PHẦN I. CÁC KIẾN THỨC CĂN BẢN

1.Số nguyên tố

1.1 Thuật toán kiểm tra tính nguyên tố

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

```
Function isprime (n:integer):boolean;
Begin
        If n<=1 then exit(false);
        I:=2;
        While (i*i<=n) do
        Begin
            If n mod i=0 then exit(false);
            Inc(i);
        End;
        Exit(true);
End;
```

1.2 Thuật toán Sàng nguyên tố Eratosthenes

```
  procedure eratosthenes(n:integer);
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;
                              inc(j,i);
               end;
               end;
      end;
```

Ta có mảng P, P[i]=true (i>1) nếu I là số nguyên tố và ngược lại.

Thuật toán này chỉ có độ phức tạp là O(n) thế nên ta có thể hoàn toàn thực hiện với $n=10^8$

1.3 Phân tích một số ra thừa số nguyên tố.

```
procedure      phantich(n);
     Begin
              i:=2;  t:=0;
             While (i*i<=n) do
               If (n mod i=0) then
                    Begin
                          x:=0;
                          While (n mod i=0) do
                          Begin
                                n:=n div i;
                                inc(x);
                          end;
                          inc(t);
                          coso[t]:=i; somu[t]:=x;
                          inc(i);
                     end;
             If (n>1) then
                     Begin
                          inc(t);
                           coso[t]:=n; somu[t]:=x;
                     end;
     end;
```

Thông tin được trả về trong mảng coso[i] và somu[i] lần lượt là cơ số và số mũ của thừa số thứ I trong phép phân tích của n ra thừa số nguyên tố. Trong trường hợp xấu nhất, với n là số nguyên tố thì thuật toán chạy trong thời gian là $O(n^{1/2})$.

1.4 Ước lượng số số nguyên tố bé hơn 1 số cho trước

Gọi A(n) là số lượng số nguyên tố bé hơn N. Ta có A(n) ≈ n/ln(n). Kết quả này sẽ giúp chúng ta ước lượng được độ phức tạp cho những bài toán liên quan đến số nguyên tố.

2. Ước số, Bội Số

Giả sử có 2 số nguyên a,b.

$a:=P_1^{a1}.P_2^{a2}.P_3^{a3}\ldots.P_k^{ak}$;

$b:=P_1^{b1}.P_2^{b2}.P_3^{b3}\ldots.P_k^{bk}$. (p1,p2,..,pk là số nguyên tố)

2.1. Ước chung lớn nhất

Ta có đẳng thức : a=bq+r (r=a mod b)

Gọi d là ước chung của a,b thì a ,bq đều chia hết cho d nên r cũng chia hết cho d. Từ đó suy ra mọi ước chung d của (a,b) thì cũng là ước chung của (b,r) . Vậy nên UCLN(a,b)=UCLN(b,r). Ta có thuật toán tìm UCLN dưới dạng đệ quy :

```
Function    UCLN(a,b:longint):longint;
Begin
            If (b=0) then exit(a);
            If (a<b) then exit(UCLN(b,a));
            UCLN:=UCLN(b,a mod b);
End.
```

2.2 Bội số chung nhỏ nhất

$UCLN(a,b)=P_1^{min(a1,b1)}.P_2^{min(a2,b2)}.P_3^{min(a3,b3)}\ldots.P_k^{min(ak,bk)}$.

$BSCNN(a,b)=P_1^{max(a1,b1)}.P_2^{max(a2,b1)}.P_3^{max(a3,b3)}\ldots.P_k^{max(ak,bk)}$.

⇨ UCLN(a,b).BSCNN(a,b) = a.b

⇨ BSCNN(a,b)=

$$\frac{a.b}{UCLN(a,b)}$$

2.3 Số các ước của 1 số

Gọi S(A) là số các ước của số A.

⇨ S(a)=(a1+1)*(a2+1)*(a3+1)*…*(ak+1).

2.4 Tổng các ước số của 1 số

Gọi F(A) là số các ước của số A.

⇨ $F(A)=(P_1^{a1+1}-1)/(P_1-1) * ....*(P_k^{ak+1}-1)/(P_k-1)$.

Hay:

Ta có $= \frac{(a^{i+1}-1)}{a-1} \times \frac{(b^{j+1}-1)}{b-1} \times \ldots \times \frac{(c^{k+1}-1)}{c-1}$ P(n)

III. SÀNG NGUYÊN TỐ VÀ LƯU LẠI SỐ NGUYÊN TỐ BÉ NHẤT LÀ ƯỚC CỦA MỘT SỐ

Code C++:

```
const int maxp=1000000;
int p[maxp+1];
void sieve(){
      for(int i=2; i<=maxp, i++) if(p[i]==0) for(int j=i; j<=maxp; j+=i) if(p[j]==0) p[j]=i;
}
```

IV. DUYỆT CÁC CẶP ƯỚC, BỘI BÉ HƠN N VỚI ĐỘ PHỨC TẠP $O(nlog(n))$

```
int n=1000000;
for(int uoc=1; uoc<=n; uoc++)
     for(int boi=uoc; boi<=n; boi+=uoc)
          xuli(uoc , boi);
```

Độ phức tạp của đoạn code trên là $O(nlog(n))$, có thể chứng minh bằng tích phân. Xem Harmonic series.

V. NÂNG LÊN LŨY THỪA N VỚI ĐỘ PHỨC TẠP $O(\log(n))$

```
int power(int a, int n){
	if(n==0) return 1;
	int t=power(a, n/2);
	t=t*t;
	if(n%2) t=t*a;
	return t;
}
```

VI. TÌM MỘT NGHIỆM BẤT KÌ CỦA PHƯƠNG TRÌNH DIOPHANTINE TUYẾN TÍNH BẰNG GIẢI THUẬT EUCLID MỞ RỘNG

Lưu ý phương trình Diophantine $ax+by=c$ có nghiệm khi $gcd(a,b) \vee c$.

Code C++:

```
int extended_euclid(int a, int b, int &x, int &y){
//giải phương trình ax+by=1 (biết trước gcd(a, b)=1), trả kết quả ra tham biến x và y.
    if(b==0){
        x=1; y=0; //vì gcd(a, b)=1 nên x=1
    }
    else{
        extended_euclid(b, a%b, y, x);
        ///b*y+(a%b)*x=1
        ///(y-(a/b)*x)b+ax=1
        y-=(a/b)*x;
    }
}
int diopantine(int a, int b, int c, int &x, int &y){ ){
//giải phương trình ax+by=c trả kết quả ra tham biến x và y và trả //về 1 nếu có nghiệm hoặc trả về 0 nếu không có nghiệm.
      int d=__gcd(abs(a), abs(b));
      if(c%d) return 0;
      extended_euclid(a/d, b/d, x, y);
      //a/d*x+b/d*y=1
      //a*x+b*y=d
      x*=c/d;
      y*=c/d;
      return 1;
}
```

PHẦN II. HÀM NHÂN TÍNH

I. ĐỊNH NGHĨA VÀ CÁC TÍNH CHẤT CỦA HÀM NHÂN TÍNH

I.1. Định nghĩa

-Hàm$f(n)$ (n là số nguyên) là một hàm nhân tính nếu với mọi cặp số nguyên m,n nguyên tố cùng nhau thì $f(m*n)=f(m)*f(n)$.

-Hàm $f(n)$ (n là số nguyên) là một hàm nhân tính hoàn toàn nếu với mọi cặp số nguyên m,n thì $f(m*n)=f(m)*f(n)$.

I.2. Tính chất

-Theo định lí số học cơ bản: với mọi số nguyên dương n, tồn tại duy nhất một cách viết n dưới dạng tích của luỹ thừa các số nguyên tố:

$n=p_1^{k_1}*p_2^{k_2}*p_3^{k_3}*\ldots*p_a^{k_a}$ sao cho $k_1,k_2,k_3,\ldots,k_a$ là các số nguyên dương và $p_1,p_2,p_3,\ldots,p_a$ là các số nguyên tố tăng dần.

Nếu $f(n)$ là một hàm nhân tính thì:

$$f(n)=f(p_1^{k_1})*f(p_2^{k_2})*\ldots*f(p_a^{k_a}).$$

$$f(n)=f\left(p_1^{k_1}\right)*f\left(\frac{n}{p_1^{k_1}}\right)=f\left(p_2^{k_2}\right)*f\left(\frac{n}{p_2^{k_2}}\right)=\ldots=f\left(p_a^{k_a}\right)*f\left(\frac{n}{p_a^{k_a}}\right).$$

-Dirichlet Convolution: Nếu $f(n)$ và $g(n)$ là hai hàm nhân tính thì:

$$(f*g)(n)=\sum_{d\vee n} f(d)*g(n/d)$$

cũng là một hàm nhân tính.

(Xem Dirichlet Convolution.)

I.3. Một số hàm nhân tính cơ bản.

-Hàm hằng $1(n)$ định nghĩa là $1(n)=1$.

-Hàm định nghĩa $id(n)=n$.

