phải là con số mà anh ta cần. Anh ta muốn đổi con số X này lấy con số Y mà anh ta thích. Để đổi được con số Y này anh ta phải chọn ra một cơ số K sao cho ta có thể đạt được số Y bằng cách xoá đi một vài chữ số của số X trong biểu diễn của hệ cơ số K .

Ví dụ : X = 19 , Y = 4 , K = 3.

Biểu diễn của X trong hệ cơ số 3 = 201

Biểu diễn của Y trong hệ cơ số 3 = 21

Vậy ta có thể đạt được số Y bằng cách xoá đi số 0 trong biểu diễn của số X .
Yêu cầu : Hãy tìm cơ số K nhỏ nhất có thể được .
Giới hạn : + $1 \le X, Y \le 10^6$.
+ Time limit 1 s , bộ nhớ 200 KB.
Input
X Y
Output
Nếu tồn tại K thì ghi ra K nhỏ nhất , nếu không tồn tại thì ghi ra "No Solution".
(Lưu ý số K có thể ≥ 10 , không nhất thiết là nhỏ hơn 10)
Ví dụs
Input
19 4
Output
2
Thuật giải :
Ta duyệt tất cả các số K từ 2 -> Y . Tuy nhiên điều muốn nói ở đây là làm cách nào để có thể kiểm tra xem với một cơ số K có đáp ứng được yêu cầu hay không một cách nhanh nhất ! Nếu như làm bình thường thì ta sẽ xây dựng được biểu diễn của X, của Y .Nếu xâu con chung dài nhất của X và Y = Y -> Đáp ứng yêu cầu ! Hoặc một cách cải tiến khác cũng đưa về xâu dựa trên nhận xét : xâu con chung dài nhất của X , Y = Y tương đương với các phần tử của Y xuất hiện đúng theo thứ tự này trong xâu X , tức là xâu con chung dài nhất $Y = X_{i1}X_{i2}\ldots X_{ik}$ thì $i1 < i2 < \ldots < ik$. -> Ta có thể sủ dụng thuật toán được mô tả sau đây : (Cải tiến 1)

```
Ok  := True ;
While Y <>  ∅ do Begin
      While X[1] <> Y[1] do Begin
            Delete(X,1,1) ;
            If  X =  ∅ then Begin
               Ok := False ; Exit ;
            End ;
      End ;
      Delete(X,1,1) ; Delete(Y,1,1) ;
End ;
```

Tuy có cải tiến như vậy nhưng về cơ bản cấp độ của thuật toán là khá cao (+ thời gian tạo xâu) và không thể chấp nhận được trong thời gian giới hạn là 1 s. Vì vậy ta cần có một thuật toán kiểm tra tốt hơn là đưa về xâu ! Ý tưởng của thuật toán là thay vì đổi ra xâu ta sử dụng phép **div** , nếu như trong hệ cơ số K ta sử dụng phép **div** K thì cũng giống như ta sử dụng phép **shr** trong hệ nhị phân ! Nó sẽ dịch chuyển số X về bên phải một chữ số ! Ta áp dụng cải tiến 1 trong trường hợp này là so sánh chữ số cuối cùng chứ không phải chữ số đầu tiên nữa ! Và chữ số cuối cùng của X = X mod K , chữ số cuối cùng của Y = Y mod K . Thủ tục Delete sẽ được thay bằng X = X div K . Như vậy thuật toán với 2 cải tiến này sẽ giảm được rất nhiều thời gian lãng phí do phải tạo xâu, sinh luỹ thừa ! Về mặt thời gian là chạy khá tốt !

Ngoài ra còn một thuật toán có cấp độ $O((\log Y)^3)$ mà ta không đề cập tới ở đây mà sẽ nhắc tới ở một chương khác. Mặc dù vậy với X , $Y \le 10^6$ thì ta có thể yên tâm là thuật toán này chạy chậm gấp 2,5 lần thuật toán duyệt với 2 cải tiến ở trên !

Bài 3 : Central Heating

Trong một trường đại học có một hệ thống N lò sưởi được đánh số từ 1-> N ! N lò sưởi này được điều khiển bởi N kỹ sư được đánh số từ 1 -> N ! Mỗi lò sưởi chỉ có 1 công tắc là tắt/bật lò sưởi mà thôi ! Tuy nhiên oái oăm thay là mỗi kỹ sư lại điều khiển 1 vài lò sưởi chứ không phải chỉ 1 lò sưởi! Và khi họ đi làm thì nếu như được chỉ định là trực ở lò sưởi thì họ sẽ đi thay đổi công tắc ở tất cả các lò sưởi mà

họ điều khiển, nếu đang bật -> tắt và tắt -> bật. Kỹ sư thứ i sẽ tới vào trường vào lúc i giờ. Ban giám hiệu trường muốn đảm bảo sức khoẻ cho sinh viên nên phải phân công kỹ sư trực sao cho đến giờ thứ N+1 thì tất cả lò sưởi đều đang ở trạng thái bật !

Yêu cầu : Bạn được thuê giúp đỡ Ban Giám Hiệu của trường ! Hãy chỉ ra xem những kỹ sư phải được chỉ định là trực để đáp ứng yêu cầu của ban giám hiệu là những kỹ sư nào ! Nếu có nhiều phương án thì cho biết phương án mà số kỹ sư bị chỉ định là ít nhất.

Giới hạn : + $1 \leq N \leq 200$.

+ Time limit 2 s , bộ nhớ 200 KB

Input

- Dòng 1 ghi số nguyên N
- N dòng tiếp theo có cấu trúc gồm 1 số Li (số lò sưởi mà kỹ sư thứ i điều khiển) và tiếp sau đó là Li số là chỉ số của các lò sưởI mà kỹ sư i điều khiển)

Output

- Nếu không có phương án thì ghi ra 1 dòng duy nhất : “No solution”. Ngược lại :
- Dòng 1 ghi số K là số kỹ sư tối thiểu được chỉ định
- Dòng 2 gồm K số là chỉ số của các kỹ sư được chỉ định .

Ví dụ :

Input

4

2 1 2

3 2 3 4

1 2

1 4

Output

3

1 2 3

---> Cách giải của tôi: QHĐ f[i][s]= min(f[i+1][s], f[i+1][s^a[i]]+1).