-Hàm luỹ thừa: $f(x)=x^c$. (c là hằng số)

-Cả ba hàm trên đều là hàm nhân tính hoàn toàn.

II. PHƯƠNG PHÁP ĐỂ TÍNH MỘT HÀM NHÂN TÍNH

II.1. Chứng minh hàm cần tính là một hàm nhân tính

II.1.a. Chứng minh bằng công thức

Nếu hàm $f(n)$ có công thức không phụ thuộc vào việc nó có là hàm nhân tính không, để chứng minh$f(n)$ là hàm nhân tính, ta có thể chứng minh $f(n)*f(m)=f(m*n)$(với

mọi m, n nguyên tố cùng nhau) bằng cách biến đổi công thức. Ví dụ xét hàm $f(n)$ là số ước của n.

Nếu n phân tích ra thừa số nguyên tố có dạng $n=p_1^{k_1}*p_2^{k_2}*p_3^{k_3}*\ldots*p_a^{k_a}$ thì

$f(n)=(k_1+1)*(k_2+1)*\ldots*(k_a+1)$.

Xét n và m nguyên tố cùng nhau và:

$n=p_1^{k_1}*p_2^{k_2}*p_3^{k_3}*\ldots*p_a^{k_a}$.

$m=q_1^{h_1}*q_2^{h_2}*q_3^{h_3}*\ldots*q_b^{h_b}$.

thì $m*n=p_1^{k_1}*p_2^{k_2}*p_3^{k_3}*\ldots*p_a^{k_a}*q_1^{h_1}*q_2^{h_2}*q_3^{h_3}*\ldots*q_b^{h_b}$.

$f(n)=(k_1+1)*(k_2+1)*\ldots*(k_a+1)$.

$f(m)=(h_1+1)*(h_2+1)*\ldots*(h_b+1)$.

$f(m*n)=(k_1+1)*(k_2+1)*\ldots*(k_a+1)*(h_1+1)*(h_2+1)*\ldots*(h_b+1)=f(m)*f(n)$

nên $f(n)$ là một hàm nhân tính.

II.1.a. Chứng minh sử dụng Dirichlet Convolution

-Xét hai hàm $f(n)=1$ và $g(n)=1$, cả hai hàm này đều là hàm nhân tính nên theo Dirichlet Convolution,

$(f*g)(n)=\sum_{d\vee n} f(d)*g\left(\frac{n}{d}\right)=\sum_{d\vee n} 1$. Đây chính là hàm đếm số ước của n.

-Đôi khi có những hàm chứng minh bằng cách biến đổi vất vả hơn rất nhiều so với sử dụng Dirichlet Convolution. Xét vì dụ hàm $h(n)$ là tổng các ước của n. Sử dụng Dirichlet Convolution với hai hàm $f(n)=n$ và $g(n)=1$ có:

$(f*g)(n)=\sum_{d\vee n} f(d)*g\left(\frac{n}{d}\right)=\sum_{d\vee n} d=h(n)$.

Nên $h(n)$ là một hàm nhân tính.

II.2. Tìm cách tính trường hợp cơ bản

-Hàm nhân tính giúp ta rút gọn việc tính $f(n)$ cho mọi số nguyên dương n thành tính $f(n)$ với $n=p^k$, p là một số nguyên tố, k là số nguyên dương.

-Nếu có công thức cho f(n) thì sử dụng công thức này.

-Nếu không có công thức nhưng có một thuật toán để tính thì tính cho trường hợp này riêng.

II.3. Tính hàm nhân tính

-Nếu chỉ tính một giá trị của hàm thì phân tích ra thừa số nguyên tố và tính riêng cho từng thừa số nguyên tố. Chỉ có tác dụng khi hàm này khó/ không có công thức để tính trong trường hợp chung.

-Tính cho tất cả các giá trị x thoả mãn $1 \le x \le n$: Thực hiện sàng nguyên tố, trong lúc sàng với mỗi số lưu lại số nguyên tố bé nhất là ước của nó. Duyệt x tăng dần, kiểm tra kiểu của x. Nếu $x$là luỹ thừa của một số nguyên tố thì dùng thuật toán riêng để tính giá trị, nếu không thì dùng tính chất của hàm nhân tính để tính.

Thuật toán trên có độ phức tạp là $O(nlog(n)+m*C)$ với $m$là số luỹ thừa của số nguyên tố không vượt quá n và Clà độ phức tạp để tính hàm cho luỹ thừa của một số nguyên tố. Thuật toán thực hiện sàng nguyên tố mất O¿, kiểm tra xem mỗi số có phải là số nguyên tố không mất $O(nlog(n))$. Đặc biệt nếu hàm cần tính là hàm nhân tính hoàn toàn, độ phức tạp giảm còn là $O(nlog(\log(n))+\frac{n}{\log(n)}*C)$. Vì có $O(\frac{n}{\log(n)})$ số nguyên tố ¿n. Xem Prime number theorem.

PHẦN 3. TÍNH TOÁN ĐỒNG DƯ

I. TÍNH TOÁN ĐỒNG DƯ CƠ BẢN

I.1. Định nghĩa

-Với mọi cặp số nguyên a và $b(b>0)$, tồn tại duy nhất một cặp số nguyên q và $r(0\le r<b)$ sao cho: $a=b*q+r$. Khi đó ta nói r là số dư khi chia $a$cho b và kí hiệu $r=a \bmod b$. Trong tài liệu này tác giả sử dụng kí hiệu % để biểu diễn phép tính chia lấy dư ($r=a\%b$).

-Hai số a và b thoả mãn $a\%m=b\%m$ thì ta nói a và b đồng dư với nhau qua modulo m. Kí hiệu $a\equiv b(\bmod m)$.

I.2. Các tính chất cơ bản và các phép tính

Các số a,b,c,d,k,m đều là số nguyên và $m>0$.

-Tính đối xứng: $a\equiv b(\bmod m)$ thì $b\equiv a(\bmod m)$.

-Tính bắc cầu: $a\equiv b(\bmod m)$ và $a\equiv c(\bmod m)$ thì $a\equiv c(\bmod m)$.

-Tính tương đồng khi cộng cùng giá trị: $a\equiv b(\bmod m)$ thì $a+k\equiv b+k(\bmod m)$.

-Tính tương đồng khi nhân cùng giá trị: $a\equiv b(\bmod m)$ thì $a*k\equiv b*k(\bmod m)$.

-Tính tương đồng khi cộng: $a\equiv b(\bmod m)$ và $c\equiv d(\bmod m)$ thì $a+c\equiv b+d(\bmod m)$.

-Tính tương đồng khi trừ: $a\equiv b(\bmod m)$ và $c\equiv d(\bmod m)$ thì $a-c\equiv b-d(\bmod m)$.

-Tính tương đồng khi nhân: $a\equiv b(\bmod m)$ và $c\equiv d(\bmod m)$ thì $a*c\equiv b*d(\bmod m)$.

-Tính tương đồng khi nâng lên cùng lũy thừa: $a\equiv b(\bmod m)$ thì $a^k\equiv b^k(\bmod m)$.

-Tính tương đồng khi tính toán đa thức: $a\equiv b(\bmod m)$ thì $P(a)\equiv P(b)(\bmod m)$ với mọi đa thức $P(x)$ có các bậc và hệ số của các bậc nguyên.

-Tính loại trừ khi cộng: $a+k\equiv b+k(\bmod m)$ thì $a\equiv b(\bmod m)$

-Tính loại trừ khi nhân: $a*k\equiv b*k(\bmod m)$thì $a\equiv b(\bmod m)$ nếu k và m là hai số nguyên tố cùng nhau.

I.3. Nghịch đảo modulo

-Sự tồn tại: Với hai số nguyên a và $m(m>0;0\le a<m)$, tồn tại duy nhất một số nguyên $b(0\le b<m)$thoả mãn $(a*b)\%m=1$ khi và chỉ khi a và m nguyên tố cùng nhau. Khi đó ta kí hiệu $b=a^{-1}$ là nghịch đảo nhân modulo của $a(\bmod m)$. Lưu ý rằng a^{-1} là một sự lạm dụng kí pháp vì a^{-1} không phải là một số nguyên khi a khác 1 hay -1, tuy nhiên vẫn được sử dụng thường xuyên để tiện cho việc trình bày.

-Tính tương đồng khi nghịch đảo modulo: Nếu $a\equiv b(\bmod m)$ và tồn tại a^{-1} thì $a^{-1}\equiv b^{-1}\ (\bmod m)$.

-Nếu $a*x\equiv b \bmod(m)$ và tồn tại a^{-1} thì $x\equiv b*a^{-1}$.

-Với số nguyên tố p thì tồn tại a^{-1} cho tất cả số nguyên a thoả mãn $a\%b$ khác 0.

II. CÁC ĐỊNH LÍ DÙNG TÍNH TOÁN ĐỒNG DƯ VÀ HỆ QUẢ

II.1. Định lí nhỏ Fermat.