Thuật giải :

Đây là một bài giải hệ phương trình nhị phân khá đơn giản. Ta có thể sử dụng khử Gauss để giải hệ . Nghiệm có số phần tử bằng = 1 ít nhất chính là phương án sử dụng ít kỹ sư nhất ! Cấp độ của khử Gauss vào khoảng $O(N^3)$ là chấp nhận được.

---> Khử Gauss thực ra là giải hệ phương trình tuyến tính.

Bài 4 : Simple Calculations

Có một dãy số thực gồm N+2 phần tử A_0 , A_1, … , A_{n+1}. Dãy này có tính chất đặc biệt là A[i] = (A[i-1] + A[i+1]) / 2 – C[i] với i = 1,2,3…,n.

Yêu cầu : Cho dãy C và A_0 , A_{n+1}. Hãy tính A_1.

Giới hạn : + $3 \leq N \leq 3000$.

+ $-1000 \leq A_i , C_i \leq 1000$.

+ Time limit 1 s , bộ nhớ 200 KB .

Input

N

A_0 A_{n+1}

C_1

C_2

…

C_n

Output

A_1 (Ghi chính xác đến 2 chữ số sau dấu phẩy) .

Ví dụ :

Input

1

50.50 25.50

10.15

Output

27.85

---> Cách giải của tôi: Lập 3 pt của A1, An, An-1 dựa vào cộng vế. Dpt O(n)

Thuật giải :

Ta có A(1) = (A(0) + A(2)) / 2 – C(1)

A(2) = (A(1) + A(3)) / 2 – C(2)

….

A(n) = (A(n-1) + A(n+1)) / 2 – C(n)

→ A(1) + A(2) + … A(n) = (A(0) + A(n+1) + A(1) + A(n)) / 2 + A(2) + A(3) + …+A(n-1) – (C(1) + C(2) + C(3) +…+C(n)) .

⇔ A(1) + A(n) = A(0) + A(n+1) – 2*(C(1) + C(2) + C(3) +…+C(n)) .

Tương tự ta có : A(1) + A(n-1) = A(0) + A(n) – 2*(C(1) + C(2) + C(3) +…+C(n-1)) .

……..

A(1) + A(1) = A(0) + A(2) – 2* C(1) .

→ A(1) * (n+1) = A(0) * n + A(n+1) – 2 *(C(1) + C(2) + C(3) +…+C(n))

- 2*(C(1) + C(2) + C(3) +…+C(n-1)) – …- 2*C(1).

➔ Ta tính được A(1). Cấp độ thuật toán O(n). Cũng với phương pháp này ta có thể tính được tất cả các phần tử của dãy A cũng chỉ với cấp độ O(n).

∆ Bài 5 : Nikifor – 3

Nikifor có một số tự nhiên N. Anh ta là một người rất yêu thích con số 7 (giống mình). Bởi vậy con số của anh ta phải là một con số chia hết cho 7. Nhưng rất tiếc con số N của anh ta lại chưa phải là bội của 7 . Bây giờ anh ta muốn đổi vị trí của các chữ số của số N sao cho tạo được một số mới mà số này chia hết cho 7. Vì Nikifor là một người rất đặc biệt nên con số N của anh ta cũng rất đặc biệt. Số N này luôn có đủ 4 chữ số 1 , 2 , 3 4 trong biểu diễn thập phân của mình ! Hơn nữa nếu số N này đã là bội của 7 rồi thì anh ta vẫn muốn biến đổi để ra được 1 số mới khác cũng chia hết cho 7.

Yêu cầu : Hãy giúp Nikifor đổi vị trí của các chữ số trong số N sao cho được số mới chia hết cho 7.

Giới hạn : + số N này có không quá 1000 chữ số và có ít nhất 4 chữ số .

+ Time limit : 1s , bộ nhớ 200KB.

Input

- Dòng 1 : 1 số X là số bộ test. ($1 \le X \le 10000$).
- X dòng tiếp theo mỗi dòng ghi 1 số N.

Output

N dòng ghi ra kết quả của từng test , nếu test nào vô nghiệm thì ghi ra “No solution” ngược lại ghi ra số N sau khi biến đổi .

Ví dụ :

Input

2

10234

531234

Output

32410

353241

Thuật giải :

Nhận thấy số N luôn có đủ 4 chữ số 1,2,3,4 . Với 4 số này ta có thể tạo thành bất kỳ số dư nào trong các số dư từ 0->6 :

Ví dụ : 1234 mod 7 = 2 , 3241 mod 7 = 0 …

Như vậy ta có một cách làm rất hay cho bài này đó là : đặt tuỳ ý các chữ số # 0 vào đầu sao cho còn dư lại 4 chữ số 1 , 2 , 3 , 4 . Với 4 chữ số này ta sẽ đặt vào tiếp theo , sau cùng mới đặt các chữ số 0. 4 chữ số 1,2,3,4 cuối ta sẽ duyệt hoán vị của nó xem hoán vị nào của nó thì số N tạo được sẽ chia hết cho 7. Vì với mỗi số dư R đều có ít nhất 2 hoán vị nên dù N cũng có dạng như trên thì cũng có ít nhất 1 số khác cũng cùng dạng này nữa -> Bài toán không bao giờ vô nghiệm ! Vì 4! = 24 -> Chạy rất nhanh !

Bài 6 : Pinocchio

Cha của Pinocchio muốn làm lại cho Pinocchio một cái mũi mới . Ông có N thanh gỗ , mỗi thanh gỗ có độ dài A_i. Là người yêu thích toán học ông ta đưa ra một giải thuật sau để lấy ra thanh gỗ có độ dài cần thiết :

- Nếu còn lại 1 thanh gỗ thì ông ta sẽ lấy thanh gỗ này làm mũi cho Pinocchio.
- Nếu còn >= 2 thanh gỗ thì ông ta sẽ làm như sau :
  - Chọn ra 2 thanh gỗ i , j sao cho A_i và A_j có độ dài nhỏ nhất trong số N thanh.
  - Nếu $A_i = A_j$ thì vứt bỏ một thanh đi, lại quay về bước 1
  - Nếu Ai < A_j thì ta sẽ cắt thanh A_j đi một đoạn = Ai (tức là $A_j = A_j$ – Ai).

Quay lại bước 1.

Yêu cầu : Hãy tính xem độ dài mà thanh gỗ ông ta nhận được sẽ là bao nhiêu.

Giới hạn : + $1 \le N \le 100000$.