Nếu p là một số nguyên tố thì với mọi số nguyên a, $a^p - a$ là một bội số của p.

-Kí hiệu: $a^p \equiv a (mod\ p)$.

-Hệ quả khi a và p nguyên tố cùng nhau: $a^{p-1} \equiv 1 (mod\ p)$. Khi đó$a^{p-2}$ là nghịch đảo modulo của $a (mod\ p)$.

II.2. Hàm phi Euler

II.2.a. Định nghĩa và kí hiệu

Hàm phi Euler cho một số nguyên dương n được định nghĩa là số lượng số nguyên dương a nguyên tố cùng nhau với n sao cho $a \leq n$. Kí hiệu: $\phi(n)$.

II.2.b. Công thức tính

Công thức tích của Euler:

$$\phi(n) = n * \prod_{p \vee n} (1 - \frac{1}{p}) \text{ với mọi số nguyên tố } p.$$

II.2.c. Tính chất của hàm phi Euler

-$\phi(n)$ là một hàm nhân tính.

-$\phi(n)$ là số chẵn nếu $n > 2$.

-Nếu $n = p^k$ (n, k nguyên) với p là một số nguyên tố thì $\phi(n) = p^k - p^{k-1}$.

-Lưu ý $\phi(1) = 1$.

II.2.d. Phương pháp tính hàm phi Euler

-Tính một giá trị $\phi(n)$: Phân tích $n$ra thừa số nguyên tố và sử dụng công thức nhân Euler. Độ phức tạp của thuật toán này là độ phức tạp để phân tích n ra thừa số nguyên tố.

-Tính tất cả các giá trị $\phi(x)$ với $1 \leq x \leq n$: Thực hiện sàng nguyên tố cho tất cả các số nguyên ¿n.

Duyệt các giá trị của x tăng, trong lúc sàng nguyên tố cũng lưu lại số nguyên tố nhỏ nhất là ước của x gọi số đó là p. Tìm p^k với klớn nhất sao cho $p^k \vee x$.

Nếu $p^k = x$ thì dùng công thức dành cho luỹ thừa của số nguyên tố, nếu không thì dùng tính chất của hàm nhân tính: $\phi(x) = \phi(p^k) * \phi(x / p^k)$.

Code mẫu C++:

```
const int maxp=1000000;
int p[maxp+1];
int phi[maxp+1];
void calculate_phi(){
    phi[1]=1;
    for(int i=2; i<=maxp; i++)
      if(p[i]==0)
```

-Hệ quả của định lí Euler: $a^{\phi(n)-1} \equiv a^{-1} (mod\, n)$ nếu a và n nguyên tố cùng nhau.

-Hệ quả của định lí Euler: nếu hai số a và b thoả mãn $a \equiv b (mod\, \phi(n))$ thì $k^a \equiv k^b (mod\, n)$ nếu k và n nguyên tố cùng nhau.

II.3. Định lí phần dư Trung Hoa.

II.3.a. Định nghĩa.

-Xét hệ phương trình đồng dư:

$$\begin{cases} x \equiv a_1 (mod\, m_1) \\ x \equiv a_2 (mod\, m_2) \\ x \equiv a_3 (mod\, m_3) \\ x \equiv a_4 (mod\, m_4) \\ \quad \dots \\ x \equiv a_k (mod\, m_k) \end{cases}$$

trong đó $m_1, m_2, \dots, m_k$ đôi một nguyên tố cùng nhau.

Định lí phần dư Trung Hoa chỉ ra rằng:

Hệ phương trình đồng dư nói trên có nghiệm duy nhất theo modul

$$M = m_1 * m_2 * \dots * m_k.$$

nghiệm đó là $x \equiv$ ¿ với $M_i = M / m_i$ và $y_i = M_i^{-1} (mod\, m_i)$

II.3.b. Trường hợp $k = 2$.

-Vì $m_1, m_2, \dots, m_k$ đôi một nguyên tố cùng nhau nên giải được trường hợp khi $k = 2$ là sẽ giải được trường hợp tổng quát bằng cách thực hiện thuật toán $k - 1$ lần, kiến cho việc cài đặt đơn giản hơn.

-Ta có $x \equiv a_1 (mod\, m_1)$ nên $x = m_1 * q_1 + a_1$.

$x \equiv a_2 (mod\, m_2)$ nên $x = m_2 * q_2 + a_2$.

Vậy $m_1 * q_1 + a_1 = m_2 * q_2 + a_2$. Phương trình này có thể viết lại thành

$m_1 * q_1 + m_2 * q_2 = c$ với q_1, q_2, c là các số nguyên. Đây là một phương trình Diophantine tuyến tính, nên có thể giải được giá trị q_1, q_2 bằng giải thuật Euclid mở rộng, từ đó suy ra x.

PHẦN 4. BÀI TẬP ỨNG DỤNG.

BÀI 1. POWERTOWER - THÁP LUỸ THỪA

1.1 Đề bài

Cho 2 số nguyên không âm n, a với $a>0$.

Tháp luỹ thừa bậc n của a được định kí hiệu là $a \uparrow\uparrow n$ (tham khảo Knuth's up-arrow notation) được định nghĩa:

$$a \uparrow\uparrow n = \begin{cases} 1 khi\ n=0 \\ a^{a \uparrow\uparrow (n-1)} khi\ n>0 \end{cases} = a^{a^{a^{\cdot^{\cdot^{\cdot}}}}} (n \text{ lần})$$

Chú ý rằng khi tính giá trị tháp lũy thừa thì các phép lũy thừa được thực hiện theo thứ tự từ trên xuống.

Yêu cầu: Cho 3 số a, n, m tính $(a \uparrow\uparrow n)\%m$.

Input: Nhập vào dữ liệu từ file POWERTOWER.INP:

Dòng đầu là số nguyên dương t, số bộ test.

Trong t dòng tiếp theo, mỗi dòng 3 số nguyên dương a, n, m <=10^5 cách nhau bởi dấu cách.

Output: Với mỗi dòng dữ liệu từ file input, ghi ra file POWERTOWER.OUT một dòng là kết quả bài toán.

Ví dụ:

POWERTOWER.INP	POWERTOWER.OUT
6 1 4 100 4 2 100 3 3 1000 6 5 10000 2 100000 100 100000 20 78923	1 56 987 8656 36 41154

1.2. Thuật toán

- Định nghĩa hàm F(a, n, m) là kết quả của bài toán. Các trường hợp đặc biệt gồm có:

F(1, n, m)=1%m;

F(a, n, 1)=0;

F(a, 0, m)=1%m;

F(a, 1, m)=a%m;

- Khi không trường hợp đặc biệt nào xảy ra, ta biết rõ a>1, m>1, n>1;

- Nhận thấy rằng để rút gọn bài toán, cần sử dụng hệ quả thứ 3 của định lí Euler, tuy nhiên nhận thấy rằng gcd(a, m) chưa chắc chắn đã bằng 1, vì vậy cần chia m thành hai phần:

m1 và m2 với m1*m2=m và gcd(a, m2)=1=gcd(m1, m2)=1 và sử dụng định lí phần dư Trung Hoa.

- $F(a, n, m2) \equiv a^{F(a, n-1, \phi(m2))} \pmod{m2}$ tính được bằng cách gọi đệ quy tính F(a, n-1, ϕ(m2)) và dùng thuật toán luỹ thừa nhanh để tính trong O(log(F(a, n-1, ϕ(m2)))).

- Để tính F(a, n, m1) thì nhận thấy là m1<=10^5 nên một số mũ tối đa của một số nguyên tố nào đó của m1 không vượt quá 20 (2^20>10^5) nên a^x%m1=0 nếu x>=20. Xét các trường hợp sau:

Nếu a=2 thì

n>4=>F(a, n, m1)=0

n<=4 có thể tính chính xác a↑↑n

Nếu a>=32 thì a↑↑n>=20 vì n>1.

Nếu n>2 thì a↑↑n>=20 vì 3↑↑3>=20.

Nếu không thì n=2 có thể tính được a^a bằng thuật toán luỹ thừa nhanh.

- Sau khi tính được F(a, n, m1) và F(a, n, m2) sử dụng định lí đồng dư Trung Quốc và tính F(a, n, m). Có thể sử dụng cách tính nêu trên hoặc áp dụng thẳng công thức tổng quát.

1.3. Đánh giá độ phức tạp

Với mỗi hàm F(a, n, m) được gọi, tính mất O(log(a)) vì:

Tối đa 2 lần tính luỹ thừa nhanh. m1 và m2 tính trong log(a) vì có thể phân tích ra thừa số nguyên tố các giá trị này trong O(log(a)) nếu chuẩn bị sẵn mảng số nguyên tố bé nhất là ước trong khi sàng nguyên tố tính hàm phi Euler.

Thoạt nhìn, độ phức tạp thuật toán sẽ là O(n*log(a)) cho mỗi test vì hàm F(a, n, m) có thể bị gọi tới n lần, tuy nhiên độ phức tạp thực tế là O(min(n, log(m))*log(a)) vì:

Hàm F(a, n, m) gọi hàm F(a, n-1, ϕ(m2)) mà m2<=m và ϕ(m2) là số chẵn nếu m2>3 nên từ lần đệ quy thứ 2, m2 chẵn hoặc m2<3. Nếu m2 là số chẵn thì ϕ(m2)<=m2/2 (dựa theo công thức nhân của Euler), nếu m2<3 thì m2=1. Do đó từ lần thứ 2 sau mỗi lần gọi hàm, m trong F(a, n, m) sẽ bị giảm đi ít nhất 2 lần, hay chỉ tốn tối đa log(m) lần gọi hàm cho đến khi gặp một trường hợp đặc biệt.

1.4. Code

```
#include <bits/stdc++.h>
using namespace std;
#define FOR(i, j, k) for(int i=(j); i<=(k); i++)
#define FFOR(i, j, k) for(int i=(j); i<(k); i++)
#define DFOR(i, j, k) for(int i=(j); i>=(k); i--)
#define bug(x) cerr<<#x<<" = "<<(x)<<'\n'
```

```
#define pb push_back
typedef long long ll;
template <typename T> inline void read(T &x){
    char c;
    bool nega=0;
    while((!isdigit(c=getchar()))&&(c!='-'));
    if(c=='-'){
        nega=1;
        c=getchar();
    }
    x=c-48;
    while(isdigit(c=getchar())) x=x*10+c-48;
    if(nega) x=-x;
}
template <typename T> inline void writep(T x){
    if(x>9) writep(x/10);
    putchar(x%10+48);
}
template <typename T> inline void write(T x){
    if(x<0){
        putchar('-');
        x=-x;
    }
    writep(x);
}
template <typename T> inline void writeln(T x){
    write(x);
    putchar('\n');
}
#define taskname "POWERTOWER"
int pri[100001];
ll phi[100001];
ll power(const ll &a, ll x, const ll &mod){
    if(x==0) return 1;
    ll t=power(a, x/2, mod);
    t=(t*t)%mod;
    if(x%2) t=(t*a)%mod;
    return t;
}
vector <int> prime;
ll F(ll &a, ll n, ll m){
    if(m==1) return 0;
    if(n==0) return 1%m;
    if(n==1) return a%m;
    ll m0=1;
    ll m1=m;
    for(int d: prime){
        while(m1%d==0){
            m1/=d;
            m0*=d;
        }
```

```
        }
        ll r0;
        ll r1;
        if(a==2){
            if(n>4) r0=0;
            else if(n==4) r0=65536%m0;
            else if(n==3) r0=16%m0;
            else r0=4%m0;
        }
        else if(a>=32) r0=0;
        else if(n>2) r0=0;
        else r0=power(a, a, m0);
        r1=power(a, F(a, n-1, phi[m1]), m1);
                    return    (r0*m1*power(m1,    phi[m0]-1,
m0)+r1*m0*power(m0, phi[m1]-1, m1))%m;
    }
    int t;
    int main(){
        freopen(taskname".inp", "r", stdin);
        freopen(taskname".out", "w", stdout);
        FOR(i, 2, 100000) if(!pri[i]) for(int j=i; j<=100000;
j+=i) pri[j]=i;
        FOR(i, 1, 100000){
            phi[i]=i;
            int u=i;
            int old=-1;
            while(u>1){
                if(pri[u]!=old){
                    old=pri[u];
                    phi[i]=phi[i]/pri[u]*(pri[u]-1);
                }
                u/=pri[u];
            }
        }
        read(t);
        ll a, n, m;
        while(t--){
            read(a);
            read(n);
            read(m);
            prime.clear();
            ll b=a;
            while(b>1){
                prime.pb(pri[b]);
                while((b%prime.back())==0) b/=prime.back();
            }
            if(a>1) writeln(F(a, n, m));
            else writeln(1%m);
        }
    }
```

1.5. Test và Cảm nhận

https://drive.google.com/file/d/16qWLNDlcVQMnz-ue2ob76__hKwXEHK1a/view

Bài toán tháp luỹ thừa về mặt thuật toán thì thẳng thắn, đơn giản tuy nhiên lại yêu cầu có một số hiểu biết về tính toán đồng dư. Việc đánh giá đúng độ phức tạp cho ta một kết quả hay nhưng đơn giản mà lại khó tìm thấy.

BÀI 2. DPEQN - CONGRUENCE EQUATION

2.1. Đề bài https://vn.spoj.com/problems/DPEQN/

Cho phương trình đồng dư:

$$a_1x_1+a_2x_2+...+a_nx_n=b\,(mod\,m)$$

Trong đó $a_1, a_2, \ldots, a_n, b$ và m là các hằng số nguyên dương cho trước. $x_1, x_2, \ldots, x_n$ là các ẩn.

Tìm một nghiệm của phương trình trên, hoặc thông báo phương trình vô nghiệm.

Dữ liệu: Dòng đầu tiên ghi số bộ test, mỗi bộ test có dạng như sau:

Dòng 1: $n\,(1\le n\le 100)$

Dòng 2: gồm n số nguyên $a_1, a_2, \ldots, a_n\ (1\le ai\le 10^8)$

Dòng 3: $b, m\,(1\le b, m\le 10^8)$

Mỗi bộ test được phân cách bởi một dòng trắng ở đầu.

Kết quả:

Với mỗi bộ test, nếu phương trình không có nghiệm, in ra dòng "NO". Trong trường hợp có nghiệm, in ra trên một dòng n số nguyên $x_1, x_2, \ldots, x_n$ $(0\le xi<m)$ là một nghiệm tìm được.

Ví dụ:

Dữ liệu	Kết quả
2 2 4 6 6 10 2 4 6 3 8	1 2 NO

2.2. Thuật toán

Vì đây là một phương trình đồng dư nên ta không cần quan trọng phải giải được nghiệm nguyên trong nửa khoảng [0, m], chỉ cần giải nghiệm nguyên bất kì là tìm được nghiệm trong khoảng đó.

Trước hết xét phương trình Diophantine

$ax+by=c$ với a,b,c là hằng số có nghiệm khi $gcd(a,b)\vee c$. Thay đổi góc nhìn, ta có hàm hai chiều $f(x,y)=ax+by=dk$ với $d=gcd(a,b)$ có nghiệm với k là một số bất kì hay nói cách khác, nếu phương trình là $ax+by$, ta có thể rút gọn các giá trị có thể của nó về dk.

Cứ đệ quy như vậy, phương trình trong đề bài trở thành dạng:

$$gcd(a¿¿1,a_2,...,a_n)*z=b(mod\ m)¿$$

với z là một số nguyên nào đó. Phương trình trên có thể viết lại thành:

$$gcd(a¿¿1,a_2,...,a_n)*z+t*m=b¿$$

với t cũng là một số nguyên.

Đây lại là một phương trình Diophantine. Phương trình này có nghiệm thì có kết quả, nói cách khác bài toán có nghiệm khi và chỉ khi $gcd(a¿¿1,a_2,...,a_n,m)¿$|b. Việc kiểm tra xem phương trình có nghiệm hay không là đơn giản. Vậy làm thế nào để giải ra nghiệm khi có nghiệm?

Đầu tiên giải $gcd(a¿¿1,a_2,...,a_n)*z+t*m=b¿$ để tìm ra z. Đưa z về dạng $0\le z<m$ để tránh bị tràn số khi xử lí tiếp. Bây giờ phải giải

$$gcd(a¿¿1,a_2,...,a_{n-1})*p+a_n*q=gcd(a¿¿1,a_2,...,a_n)*z¿¿.$$

Phương trình này không khác gì phương trình mà ta bắt đầu với, nên giải tượng tự và thi được q, chính là ra trí của x_n, sau đó lại đi giải tiếp

$$gcd(a¿¿1,a_2,...,a_{n-2})*u¿+a_{n-1}*v=gcd(a¿¿1,a_2,...,a_{n-1})*p¿$$

Các bước đều như nhau nên có thể đệ quy để giải. Khi phương trình được rút gọn về dạng $a_1*x_1+a_2*x_2=c$ thì có thể giải thẳng giá trị x_1 và x_2 luôn. Sau đó chỉ việc chuyển các giá trị về đúng dạng $0\le x<m$ (bằng cách mod cho m).

Mất $O(n)$ lần tìm ước chung lớn nhất và $O(n)$ lần giải phương trình Diophantine nên độ phức tạp là $O(nlog(m))$.

Chú ý có thể cải thiện thời gian chạy bằng cách đệ quy tìm gcd của k số đầu tiên, thử xem k số đó đã đủ để giải phương trình chưa, nếu rồi thì cho tất cả các x còn lại bằng 0 là được. Cách này độ phức tạp xấu nhất vẫn sẽ là $O(nlog(m))$ tuy nhiên độ phức tạp trung bình xuống còn $O(\log(m)^2+n)$ vì: Với số a, chuyển từ a đến gcd(a, b) thì gcd(a, b)=a hoặc gcd(a, b)|a=>gcd(a, b)<a/2. Xác suất để b là bội của a tương đối thấp nếu a lớn nêu sau trung bình O(log(a)) bước thì gcd(a, b) sẽ trở thành 1, đảm bảo giải được phương trình.