+ $1 \le A_i \le 10^9$.

+ Time limit 1s , bộ nhớ 200 KB.

Input

N

A_1

A_2

…

A_n

Output

Gồm 1 số duy nhất X là độ dài thanh gỗ đạt được.

Ví dụ:

Input

3

2

3

4

Output

1

Thuật giải :

Dễ dàng CM được đó chính là ước chung lớn nhất của N số . Ta có thể dùng thuật toán Oclit. Cấp độ tính toán của bài này là O(n).

Bài 7 : Button

Trong đại hội thể thao Olympic năm 3000, người ta quyết định đưa môn bốc sỏi vào thi đấu. Trận đấu sẽ gồm 2 người thi đấu và có K viên sỏi , đến lượt người nào chơi thì được bốc không quá L viên ! Ai đến lượt mình mà không còn gì để bốc nữa thì sẽ thua ! Đấu thủ thứ nhất được quyền chọn xem số K sẽ là bao nhiêu ! Còn đấu thủ thứ 2 được quyền chọn xem số L sẽ là bao nhiêu ! Giả sử đã biết được người thứ

nhất chọn số K là bao nhiêu ,bạn hãy giúp người thứ 2 chọn ra số L nhỏ nhất sao cho người thứ 2 chắc chắn giành phần thắng và số L > 1.

Giới hạn : + $2 < K < 10^9$.

+ Time limit 1s , bộ nhớ 200KB.

Input

K

Output

L

Ví dụ :

Input

6

Output

2

Thuật giải :

Đây là một bài toán trò chơi cổ điển, 1 số L để người 2 chắc chắn thắng thì K phải chia hết cho (L+1). Vậy thì đó sẽ là ước nhỏ nhất của K (mà > 2) – 1. Khi tìm ước cần chú ý đến 1 số trường hợp đặc biệt là 14 chẳng hạn , kết quả trả ra sẽ là 6 nhưng mà do không cẩn thận (có thể là do tìm ước = hàm kiểm tra nguyên tố lấy Sqrt trong khi Sqrt(14) = 3.7416 nên sẽ dẫn tới sai xót !).

Bài 8 : Combinations

Yêu cầu : Rất đơn giản ! Hãy tính xem C có bao nhiêu ước nguyên tố với C được tính = CT :

$C = N! / (((N-M)!) * (M!))$

Tất nhiên ở đây N và M là 2 số cho trước .

Giới hạn : + $1 \le M \le N \le 60000$.

+ Time limit 1s , Bộ nhớ 200KB.

Input

N M

Output

Gồm 1 số duy nhất là số ước của C.

Ví dụ :

Input

7 3

Output

2

(Giải thích : C = 35 = 5*7).

Thuật giải :

Ta phân tích N! , M! thành tích của các số nguyên tố .

$\rightarrow$ $N! = 2^{x1} * 3^{x2} * \dots$

$M! = 2^{y1} * 3^{y2} * \dots$

$(N-M)! = 2^{z1} * 3^{z2} * \dots$

$\rightarrow C = 2^{x1-y1-z1} * 3^{x2-y2-z2} * \dots.$

Ta chỉ việc đếm thôi , với xi-yi-zi > 0 thì ta tăng số phần tử tìm thấy lên 1. Thuật toán hoàn toàn rất đơn giản. Tuy nhiên để thuật toán chạy nhanh ta nên áp dụng công thức Lagrăng như sau :

Số mũ của số nguyên tố P trong N! = $[N/P] + [N/(P^2)] + \dots [N/(P^k)]$.

với [q] = phần nguyên của số q.

Bài 9 : Fibonacci sequence

Có một dãy số dài vô hạn thoả mãn tính chất Fibonacci .

Đó là : F[i] = F[i-1] + F[i-2]. (Với mọi i thuộc Z).

Yêu cầu : Cho biết F[i] và F[j] . Hãy tính F[n].

Giới hạn : + -1000 ≤ i ≠ j , n ≤ 1000.

+ -MaxLongInt ≤ F[i] , F[j] , F[n] ≤ MaxLongInt.

+ Time limit 1s , bộ nhớ 200KB.

Input

i F[i] j F[j] n

Output

F[n]

Ví dụ :

Input

3 5 –1 4 5

Output

12

Thuật giải :

Đối với bài này có 2 cách làm . Giả sử i < j .

Cách 1 : Duyệt nhị phân giá trị của F[i+1] . Với mỗi giá trị của F[i+1] ta sẽ kiểm tra xem số F[j] có đúng với đề bài không ! Nếu đúng thì dừng lại và sinh F[n]. Biết F[i] , F[i+1] thì rõ ràng ta có thể tính được toàn bộ phần tử thuộc dãy.

Cách 2 : Dựa vào CT : --> cách này phải dùng bignum, khó khăn!

F[i+1] = 1 * F[i-1] + 1 * F[i] ;

F[i+2] = 1 * F[i-1] + 2 * F[i] ;

F[i+3] = 2 * F[i-1] + 3 * F[i] ;

F[i+4] = 3 * F[i-1] + 5 * F[i] ;

F[i+5] = 5 * F[i-1] + 8 * F[i] ;

…

F[j] = x * F[i-1] + y * F[i].

Nếu để ý ta sẽ thấy hệ số của F[i-1] , F[i] chính là một dãy Fibonacci . Vậy thì ở đây công việc còn lại sẽ rất đơn giản. Ta chỉ việc tính xem x , y là bao nhiêu nữa là xong ! Chú ý một chút nho nhỏ nếu dùng Cách 2 đó là x, y có thể là rất lớn nên khai báo kiểu Extended là tốt nhất.

Bài 10 : SKBJunk 307

SKBJunk 307 là rô bốt đời mới nhất của trung tâm nghiên cứu vũ trụ VNSC (Viet Nam Space Center) . Nó được trang bị khả năng tính toán siêu việt. An là một người thích nổi tiếng, để cho mọi người thấy SKBJunk 307 tính toán còn chậm hơn cả một máy tính thông thường, anh ta đã thách đấu với nó. Nội dung của cuộc thi là tính xem $1^N + 2^N + 3^N + 4^N$ có bao nhiêu chữ số 0 ở cuối cùng. An biết chắc là sẽ thua nên đã thuê bạn giúp anh ta.

Yêu cầu : Hãy lập trình tính xem $1^N + 2^N + 3^N + 4^N$ có bao nhiêu chữ số 0 ở cuối cùng.