2.3. Code

Code C++:

```
#include <bits/stdc++.h>
using namespace std;
#define FOR(i, j, k) for(int i=(j); i<=(k); i++)
#define FFOR(i, j, k) for(int i=(j); i<(k); i++)
#define DFOR(i, j, k) for(int i=(j); i>=(k); i--)
#define bug(x) cerr<<#x<<" = "<<(x)<<'\n'
#define pb push_back
#define mp make_pair
```

```
#define setbit(s, i) (s|=(1LL<<(i)))
#define bit(s, i) (((s)>>(i))&1LL)
#define mask(i) ((1LL<<(i)))
#define builtin_popcount __builtin_popcountll
typedef long long ll;
typedef long double ld;
template <typename T> inline void read(T &x){
    char c;
    bool nega=0;
    while((!isdigit(c=getchar()))&&(c!='-'));
    if(c=='-'){
        nega=1;
        c=getchar();
    }
    x=c-48;
    while(isdigit(c=getchar())) x=x*10+c-48;
    if(nega) x=-x;
}
template <typename T> inline void writep(T x){
    if(x>9) writep(x/10);
    putchar(x%10+48);
}
template <typename T> inline void write(T x){
    if(x<0){
        putchar('-');
        x=-x;
    }
    writep(x);
}
template <typename T> inline void writeln(T x){
    write(x);
    putchar('\n');
}
#define taskname "DPEQN"
int n;
ll a[101];
ll x[101];
ll b, m;
void Euclid(ll a, ll b, ll &x, ll &y){
    ///ax+by=1
    if(b==0){
        ///(a, b)=1;
        x=1;
        y=0;
        return;
    }
    Euclid(b, a%b, y, x);
    ///b*y+(a%b)*x=1
    ///b*y+(k*b+(a%b))*x=1+k*b*x
    ///b*(y-k*x)+a*x=1
    y-=(a/b)*x;
```

```
}
void Diophantine(ll a, ll b, ll c, ll &x, ll &y){
    ///ax+by=c
    ll d=__gcd(a, b);
    a/=d;
    b/=d;
    c/=d;
    Euclid(a, b, x, y);
    x*=c;
    y*=c;
    ///a*x+b*y=c
}
ll solve(int i, ll d){
    if(d){
        ///d*x=b(mod m)
        ///d*x+r*m=b
        if(b%__gcd(d, m)){
            if(i>n) return -1;
        }
        else{
            ll x, y;
            Diophantine(d, m, b, x, y);
            x%=m;
            if(x<0) x+=m;
            return x;
        }
    }
    ll c=__gcd(d, a[i]);
    ll res=solve(i+1, c);
    if(res==-1) return -1;
    ///c*res==b;
    ///solve for (d/c)*x+(a[i]/c)*y=res
    ll mres;
    Diophantine(d/c, a[i]/c, res, mres, x[i]);
    x[i]%=m;
    if(x[i]<0) x[i]+=m;
    mres%=m;
    if(mres<0) mres+=m;
    return mres%m;
}
void solve(){
    read(n);
    FOR(i, 1, n) read(a[i]);
    read(b);
    read(m);
    FOR(i, 1, n) a[i]%=m;
    FOR(i, 1, n) x[i]=0;
    b%=m;
    if(solve(1, 0)==-1){
        puts("NO");
        return;
```

```
        }
        FOR(i, 1, n){
            write(x[i]);
            putchar(" \n"[i==n]);
        }
    }
    int main(){
        int t;
        read(t);
        while(t--) solve();
    }
```

2.4. Test và cảm nhận

Có thể chấm bài tại đây Submit.

Bài DPEQN yêu cầu học sinh hiểu rõ bản chất của thuật toán Euclid mở rộng, phương trình Diophantine khi áp dụng trong tính toán đồng dư. Bài cũng yêu cầu khả năng cài đặt đệ quy, giúp giảm gánh nặng cho người cài đặt.

BÀI 3. GCDSUM – TỔNG CÁC ƯỚC CHUNG LỚN NHẤT

3.1. Đề bài https://vn.spoj.com/problems/GCDSUM/

Một lần, ktuan được thầy giáo cho bài tập về nhà, yêu cầu tính tổng tất cả các ước chung lớn nhất của các cặp số (i, j) thỏa mãn : 1<=i< j<=N (N là một số tự nhiên cho trước). Rất nhanh chóng, ktuan đã cho ra một đoạn code như sau:

```
for i:=1 to N-1 do for j:=i+1 to N do sum := sum + gcd( i, j);
```

với gcd là hàm tính ước chung lớn nhất của 2 số, sum chính là kết quả cuối cùng.

Thầy giáo yêu cầu ktuan dùng chương trình trên để tính kết quả với N = 1000000. Tuy nhiên, chương trình trên chạy quá lâu. Để khắc phục vấn đề, ktuan đã viết lại đoạn mã đó bằng C++ (với hi vọng C++ sẽ chạy nhanh hơn pascal nhiều) :

```
for(int i=1;i< N;++i) for(int j=i+1;j<=N;++j) sum += gcd(i,j);
```

Thật không may, đoạn chương trình trên vẫn không giải quyết được vấn đề, bạn hãy giúp ktuan giải đáp yêu cầu của thầy giáo.

Lưu ý: bài này có thể giải bằng phương pháp Quy hoạch động và các kiến thức sơ đẳng trong toán học, không cần sử dụng những kiến thức toán học phức tạp không nằm trong phạm vi chương trình phổ thông.

Dữ liệu:

Gồm nhiều dòng, mỗi dòng là một số N (1<=N<=10^6) ứng với một test. Dữ liệu vào sẽ kết thúc sau khi gặp N=0 (bạn không cần thực hiện test này).

Kết quả:

Với mỗi giá trị của N, in ra một dòng là giá trị của sum sau khi thực hiện đoạn mã trên.

Ví dụ:

Dữ liệu	Kết quả
4 0	**7**

3.2. Thuật toán

Giả sử số thứ nhất lớn hơn số thứ hai,

Thay đổi lại góc nhìn của bài toán, thay vì duyệt mọi cặp số và tính ước chung lớn nhất, ta duyệt các cặp số thứ nhất, ước chung lớn nhất của hai số và tính số lượng số thứ hai. Rõ ràng ước chung lớn nhất là ước của số thứ nhất, nên đây chính xác là duyệt số cặp ước bội. Điều này làm được trong $O(nlog(n))$.

Với mỗi cặp ước bội $d, n(d \vee n)$, có bao nhiêu số nguyên dương $m<n$ mà $gcd(n,m)=d$?

Khi $d=n$ thì kết quả là 0, nếu không

Khi $gcd(n,m)=d$ thì ta có thể viết $n=d*p$ và $m=d*q$ với $gcd(p,q)=1$, $p>1$

Do $m<n$ nên $p>q$. Nên ta cần đếm số lượng $q<p$ mà $gcd(p,q)=1$. Đây chính là $\phi(p)$. Do tính được các giá trị của $\phi(p)$ trước trong $O(nlog(n))$ nên ta có thể tính với mỗi số n giá trị

$$f(n)=\sum_{m=1}^{n-1} gcd\,(n,m)$$

Kết quả bài toán đơn thuần chỉ là tổng tiền tố từ $f(1)$ đến $f(n)$.

3.3.Code

```
#include <bits/stdc++.h>
using namespace std;
#define FOR(i, j, k) for(int i=(j); i<=(k); i++)
#define FFOR(i, j, k) for(int i=(j); i<(k); i++)
#define DFOR(i, j, k) for(int i=(j); i>=(k); i--)
#define bug(x) cerr<<#x<<" = "<<(x)<<'\n'
#define pb push_back
#define mp make_pair
#define setbit(s, i) (s|=(1LL<<(i)))
#define bit(s, i) (((s)>>(i))&1LL)
#define mask(i) ((1LL<<(i)))
#define builtin_popcount __builtin_popcountll
typedef long long ll;
typedef long double ld;
template <typename T> inline void read(T &x){
    char c;
    bool nega=0;
    while((!isdigit(c=getchar()))&&(c!='-'));
    if(c=='-'){
        nega=1;
        c=getchar();
```

```
        }
        x=c-48;
        while(isdigit(c=getchar())) x=x*10+c-48;
        if(nega) x=-x;
    }
    template <typename T> inline void writep(T x){
        if(x>9) writep(x/10);
        putchar(x%10+48);
    }
    template <typename T> inline void write(T x){
        if(x<0){
            putchar('-');
            x=-x;
        }
        writep(x);
    }
    template <typename T> inline void writeln(T x){
        write(x);
        putchar('\n');
    }
    #define taskname "GCDSUM"
    int n;
    ll f[1000001];
    int p[1000001];
    ll phi[1000001];
    int main(){
             FOR(i, 2, 1000000) if(p[i]==0) for(int j=i;
j<=1000000; j+=i) if(!p[j]) p[j]=i;
        FOR(i, 2, 1000000) if(p[i]==i) phi[i]=i-1; else{
            int x=i;
            int k=1;
            while(x%p[i]==0){
                x/=p[i];
                k*=p[i];
            }
            if(x==1) phi[i]=i-k/p[i];
            else phi[i]=phi[k]*phi[i/k];
        }
         FOR(i, 1, 1000000) for(int j=i+i; j<=1000000; j+=i)
f[j]+=phi[j/i]*i;
        FOR(i, 1, 1000000) f[i]+=f[i-1];
        while(true){
            read(n);
            if(n==0) return 0;
            writeln(f[n]);
        }
    }
```

3.4.Test và cảm nhận

Có thể chấm bài tại đây [Submit.]