Giới hạn : + 1 ≤ N ≤ 300000.

+ Time limit 1 s , bộ nhớ 200KB.

Input

N

Output

Gồm 1 số X duy nhất là kết quả tính được.

Ví dụ:

Input

1

Output

1

Thuật giải :

Dễ thấy kết quả trả ra luôn chỉ có thể là 1 trong 3 số : 0 , 1 , 2.

Kết quả của bài toán này đã được CM bởi Euler đó là :

X(N) = X(N mod 2000).

Bởi vậy thay vì tính số mũ từ 1-> N ta chỉ phải tính số mũ từ 1-> N mod 2000 mà thôi. Tuy nhiên trong thực tế (với N <= 300000) thì ta chỉ cần tính số mũ từ 1 -> N mod 100 mà thôi.

THUẬT TOÁN SỐ HỌC

Tác giả: Trần quang Khải

I. Giới thiệu chung

Số học thuật toán (Algorithmic number theory) là một ngành toán học đang phát triển mạnh, nghiên cứu số học trên phương diện thuật toán. Ta cần chú ý rằng Số học là một trong những ngành toán học cổ nhất còn thuật toán lại là một khái niệm mới mẻ ra đời và phát triển từ trong thế kỉ XX. Số học thuật toán được xây dựng trên cơ sở những thành tựu của cả lý thuyết số học lẫn thuật toán.

Một trong những bài toán nổi tiếng nhất đã đựoc giải quyết trong thế kỉ XX là bài toán thứ 10 của Hilbert '*Có tồn tại một thuật toán tổng quát cho phép ta trả lời một phương trình Diophantine cho trước có nghiệm hay không ?*'. Phương trình Diophantine là phương trình có dạng f(x, y, z, …) = 0 trong đó f(x, y, z, …) là đa thức của các biến x, y, z, … có các hệ số nguyên, và các biến chỉ nhận giá trị nguyên. Bài toán này đã được Michiakêvích giải quyết trọn vẹn và câu trả lời là phủ định, tức là không có thuật toán như vậy. Bài toán thứ 10 của Hilbert được giải quyết là một thành tựu quan trọng của số học cũng như thuật toán.

Số học thuật toán không chỉ phát triển trên những thành tựu của Số học sơ cấp, mà nó còn tận dụng những thành tựu của Số học hiện đại, Đại số hiện đại, Hình học đại số … Cũng như các ngành toán học - thuật toán khác, trong Số học thuật toán cũng có những bài toán NP, tức là không có thuật giải trong thời gian đa thức, tiêu biểu là bài toán phân tích một số ra các thừa số nguyên tố, thuật toán kiểm tra nguyên tố, …

Trong bài viết này, ta chỉ đề cập tới những vấn đề cơ bản nhất của Số học thuật toán, và trên cơ sở của toán học sơ cấp.

II. Các phép tính với số nguyên

Trong máy tính, các số đều được biểu diễn ở hệ nhị phân, hơn nữa việc mô tả dưới dạng nhị phân làm cho các phép tính trở nên đơn giản hơn rất nhiều.

Để cho tổng quát, ta giả sử hai toán hạng đều có đúng N bit (trong trường hợp số các bít có nghĩa nhỏ hơn N thì ta thêm 0 vào đầu cho đủ).

1. Phép cộng và phép trừ

Đối với phép cộng và phép nhân các số N bít, ta có thể thực hiện giống như phép cộng trừ trên hệ thập phân : Thực hiện cộng (hay trừ) từ phải sang trái (với bít ứng với số mũ nhỏ trước rồi số mũ lớn sau).

1. Phép nhân

Đối với phép nhân, ta có thuật toán nhân Nga được mô tả như sau :

```
Function Mult(a, b : Integer): Integer;
Var
   c : Integer;
Begin
  c := 0;
  repeat
```

```
    if Odd(b) then c := c + a;
    a := a shl 1;
    b := b shr 1;
  until b = 0;
  Mult := c;
End;
```

Nếu xét cho kĩ thì thực ra thuật toán nhân Nga có cùng một dạng với thuật toán nhân hai số bằng phương pháp mà bình thường ta vẫn dùng để tính tay, chỉ có điều khác là thao tác trên hệ nhị phân.

Thuật toán nhân Nga gồm N lần dịch bít, trong trường hợp tổng quát phải thực hiện N/2 lần phép cộng, do đó độ phức tạp tính toán là N^2.

Sau đây, ta sẽ giới thiệu một thuật toán nhân khác, tuy phức tạp hơn nhưng có độ phức tạp nhỏ hơn.

Giả sử ta phải thực hiện phép nhân với hai số có 2N bit A và B. Phân tích :

$A = a1*2^N + a2$

$B = b1*2^N + b2.$

$AB = a1*b1 * 2^{2N} + (a1b2 + a2b1)*2^N + a2b2.$

Ta chú ý rằng phép nhân với một luỹ thừa của 2 cũng như phép cộng có thời gian tỉ lệ với N.

Nhận xét : (a1+a2)(b1 + b2) - a1b1 - a2b2 = a1b2 + a2b1.

Do đó nếu biết (a1+a2)(b1 + b2), a1b1, a2b2 thì có thể tính được a1b2 + a2b1.

$\rightarrow$ Qui về nhân hai số 2N bit về 2 lần nhân N bit và một số phép tính có thời gian tỉ lệ với N.

Nếu gọi F(n) là thời gian nhiều nhất để thực hiện phép nhân hai số có 2^n bit, ta có

$F(n) = 3F(n-1) + k2^n$ (*) trong đó k là một hằng số thể hiện chi phí các phép tính cộng và dịch bit trong mỗi bước gọi đệ qui. Chia cả hai vế của (*) cho 2^n ta được

. Do đó, F(n) = O(3^n) = O( ) hay nói cách khác thời gian thực

hiện phép nhân hai số N bít có độ phức tạp cỡ O( ).

3. Phép chia

Trong phần này, ta sẽ trình bày thuật toán chia hai số nhị phân có N bít $A = (A1A2A3…AN)_2$, $B = (B1B2…BN)_2$ cho kết quả thương Q, dư R.

+Bước 1:$Q \leftarrow 0$, $R \leftarrow 0$, $i \leftarrow 0$

+Bước 2: $i = i+1$. Nếu $i > N \rightarrow$ kết thúc thuật toán, ngược lại $\rightarrow$ Bước 3.

+Bước 3: $R \leftarrow$ (R shl 1) or (A[i]). Nếu $R \geq B \rightarrow$ Bước 4, ngược lại Bước 5.

+Bước 4: $R \leftarrow R - B$, C[i] = 1. Thực hiện Bước 2.

+Bước 5: C[i] = 0. Thực hiện Bước 2.

Hay ta có thể mô tả bằng chương trình :

```
Procedure Divide(A, B : LongInt; var Q, R : LongInt);
Var
  i : Integer;
Begin
   Q := 0;
   R := 0;
   For i := 1 to N do
   Begin
      If (A and (1 shl (N-i)) <> 0) then R := (R shl 1)+1
      else R := R shl 1;
      if R >= B then
      Begin
```

```
        Q := Q or (1 shl (N-i));
        R := R - B;
      End;
   End;
End;
```

Đánh giá độ phức tạp: Thuật toán chia có độ phức tạp (N^2).

III. Thuật toán Euclid

1. Thuật toán Euclid

Thuật toán Euclid dùng để tìm ước chung lớn nhất của hai số nguyên dương m, n.

Thuật toán tiến hành như sau :

Bước 0: Đặt a = m, b = n.

Bước 1: Nếu a > b ta thay a bằng phần dư của phép chia a cho b, ngược lại thay b bằng phần dư của phép chia b cho a.

Bước 2: Nếu a hoặc b bằng 0, thì kết quả là a + b, ngược lại thực hiện bước 1.

Hay thuật toán được mô tả bằng một hàm trong ngôn ngữ Pascal như sau :

```
Function Euclid(m, n : Integer) : Integer;
Var
   a, b : Integer;
Begin
  a := m;  b := n;
  repeat
    if a > b then a := a mod b
   else b := b mod a;
  until (a = 0) or (b = 0);
 Euclid := a + b;
End;
```

Ta cũng có cách viết khác như sau :

```
 Function Euclid1(m, n : Integer) : Integer;
Var
   a, b, r : Integer;
Begin
  a := m;  b := n;
  repeat
    r := a mod b;
   a := b;
   b := r;
  until b = 0;
 Euclid := a;
End;
```

Trong số học ta đã biết rằng với hai số nguyên dương m, n bất kì và d là ước chung của chúng thì luôn tồn tại hai số nguyên u, v sao cho u*m + v*n = d. Từ thuật toán Euclid ta cũng có thể chỉ ra được cụ thế một cặp (u, v) như vậy :

Ta chú ý rằng tại mỗi bước của vòng lặp repeat, a và b đều tồn tại cặp (x, y) với x, y $\in$ Z thoả mãn x*m + y*n = a (= b). Ban đầu a = m có một cặp tương ứng là (1, 0), cặp tương ứng với b = n là (0, 1) vì :

1*m + 0*n = m = a

0*m + 1*n = n = b

Trong vòng lặp, ta có phép gán a := a mod b, phép gán này tương đương a := a - q*b trong đó q là thương của phép chia a cho b (q := a div b). Khi đó cặp (x_a, y_a) tương ứng với a biến đổi tương ứng

$x_a := x_a - q*x_b$

$y_a := y_a - q*y_b$

Như vậy, a và b luôn có thể biểu diễn dưới dạng x*m + y*n và kết quả ước chung bằng a + b sẽ có cặp x, y tương ứng là $x_a + x_b$, $y_a + y_b$. Như vậy, thuật toán tìm cặp u, v trên cơ sở thuật toán Euclid sẽ được viết như sau :

```
Procedure Euclid2(m, n : Integer; var u, v : Integer);
Var
  a, b, q, r : Integer;
  xa, ya, xb, yb, xt, yt : Integer;
Begin
   a := m;
   b := n;
   xa := 1;
   ya := 0;
   xb := 0;
   yb := 1;
   repeat
      q := a div b;
      r := a mod b;
      a := b;
      b := r;
      xt := xa;
      yt := ya;
      xa := xb;
      ya := yb;
      xb := xt - q*xb;
      yb := yt - q*yb;
   until b = 0;
   u := xa;
   v := ya;
End;
```

Đánh giá độ phức tạp của thuật toán Euclid : Nếu ta gọi (a_i, b_i) là cặp a, b ở vòng lặp thứ i, đánh số theo chiều ngược lại (kết thúc là (a_1, b_1), trước đó là (a_2, b_2), (a_3, b_3) … , (a_s, b_s)). Ta có thể chứng minh bằng qui nạp $\max\{a_k, b_k\} \geq F_k$ trong đó F_k là số thứ k trong dãy Fibonaci. Ta có công thức sau :

$F_n =$

F_n tăng theo hàm mũ, thuật toán Euclid có độ phức tạp lôgarit của max(m, n).

Ta chú ý rằng cách đánh giá này bỏ qua thời gian thực hiện phép chia (mod) (trong máy tính đây là một lệnh của CPU), tuy nhiên thuật toán chia thực hiện với hai số N bít nói chung mất khoảng N^2. Do đó nếu xét chi tiết, thuật toán Euclid có độ phức tạp cỡ $O((\log(\max\{m, n\})^3)$.

2. Thuật toán Euclid nhị phân

Sau đây, ta sẽ trình bày một dạng Quay về Bước 1.

+ Bước 3 : Chừng nào khác của thuật toán Euclid có độ phức tạp nhỏ hơn. Ta có nhận xét sau : ước chung lẻ lớn nhất của hai số a, b không phụ thuộc vào các phép biến đổi sau :

+ Chia a (hoặc b) nếu a (hoặc b) chẵn.

+ Thay a bằng a - b.

Thuật toán có thể mô tả như sau :

+ Bước 0 : a = m, b = n, dem = 0.

+ Bước 1 : Nếu cả a và b đều chẵn thì → Bước 2 ngược lại → Bước 3.

+ Bước 2 : dem = dem + 1, chia cả a và b cho 2, a còn chẵn thì chia a cho 2, thực hiện tương tự với b.

+ Bước 4 : Nếu a = b → Bước 6, ngược lại → Bước 5.;

+ Bước 5 : Nếu a > b → a := a - b, ngược lại → b := b - a. Thực hiện Bước 3.

+ Bước 6 : c := a * 2^{dem} (dịch c sanh trái số bit bằng dem) Thuật toán kết thúc và c là ước chung lớn nhất cần tìm.