Bài GCDSUM đòi hỏi học sinh phải có hiểu biết về hàm phi Euler, biết đánh giá đúng độ phức tạp của đoạn code và biết thay đổi khía cạnh của bài toán.

BÀI 4. VOSPOW – ĐỘ BÁ ĐẠO CỦA ĐỘI HÌNH

4.1. Đề bài https://vn.spoj.com/problems/VOSPOW

Hằng năm cứ vào khoảng cuối tháng 10, hai trườnng trung học danh tiếng nhất ở nước Alphabet, trường XYZ và trường ABC, sẽ tổ chức giải bóng chuyền với mục đích tạo ra sân chơi lành mạnh giữa các học sinh của hai trường và cũng là dịp để các học sinh tìm hiểu kỹ hơn về trường bạn. Vì sân vận động rất lớn nên mỗi đội có tới **N** người chơi. Trường XYZ là một trường chuyên về các môn tự nhiên còn trường ABC là trường chuyên về các môn xã hội. Để chuẩn bị chiến thuật cho giải đấu sắp tới, trường XYZ cần phải biết **mức độ bá đạo** của đội bên kia. Nhờ quen biết rộng nên trường XYZ đã biết được **chỉ số trung bình** các thí sinh sắp tới sẽ chơi cho đội của trường ABC. **Độ bá đạo** của của một đội bóng chuyền sẽ có giá trị bằng **tổng độ bá đạo** của các thành viên trong đội và **lấy phần dư trong phép chia cho BASE** trong đó độ bá đạo của mỗi thành viên sẽ bằng lũy thừa bậc Q của chỉ số trung bình của thành viên đó. Biết rằng Q có dạng là **kT**.

Yêu cầu: Tính độ bá đạo của đội bóng trường ABC.

Dữ liệu vào:

Dòng đầu chứa số N, k, T, BASE trong đó N <= $\mathbf{10^7}$, k <= **50,** T <= $\mathbf{10^5}$, BASE <= $\mathbf{10^{12}}$.

Dòng tiếp theo chứa 2 số nguyên dương là mul và seed.

Lưu ý: **30%** số test T <= **50**.

Nguyên tắc sinh dãy A với A[i] là chỉ số trung bình của thành viến thứ i như sau:

A[1] = (mul*seed + seed) mod maxC.

A[i] = (A[i-1]*mul + seed) mod maxC.

a mod b là phép lấy phần dư của phép chia a cho b.

0 <= mul, seed <= $\mathbf{10^6}$.

maxC = $\mathbf{2^{20}}$.

Kết quả ra: Gồm một dòng chứa mốt số nguyên là kết quả bài toán.

Ví dụ:

Dữ liệu	Kết quả
4 2 2 89133 3 6	50886

4.2. Thuật toán

Đề bài phát biểu rất đáng sợ, tuy nhiên có thể rút gọn còn các vấn đề sau:

Cho 4 số n, k, t và $base$.

Trong đó, $1 \le n \le 10^7, k \le 50, t \le 100000, base \le 10^{12}$

Cho dãy n số $a_1, a_2, a_3 ..., a_n$ với $1 \le a_i \le 2^{20}$.

Tính:

$$\sum_{i=1}^{n} a_i^{(k¿¿t)}¿$$

Đầu tiên nhận thấy *base* có thể lên tới 10^{12} nên không thể tính thẳng $(a*b)\%base$ mà không bị tràn số nguyên như khi $base \approx 10^9$. Sử dụng thuật toán nhân đôi liên tiếp (tương tự như tính luỹ thừa nhanh) có thể tính tới $base \approx 10^{18}$ nhưng lại mất độ phức tạp $O¿$, sẽ làm chương trình chạy rất chậm. Rất may có cách để tính $(a*b)\%base$ trong O(1) trong trường hợp $base \le 10^{12}$. Cách làm đơn giản là viết $a = 10^6 * x + y, b = 10^6 * z + t$. Khi đó thì $0 \le x, y, z, t \le 10^6$ có:

$$a*b = (10^6 * x + y) * (10^6 * z + t) = x*z*10^6*10^6 + 10^6 * x*t + 10^6 * y*z + z*t.$$

Các phép nhân bây giờ chỉ là nhân một số $¿base$ với 1 số $¿10^6$, kết quả bé hơn 10^{18} nên có thể tính chính xác bằng số 64 bit. Tham khảo hàm multi trong code của bài này để hiểu rõ hơn.

Vì giá trị $a_i^{(k¿¿t)¿}$ không phụ thuộc vào chỉ số i mà chỉ phụ thuộc vào giá trị của a_i nên có thể đếm phân phối để rút gọn bài toán về tính $min(n, 2^{20})$ giá trị. Từ đây lời giải sẽ cho rằng cần phải tính n=2^{20} giá trị $f(x) = x^{(k¿¿t)¿}$

Chia bài toán làm 2 phần để dễ xử lí. Nếu $k^t < 1000$ thì có thể thoải mái duyệt tính từng giá trị để tính. Nếu không ta có: $f(x) = x^{k^t}$ là một hàm nhân tính hoàn toàn, nên thực tế ta chỉ cần tính $f(x)$ với x là một số nguyên tố. Khi x là một số nguyên tố thì có 2 trường hợp sau:x và *base* nguyên tố cùng nhau thì áp dụng tính chất hàm phi Euler $x^{\phi(base)} \equiv 1 (mod\, base)$ sẽ tính được$f(x) = x^{k^t} = x^{(k¿¿t\, \%\phi(base))¿}$ trong $O¿$ nếu biết $\phi(base)$.x là một thừa số nguyên tố của *base*. Gọi p là số nguyên dương lớn nhất mà $x^p \vee base$. Bhì nhận thấy là $k^t \ge 1000$ mà số mũ lớn nhất của một thừa số nguyên tố nào đó trong *base* chỉ là $\log 2(10^{12}) \approx 40$ nên $f(x)\%x^p = 0$. Vì $base / x^p$ nguyên tố cùng nhau với x nên có thể dùng hàm phi Euler tính $f(x)\%¿)$ trong $O(\log(t) + \log(\phi(base/x^p)))$ nếu biết $\phi(base/x^p)$. Sử dụng định lí phần dư Trung Hoa để tính kết quả $f(x)\%base$ mất $O(\log(base))$

Phân tích *base* ra thừa số nguyên tố thì tính trước được $\phi(base)$ và $\phi(base/x^p)$ với các thừa số x. Có ít hơn log2(base) thừa số nguyên tố trong *base* nên nếu dùng thuật phân tích độ phức tạp $O(\sqrt{base})$ thì điều này sẽ tính được trong $O(\sqrt{base} + \log 2(base))$.

Tổng hợp lại, có các việc cần làm là phân tích base ra thừa số nguyên tố và tính trước các giá trị hàm phi Euler cần thiết mất O($\sqrt{base}¿$, sàng nguyên tố và lưu lại ước nguyên tố bé nhất mất $O(2^{20} \log(\log(2^{20})))$,tính hàm cho các số nguyên tố mất $O\left(\frac{2^{20}}{log(2^{20})} * \log(base)\right) = O(2^{20})$ (xem PNT), tính cho các hợp số và cộng kết quả mất $O(2^{20})$. Vậy độ phức tạp có thể được đánh giá bằng $O¿$.