Ta có thể môt tả bằng chương trình Pascal như sau :

```
Function BinaryEuclid(m, n : Integer) : Integer;
Var
  a, b, dem : Integer;
Begin
   a := m;
   b := n;
   dem := 0;
   while (not Odd(a)) and (not Odd(b)) do
   begin
     Inc(dem);
     a := a shr 1;
     b := b shr 1;
   end;
   repeat
      while not Odd(a) do a := a shr 1;
      while not Odd(b) do b := b shr 1;
      if a = b then Break;
      if a > b then a := a - b
      else b := b - a;
   until a = b;
   BinaryEuclid := a shl dem;
End;
```

Vì các số đước mô tả dưới dạng nhị phân nên rõ ràng các phép chia 2 có thể thực hiện đơn giản bằng phép dịch phải. Nếu m, n có thể mô tả bằng N bít thì phép dịch có chi phí bằng N, phép trừ cũng có chi phí bằng N (xem phần II). Mặt khác, sau mỗi phép trừ là ít nhất một phép dịch bít và ta dễ thấy có không quá 2N lần dịch bit. Do đó độ phức tạp của thuật toán cỡ N^2hay $\log(\max\{m, n\})^2$.

Ta cũng cần chú ý rằng phép mod là một lệnh của hệ thống nên đối với các số m, n không quá lớn (trong phạm vi 2^{31}) thì thuật toán Euclid nếu ở mục 1 chạy nhanh hơn thuật toán mô tả ở đây. Còn khi ta phải làm việc với các số lớn thì thuật toán Euclid nhị phân thích hợp hơn. Thuật toán này không chỉ có độ phức tạp nhỏ hơn mà có một lợi thế khác là ta chỉ phải thực hiện phép dịch bít và phép trừ là những phép tính rất đơn giản, trong khi thuật toán Euclid phải thực hiện phép chia viết khá phức tạp.

IV. Giải phương trình nghiệm nguyên tuyến tính_

1. Phương trình đồng dư ax ≡ b(mod c) (*)

Với a, b, c đều là các số nguyên dương.

Gọi d là ước chung lớn nhất của a và c. Kết quả từ số học cho ta phương trình có nghiệm khi và chỉ khi b chia hết d và nếu (*) có nghiệm thì sẽ có vô số nghiệm dạng x = x_0 + k * e với k $\in$Z, e = c / d và x_0 là một nghiệm của (*). Như vậy, ta có thể dễ dàng kiểm tra phương trình có nhiệm hay không.

Vấn đề còn lại là tìm một nghiệm x_0 thoả mãn phương trình. Ta chú ý rằng thuật toán Euclid có thể chỉ ra hai số u, v sao cho u *a + v * c = d, tức là u*a $\equiv$ d (mod c)$\Rightarrow$ nếu x_0 = u * (b / d) (chú ý phương trình có nghiệm $\Leftrightarrow$ b chia hết cho d) thì a*x_0 $\equiv$ b (mod c), tức là x_0 thoả mãn (*).

Chương trình Pascal giải phương trình (*) như sau :

```
Procedure Solve1(a, b, c : Integer);
Var
  d, e, b1, u, v : Integer;
Begin
   Euclid2(a, c, u, v);
   d := u*a + v*c;
   if b mod d <> 0 then Writeln('No solution ')
   else
    begin
      e := c div d;
      b1 := b div d;
      Writeln('Equation has infinite solution');
      Write('All solutions have form : );
      Writeln(' x = ', u*b1 ,' + k*', e);
    end;
End;
```

2. **Phương trình Diophantine tuyến tính dạng a*x + b*y = c**. (**)

với a, b, c $\in$ Z^+

Ta có thể dễ thấy phương trình Diophantine này có thể đưa về giải phương trình đồng dư tuyến tính.

(**) $\Leftrightarrow$

Việc giải phương trình đồng dư tuyến tính đã được trình bày ở mục 1.

Chú ý :

Gọi d = UCLN(a, b)

a_1 = a / d

b_1 = b / d

(**) có nghiệm khi và chỉ khi c $\equiv$ 0 (mod d).

(**) nếu có nghiệm thì sẽ có vô số nghiệm, công thức nghiệm :

(x, y) = (x_0, y_0) + k (-b_1, a_1).

Với (x_0, y_0) là một nghiệm nào đó của (**).

3. **Định lý đồng dư Trung Hoa**.

Định lý : Cho m số nguyên dương a_1, a_2, a_3, … a_m đôi một nguyên tố cùng nhau, và m số nguyên b_1, b_2, b_3, … b_m thoả mãn 0 $\le$ b_i $\le$ a_i - 1 $\forall$i= 1, 2, …, m.

Đặt M = a_1*a_2*…*a_m. Khi đó tồn tại và duy nhất số nguyên x thoả mãn :

0 $\le$ x $\le$ M - 1 và

x $\equiv$ b_i (mod a_i) $\forall$i = 1, 2, …, m.

Định lý đồng dư Trung Hoa được trình bày và chứng minh trong hầu hết các giáo trình Số học. Sau đây ta sẽ tìm hiểu thuật toán chỉ ra cụ thể số x như vậy.

Ta chú ý rằng nghiệm tương ứng với k ràng buộc đầu (ứng với các số a_1, b_1, a_2, b_2, … a_k, b_k) sẽ có dạng $x = x_k + M_k$. Trong đó $M_k = a_1 * a_2 * \ldots * a_k$ và x_k là nghiệm thoả mãn định lý đồng dư Trung Hoa tương ứng với a_1, b_1, a_2, b_2, … a_k, b_k.

Dễ thấy khi đó x_{k+1} là nghiệm không âm nhỏ nhất thoả mãn hệ

$x \equiv b_{k+1} (\text{mod } a_{k+1})$

$x \equiv x_k \pmod{M_k}$.

Kết quả cuối cùng, số thỏa mãn bài toán là x_m.

Như vậy, ta đưa việc giải m phương trình về m-1 lần giải hệ hai phương trình.

Ta giải hệ hai phương trình như sau :

Vì $x \equiv x_k \pmod{M_k}$ nên $x = x_k + y * M_k$, bài toán trở thành tìm nghiệm không âm nhỏ nhất của phương trình đồng dư $x_k + y * M_k \equiv b_{k+1} (\text{mod } a_{k+1})$. Ở phần 1, ta đã giải phương trình này cho nhiệm có dạng $y = y_0 + t * a_{k+1}$ (vì a_{k+1} và M_k nguyên tố cùng nhau). Từ công thức nghiệm, dễ thấy y không âm nhỏ nhất = $y_0 \bmod a_{k+1}$.

V. Thuật toán kiểm tra nguyên tố

Bài toán : Kiểm tra một số n > 1 cho trước có phải là nguyên tố hay không ?

1. *Tiếp cận bài toán qua một số thuật toán đơn giản*

Theo đúng định nghĩa số nguyên tố: Một số là nguyên tố khi và chỉ khi nó không có các ước không tầm thường. Từ đó ta có thể viết một hàm kiểm tra nguyên tố như sau :

```
Function Prime(n : Integer) : Boolean;
Var
  i : Integer;
Begin
  Prime := False;
  For i := 2 to n-1 do
   if n mod i = 0 then Exit;
  Prime := True;
End;
```

Ta có thể thấy rõ thuật toán trên chi phí trong trường hợp xấu nhất lên tới O(n). Ta có thể thấy nó rất thô dựa vào nhận xét : Nếu n là hợp số thì nó phải có một ước nhỏ hơn , và do đó ta chỉ cần xét các ước không quá của nó. Thuật toán cải tiến sẽ được mô tả như sau :

```
Function Prime(n : Integer) : Boolean;
Var
  i : Integer;
Begin
  Prime := False;
  For i := 2 to Trunc(Sqrt(n)) do
   if n mod i = 0 then Exit;
  Prime := True;
End;
```

Dễ thấy thuật toán cải tiến trong trường hợp tồi nhất mất O( ), tuy nhiên ta cũng cần chú ý rằng thuật toán cải tiến chỉ khác thuật toán ban đầu khi n là số nguyên tố, còn với n là hợp số thì hai thuật toán kết thúc sau cùng một số phép tính.

Ta có thể có một hướng cải tiến nữa nếu nhận xét thêm rằng : Nếu n là hợp số thì phải có một ước nguyên tố ≤ , do đó thay vì tìm ước trong tất cả các số trong khoảng từ 2 đến , ta chỉ xét các số nguyên tố trong khoảng đó. Nhưng khi đó một vấn đề được đặt ra là làm thế nào để có danh sách

số nguyên tố không vượt quá , có thể thấy chi phí tạo danh sách số nguyên tố sẽ không dưới .
Tuy vậy, trong trường hợp phải kiểm tra rất nhiều số n như thế thì ta có thể áp dụng phương pháp trên. Trong trường hợp không có danh sách các số nguyên tố, ta có thể hạn chế tập tìm kiếm bằng phương pháp sau : Chọn một số k < n, nếu UCLN(n, k) > 1 thì rõ ràng n là hợp số, ngược lại ta hạn chế tập tìm

kiếm các số trong khoảng 2 đến mà phải nguyên tố cùng nhau với k. Thuật toán có thể mô tả như sau :

```
Function Prime(n, k : Integer) : Boolean;
Var
  List  : array[1..maxk] of Integer;
  j, i, l : Integer;
Begin
  Prime := False;
  if Euclid(n, k) > 1 then Exit;  {Hàm Euclid lấy ước chung lớn
                                   nhất của hai số nguyên dương đã được môt tả ở phần III}
  l := 0;
  for i := 1 to k-1 do
  if Euclid(i, k) = 1 then
   begin
      Inc(l);
      List[l] := i;
   end;
    for i := 2 to l do
    if n mod L[i] = 0 then Exit;
    j := k;
    while Sqr(j) < n do
     begin
       for i := 1 to l do
        if n mod (j+List[i]) = 0 then Exit;
       j := j + k;
     end;
    Prime := True;
End;
```

Vì k rất nhỏ nên thời gian tính toán chủ yếu ở phép tìm ước. Trong phần tìm ước, cứ trong k số ta chỉ xét l số. Chú ý rằng ở đây l = φ(k) là số các số tự nhiên không quá k và nguyên tố cùng nhau với k. Độ

phức tạp của thuật toán cỡ O( ).

Theo công thức Euler, nếu k có dạng phân tích tiêu chuẩn k = $p_i^{\alpha_i}$

thì φ(k) = k , tức là = .
Do đó phương pháp chọn k tối ưu nhất là lấy k bằng tích của những số nguyên tố đầu tiên. Nếu chọn k

= 6 thì độ phức tạp là O( ), độ phức tạp của thuật toán giảm đáng kể nếu so với O( ). Tuy nhiên, ta sẽ không thu được một sự giảm đáng kể hơn nếu chọn thêm số nguyên tố mới.

K	Phân tích tiêu chuẩn	
6	2*3	
30	2*3*5	
210	2*3*5*7	

Trong các hướng tiếp cận trên, có một nguyên tắc cơ bản để kiểm tra nguyên tố là tìm một ước không tầm thường của n, và trong trường hợp tồi nhất (khi n là số nguyên tố), thì ta phải xét hết các số nguyên tố không quá .

Nếu gọi $\pi(x)$ là số các số nguyên tố không vượt quá x, người ta đã có kết quả nổi tiếng sau đây :

= 1.

Điều này cho thấy trong trường hợp n là nguyên tố, số phép toán không ít hơn .

Trong thực tế, nếu n lên tới hàng trăm chữ số những thuật toán nêu trên rõ ràng không thể chạy được trong thời gian cho phép. Ta cũng cần chú ý rằng hiện tại chưa có một thuật toán tổng quát cho kết quả chính xác trong thời gian chấp nhận được.

2. *Thuật toán xác suất*

Thuật toán xác xuất tiếp cận bài toán theo hướng khác, không theo con đường tìm một ước không tầm thường mà dựa vào hai tính chất khác của số nguyên tố. Cho p là một số nguyên tố, a là số nguyên không chia hết cho p, ta có :

$a^{p-1} \equiv 1 \pmod p$ (1)

$a^2 \equiv 1 \pmod p \Leftrightarrow a \equiv 1 \pmod p$ hoặc $a \equiv -1 \pmod p$. (2).

Hai tính chất trên khá quen thuộc trong số nguyên tố, ta có thể chứng minh không mấy khó khăn.