4.3. Code

Code C++:

```
#include <bits/stdc++.h>
using namespace std;
#define FOR(i, j, k) for(int i=j; i<=k; i++)
#define FFOR(i, j, k) for(int i=j; i<k; i++)
```

```
    #define DFOR(i, j, k) for(int i=j; i>=k; i--)
    #define bug(x) cerr<<#x<<" = "<<x<<'\n'
    #define pb push_back
    #define mp make_pair
    typedef long long ll;
    template <typename T> inline void read(T &x){
        char c;
        bool nega=0;
        while((!isdigit(c=getchar()))&&(c!='-'));
        if(c=='-'){
            nega=1;
            c=getchar();
        }
        x=c-48;
        while(isdigit(c=getchar()))
            x=x*10+c-48;
        if(nega)
            x=-x;
    }
    template <typename T> inline void writep(T x){
        if(x>9) writep(x/10);
        putchar(x%10+48);
    }
    template <typename T> inline void write(T x){
        if(x<0){
            putchar('-');
            x=-x;
        }
        writep(x);
    }
    template <typename T> inline void writeln(T x){
        write(x);
        putchar('\n');
    }
    #define taskname "VOSPOW"
    const ll cmax=1048576;
    ll cnt[cmax+1];
    ll n, k, t;
    ll  base, mul, seed, a;
    ll phibase;
    ll b, d;
    inline ll multi(ll a, ll c, const ll &base){
        b=a%1000000LL;
        a/=1000000LL;
        d=c%1000000LL;
        c/=1000000LL;
                                return      ((a*1000000000000LL)
%base*c+b*c*1000000LL+a*d*1000000LL+b*d)%base;
    }
    ll power(const ll &a, ll x, const ll &base){
        if(x==0) return 1;
```

```
        ll temp=power(a, x/2, base);
        temp=multi(temp, temp, base);
        if(x%2) temp=multi(temp, a, base);
        return temp;
    }
    void fact(ll base){
        ll i=2, old, now;
        phibase=1;
        while(i*i<=base){
            if(base%i==0){
                old=now=1;
                ll phi=1;
                while(base%i==0){
                    base/=i;
                    now*=i;
                    phi=now-old;
                    old=now;
                }
                phibase*=phi;
            }
            i+=1;
        }
        if(base>1) phibase*=base-1;
    }
    ll fpri[cmax+1];
    bool done[cmax+1];
    ll AR[cmax+1];
    void sieve(){
        for(ll i=2; i<=cmax; i+=2)
            fpri[i]=2;
        for(ll i=3; i<=cmax; i+=2){
            if(!fpri[i]){
                for(ll j=i, temp=i<<1; j<=cmax; j+=temp)
                    if(!fpri[j])
                        fpri[j]=i;
            }
        }
    }
    ll R(ll i){
        if(done[i]) return AR[i];
        done[i]=1;
        if(i<=1) return AR[i]=i%base;
        if(i>=base) return AR[i]=R(i%base);
        if(fpri[i]==i){
                if(base%i)  return  AR[i]=power(i,  power(k,  t,
phibase), base);
            ll n1=1, n2=base;
            while(n2%i==0){
                n1*=i;
                n2/=i;
            }
```

```
            if(n2==1) return AR[i]=0;
               ll r2=power(i, power(k, t, phibase/(n1-n1/i)),
n2);
                           r2=multi(r2,  multi(n1,  power(n1,
phibase/(n1-n1/i)-1, n2), base), base);
            return AR[i]=r2;
        }
        ll temp1=R(fpri[i]);
        ll temp2=R(i/fpri[i]);
        return AR[i]=multi(temp1, temp2, base);
    }
    int main(){
        read(n);
        read(k);
        read(t);
        read(base);
        read(mul);
        read(seed);
        mul%=cmax;
        seed%=cmax;
        a=(mul*seed+seed)%cmax;
        FOR(i, 1, n){
            cnt[a]++;
            a=(a*mul+seed)%cmax;
        }
        ll ans=0;
        if(log((double)k)*double(t)>log(double(50))){
            fact(base);
            sieve();
            ll i=1;
            while(i<cmax){
                if(cnt[i])
                    ans=(ans+multi(R(i), cnt[i], base))%base;
                i++;
            }
        }
        else{
            ll p=1;
            FOR(i, 1, t)
                p*=k;
            DFOR(i, cmax-1, 1)
                if(cnt[i])
                    ans=(ans+multi(power(i, p, base), cnt[i],
base))%base;
        }
        writeln(ans);
    }
```

4.4.Test và Cảm nhận

Có thể chấm bài tại đây [Submit.]

Bài VOSPOW yêu cầu học sinh phải nắm rõ về hàm nhân tính, hàm phi Euler, định lí phần dư Trung Hoa, sàng nguyên tố cũng như biết cách tính mod cho "số lớn" nhanh để có thể cài đặt được thuật toán có thời gian chạy tốt. So với các bài được nộp trên spoj, code ở trên là một trong những bài có thời gian chạy tốt nhất.

Bài 5. GROUP

Bờm rất thích chơi với những con số. Để thử thách cậu ta, thầy giáo đã giao cho Bờm một bài toán. Thầy đưa cho Bờm N số nguyên. Sau đó thầy sẽ chọn ngẫu nhiên 1 số tự nhiên M. Yêu cầu của thầy rất đơn giản: chia N số nguyên vào M nhóm sao cho tổng chi phí của M nhóm là nhỏ nhất có thể. Chi phí của 1 nhóm được tính bằng chênh lệch giữa phần tử có giá trị lớn nhất và bé nhất trong nhóm đó. Nếu 1 nhóm có 0 hoặc 1 phần tử, chi phí của nhóm đó = 0.

Bờm rất giỏi trong việc tính toán, nhưng cậu ta lại không biết gì về lập trình. Bạn hãy giúp Bờm tìm cach phân chia N số vào M nhóm sao cho tổng chi phí là nhỏ nhất có thể.

Dữ liệu: vào từ file GROUP.INP gồm:

- Dòng đầu tiên chứa 2 số nguyên N, M (0<N, M <= 200).
- Dòng thứ 2 chứa N số nguyên a_1, a_2, ..., a_N tương ứng với N số nguyên thầy giáo đưa cho Bờm. (0<=a_i<=10^9).

Kết quả: ghi ra file GROUP.OUT gồm một dòng duy nhất chứa tổng chi phí nhỏ nhất có thể để chia N số vào M nhóm.

Ví dụ:

GROUP.INP	GROUP.OUT
8 3 5 2 3 10 7 2 6 8	4

Giải thích: Chia thành 3 nhóm (5 7 6 8), (10), và (2 3 2) chi phí từng nhóm tương ứng là 8-5 = 3, 10 – 10 = 0, 3 – 2 = 1 => tổng chi phí là 4.

Thuật toán:

Đầu tiên ta nhận thấy chi phí 1 nhóm là chênh lệch phần tử lớn nhất và bé nhất trong nhóm, vì thế nên nếu trong nhóm đó phần tử lớn nhất giá trị là b, phần tử bé nhất giá trị là a, thì nhóm đó sẽ chứa tất cả các số từ a đến b.

Sắp xếp dãy số theo thứ tự tăng dần => mỗi nhóm sẽ là 1 đoạn các số liên tiếp.

Ta quy hoạch động như sau: gọi f[i][j] là chi phí nhỏ nhất chia các số từ 1->i thành j nhóm -> xem xét nhóm thứ j chứa bao nhiêu phần tử: f[i][j] = min(f[k][j-1] + a[i]-a[k+1]) nhóm thứ j chứa các số a[k+1], a[k+2],..., a[i], max là a[i], min là a[k+1] với k = 0 ->i-1.

Code mẫu:

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 222;

int f[maxn][maxn];
int n, m;
int a[maxn];

int solve() {

     int largeNumber = 1000000001;

     sort(1+a, 1+n+a);

     for(int i=0; i<=m; i++)
          for(int j=0; j<=n; j++) {
               f[i][j] = largeNumber;
          }

    for(int i=0; i<=m; i++) {
        f[i][0] = 0;
    }

     for(int i=1; i<=n; i++) {
          f[1][i] = a[i] - a[1];
     }

     for(int i=2; i<=m; i++)
          for(int j=1; j<=n; j++) {
               for(int k=j-1; k>=0; k--) {
                    f[i][j] = min(f[i][j], f[i-1][k] + a[j]-
a[k+1]);
               }
          }
     return f[m][n];
}

int main() {

    freopen("Group.inp", "r", stdin);
    freopen("Group.out", "w", stdout);
    cin>>n>>m;
    for(int i=1; i<=n; i++) {
        cin>>a[i];
```