Giả sử $p-1 = q2^t$ trong đó q là một số nguyên dương lẻ. Theo (1) và (2), với mọi số a không chia hết cho p ta đều có :

+ $\equiv 1 \pmod p$ hoặc

+ $\equiv -1 \pmod p$ với một số nguyên r nào đó thoả $0 \le r < t$. (*)

Như vậy, một số n là nguyên tố thì nó phải thoả mãn điều kiện trên với mọi $1 \le a \le n-1$, ngược lại nếu tồn tại một a không thoả mãn tính chất trên thì chắc chắn n không phải là số nguyên tố. Mặt khác, ta có thể chứng minh được rằng, nếu n là một hợp số lẻ > 3 thì có không quá (n-1) / 4 giá trị a trong khoảng 1 đến n-1 thoả mãn (*). Do đó, với một số a chọn ngẫu nhiên trong khoảng 1 đến n-1, thì xác suất a thoả mãn (*) không quá , và nếu k lần chọn thì xác suất để mọi lần chọn đều thoả mãn là không quá . Từ đó, người ta đã đưa ra một thuật toán xác suất cho phép kiểm tra một số n có phải nguyên tố hay không trong thời gian đa thức với độ chính xác rất cao.

Thuật toán (Rabin - Miller) có thể được mô tả như sau :

```
Function Prime(n : Integer) : Boolean;
Var
  i, a : Integer;
Begin
   Prime := False;
   Cal(n, q, t)                    {phân tích n-1 = q2^t}
   For i := 1 to k do
   Begin
     a := Random(n-1) + 1;
     if MillerTest(n, a, q, t) = False then Exit;
         {Kiểm tra với một cơ sở a được chọn ngẫu nhiên}
   End;
   Prime := True;
End;
```

Thủ tục Cal(n, q, t) phân tích $n-1 = q2^t$ với q là số nguyên lẻ khá đơn giản ta nên ta không trình bày ở đây.

Phần quan trọng nhất là kiểm tra a có thoả mãn tính chất (*) hay không.

Trước hết, ta phải tính $a^q \bmod n$, nếu ta tính bằng thực hiện tuần tự các phép nhân q lần thì chi phí sẽ rất lớn (q có thể tỉ lệ với n). Vì ta chỉ cần tính luỹ thừa modulo n nên ta có một phương pháp rất hay : Phương pháp bình phương liên tiếp. Ý tưởng cơ bản của nó như sau : phân tích q dạnh nhị phân, q sẽ có dạng tổng của các luỹ thừa của 2.

Ta xây dựng dãy {u} như sau :

$u_0 = a$

$u_{k+1} = u_k^2 \bmod n \ \forall \ k \in N$.

Khi đó, ta dễ dàng chứng minh $u_k =$ mod n.

Khi đó, $a^q \bmod n$ sẽ được tính bằng tích theo modulo n của các u_k tương ứng với các luỹ thừa của 2 trong phân tích nhị phân của q.

Ta có thể mô tả bằng thủ tục như sau :

```
Function Power(a, q, n : Integer) : Integer;
Var
  c, p2, u : Integer;
Begin
   c := 1;
   u := a;
   p2 := 1;
   repeat
      if q and p2 <> 0 then c := (c * u) mod n;
     p2 := p2 shl 1;
      u := (u * u) mod n;
   until p2 > q;
   Power := c;
End;
```

Ta có thể dễ dàng chứng minh việc tính $a^q \bmod n$ bằng thủ tục nêu trên phải thực hiện không quá $2\log_2 q$ lần phép nhân modulo n. Do đó chi phí tính toán (phép nhân và phép chia mất chi phí $O((\log n)^2)$) cỡ $O(\log q(\log n)^2)$.

Phép kiểm tra điều kiện (*) có thể được viết như sau :

```
Function MillerTest(n, a, q, t : Integer) : Boolean;
```

```
Var
   pre, c, i, j : Integer;
Begin
   c := Power(a, q, n);
   pre := n-1;
   for i := 0 to t do
   begin
      if c = 1 then
      begin
         MillerTest := (pre = n-1);
         Exit;
      end;
      pre := c;
      c := (c * c) mod n;
   end;
   MillerTest := False;
End;
```

Thủ tục MillerTest gồm phần tính $a^q \bmod n$ có chi phí $O(\log q(\log n)^2)$ và t lần thực hiện tính bình phương modulo n có chi phí cỡ $O(t(\log n)^2)$. Ta có $n-1 = q2^t$ nên độ phức tạp của một lần kiểm tra cơ sở a có chí phí cỡ $O((\log n)^3)$.

Thuật toán thực hiện k lần kiểm tra có chi phí $O(k(\log n)^3)$, xác suất sai không quá . Như vậy, độ phức tạp của thuật toán tăng tuyến tính theo k, xác suất sai lầm giảm theo hàm mũ với k. Chỉ cần chọn k không quá lớn ($k \geq 30$) thì xác suất sai lầm quá nhỏ, do đó nếu n trải qua phép thử với k cơ sở a chọn ngẫu nhiên thì có thể khẳng định gần như chắc chắn n là số nguyên tố.

3. Thuật toán kiểm tra nguyên tố với số Fecmat và số Mersenne.

Số Fecmat $F_n =$ **.** Nhà toán học Fecmat đã đưa ra giả thiết F_n là số nguyên tố với mọi $n \in N$. Ta có thể dễ dàng kiểm tra điều này đúng với n = 0, 1, 2, 3 và 4. Tuy nhiên Euler đã chỉ ra F_5 là hợp số vì nó chia hết cho 641.

Định lý : F_n là số nguyên tố khi và chỉ khi $\equiv -1 \pmod{F_n}$.

Số Mersenne $M_p = 2^p - 1$. Ta có thể dễ thấy nếu M_p là số nguyên tố thì p phải là số nguyên tố. Số Mersenne có ý nghĩa quan trọng trong Số học vì nó liên quan tới số hoàn chỉnh. Một số nguyên dương n gọi là số hoàn chỉnh nếu tổng các ước bé hơn n của n bằng n. Euler đã chứng minh n là số hoàn chỉnh chẵn khi và chỉ khi $n = M_p 2^{p-1}$ với M_p là số nguyên tố.

Định lý : Cho p là một số nguyên tố, dãy $\{L_n\}$ xác định như sau :

$L_0 = 4$

$L_{n+1} = L_n^2 - 2 \quad \forall n \in N$.

M_p là số nguyên tố khi và chỉ khi $L_{p-2} \equiv 0 \pmod{M_p}$.