```
    }
    cout<<solve()<<endl;
    return 0;
}
```

Test: Link download: https://drive.google.com/drive/folders/1tJV_PVhPAQZ1ND2aXOXXYLeh2H0nXRUf?usp=sharing

Bài 6. CHU TRÌNH HOÁN VỊ

Bờm rất thích hoán vị. Một hoán vị N là một cách sắp xếp N số nguyên dương từ 1 đến N, mỗi số chỉ xuất hiện một lần. Ví dụ 1 3 5 2 4 là một hoán vị 5.

Phép nhân 2 hoán vị N $(a_1, a_2, a_3, \dots, a_n)$ và $(b_1, b_2, b_3, \dots, b_n)$ được định nghĩa như sau:

$(a_1, a_2, a_3, \dots, a_n) \times (b_1, b_2, b_3, \dots, b_n) = (a_{b1}, a_{b2}, a_{b3}, \dots, a_{bn})$

Ví dụ : (2 5 1 4 3) x (3 4 2 5 1) = (1 4 5 3 2)

Phép lũy thừa hoán vị được định nghĩa theo phép nhân hoán vị :

$(a_1, a_2, a_3, \dots, a_n)^2 = (a_1, a_2, a_3, \dots, a_n) \times (a_1, a_2, a_3, \dots, a_n)$

$(a_1, a_2, a_3, \dots, a_n)^k = (a_1, a_2, a_3, \dots, a_n) \times (a_1, a_2, a_3, \dots, a_n) \times \dots \times (a_1, a_2, a_3, \dots, a_n)$ (k phép nhân hoán vị).

Bờm nhận thấy có những số nguyên X mà $(a_1, a_2, a_3, \dots, a_n)^X = (a_1, a_2, a_3, \dots, a_n)$. Khi đó ta gọi X là một chu trình của $(a_1, a_2, a_3, \dots, a_n)$.

Với một một hoán vị ban đầu $(a_1, a_2, a_3, \dots, a_n)$. Bờm muốn tìm số nguyên dương K nhỏ nhất sao cho K+1 là một chu trình của $(a_1, a_2, a_3, \dots, a_n)$. Hãy giúp Bờm.

Dữ liệu: vào từ file HOANVI.INP gồm:

- Dòng đầu ghi số nguyên dương N ($1 <= N <= 2x10^5$).
- Dòng thứ 2 gồm N số nguyên dương khác nhau đôi một thể hiện hoán vị ban đầu.

Kết quả:

- Gồm một số nguyên M duy nhất là số dư của K cho 10^9+7.

Ví dụ:

HOANVI.INP	HOANVI.OUT	HOANVI.INP	HOANVI.OUT
5	6	5	1

5 3 2 1 4		1 2 3 4 5	

Thuật toán:

Nhận thấy nếu coi mỗi số là 1 đỉnh => mỗi đỉnh có duy nhất 1 đường đi vào và 1 đường đi ra => đồ thị sẽ tạo thành các vòng tròn.

Bài toán trở về tìm các vòng tròn của đồ thị, khi tìm được 1 vòng tròn độ dài x (gồm x đỉnh) => từ 1 vị trí bất kì trên vòng tròn, sau x phép nhân sẽ quay trở về đúng vị trí đó => tìm tất cả độ dài các vòng tròn, sau đó tìm BCNN của các độ dài => kq của bài toán.

Code mẫu:

```
{$M 100000000}
uses math;
const fi='HOANVI.INP';
      fo='HOANVI.OUT';
      maxn=1000000007;
var   a,vt,b:array[0..2000001] of longint;
      n,sl,m:longint;
      inq:array[0..2000001] of boolean;
procedure   docfile;
   var i:longint;
   begin
      assign(input,fi);
      reset(input);
      readln(n);
      for i:=1 to n do
         begin
            read(a[i]);
            vt[a[i]]:=i;
         end;
      assign(output,fo);
      rewrite(output);
   end;
procedure   dongfile;
   begin
      close(input);  close(output);
   end;
procedure   DFS(i:longint);
   begin
      inc(sl);
      if inq[vt[i]]=false then
         begin
            inq[vt[i]]:=true;
            DFS(vt[i]);
         end;
   end;
procedure   phantich(n:longint);
   var i,j,x:longint;
   begin
```

```
        i:=n; j:=2;
        while i<>1 do
           begin
              while (j*j<i) and (i mod j<>0) do inc(j);
              if j*j>i then
                 begin
                    b[i]:=max(b[i],i);
                    break;
                 end
              else
                 begin
                    x:=1;
                    while i mod j=0 do
                       begin
                          x:=x*j;
                          i:=i div j;
                       end;
                    b[j]:=max(b[j],x);
                 end;
           end;
     end;

  procedure   xuli;
     var i,j:longint;
         res,x:int64;
     begin
        res:=1;
        for i:=1 to n do
           if inq[i]=false then
              begin
                 sl:=0;  inq[i]:=true;
                 DFS(i);  phantich(sl);
              end;
        for i:=1 to n do
           if b[i]>0 then
              begin
                 x:=b[i];
                 res:=(res*x) mod maxn;
              end;
        writeln(res);
     end;
  BEGIN
     docfile;
     xuli;
     dongfile;
  END.
```

Test: Link download: https://drive.google.com/drive/folders/1S5tKX-gNi_JGRfOqCY1ldD8xbe4iDTs4?usp=sharing

Bài 7. DIVISORS

Nếu một số nguyên b chia hết cho a, thì a được gọi là ước của b.

Một số tự nhiên có đúng bốn ước dương được gọi là số kì diệu. Tèo định nghĩa một hàm D(n) với ý nghĩa là số lượng số kì diệu không vượt quá n.

Ví dụ, từ 1 đến 10 có 3 số kì diệu là 6, 4, 10, vì thế D(10) = 3.

Cho số nguyên dương n không vượt quá 10^8. Công việc của bạn là tính D(n).

Dữ liệu

- Gồm duy nhất một số nguyên dương n $(n \le 10^8)$.

Kết quả

- Đưa ra một số nguyên duy nhất là kết quả bài toán.

Ví dụ

DIVISORS.inp	DIVISORS.out
10	3

THUẬT TOÁN

Nhận thấy rằng, một số tự nhiên có đúng 4 ước dương khi và chỉ khi nó có dạng p^3 hoặc $p*q$ với p và q là hai số nguyên tố phân biệt. Do đó, ta có hướng giải quyết bài toán này như sau:

Dùng sàng nguyên số Eratosthenes để chọn ra các số nguyên tố từ 1 đến n. Mặt khác, ngay trong khi sàng, chúng ta có thể chuẩn bị luôn mảng nt[i] với ý nghĩa nt[i] là số nguyên tố lớn nhất là ước của i.

Sau khi chuẩn bị được mảng này, ta có thể dễ dàng kiểm tra được một số x có phải kì diệu không trong O(1). Thật vậy, x là số kì diệu khi và chỉ khi $x = nt[x]^3$ hoặc $\frac{x}{nt[x]}$ là một số nguyên tố khác nt[x].

Do đó, chúng ta có thể giải quyết bài toán này trong độ phức tạp O(n*log(n)) (Độ phức tạp của sàng nguyên tố).

Bài 8. BEAUNUM

Phúc rất thích số 4, 6 và 9. Chính vì thế anh ta quan niệm các số chỉ gồm các chữ số 4, 6 và 9 là những số hoàn hảo. Anh ấy cho rằng các số chia hết cho những số hoàn hảo này là những số đẹp. Một ngày nọ khi đang trên đường trở về nhà, anh ấy nhìn thấy những con số được ai đó viết lại trên xe buýt. Anh ấy chợt nghĩ ra một bài toán, đếm số lượng những số đẹp không vượt quá 10^n với 1 số tự nhiên n cho trước. Suy đi nghĩ lại anh ấy thấy bài toán này quá khó để giải được với n lớn. Chính vì vậy anh ấy đã nghĩ ra một bài toán mới tương tự nhưng dễ hơn. Anh định nghĩa lại về một số hoàn hảo. Một số hoàn hảo là một số không vượt quá 100 chỉ gồm các chữ số 4, 6 và 9. Một số được gọi là số đẹp nếu nó chia hết cho ít nhất

một số hoàn hảo nào đó. Cho số tự nhiên n $(1 \le n \le 18)$, tìm số lượng số đẹp không vượt quá 10^n.

Dữ liệu

- Dòng đầu tiên chứa số nguyên dương T $(1 \le T \le 10)$ là số lượng bài toán cần giải.
- T dòng tiếp theo, mỗi dòng chứa một số nguyên dương n tương ứng với yêu cầu bài toán $(1 \le n \le 18)$.

Kết quả

- Đưa ra kết quả bài toán trên T dòng ứng với T bài toán cần giải.

Ví dụ

BEAUNUM.inp	BEAUNUM.out
1 1	5

THUẬT TOÁN

Gọi A1, A2, …, A12 lần lượt là tập hợp chứa các số không vượt quá 10^n và chia hết cho 4, 6, …, 99 tương ứng. Việc tính số lượng các phần tử của mỗi tập hợp rất đơn giản, ví dụ A1 gồm các số chia hết cho 4 không vượt quá 10^n nên số lượng phần tử của A1 là (10^n div 4) + 1 (tính cả số 0). Như vậy bài toán trở thành tìm tập hợp là hợp của 12 tập hợp này. Xét bài toán đơn giản hơn với 2 tập hợp U và V. Ta có công thức như sau: | U ∪ V| = |U| + |V| - |U ∩ V| trong đó |U| là số lượng phần tử của U, U ∪ V là U hợp với V, U ∩ V là U giao với V. Công thức tổng quát với 12 phần tử như sau: |A1 ∪ A2 ∪ ….. ∪ A12| = |A1| + |A2| + … + |A12| - |A1 ∩ A2| - |A1 ∩ A3| - …. - |A11 ∩ A12| + |A1 ∩ A2 ∩ A3| + …. (1) Dễ dàng nhận thấy cứ lẻ cộng, chẵn trừ. Ta nhận thấy thêm số lượng phần tử không vượt quá 10^n chia hết cho X1, X2, …, Xk là: 10^n div (BCNN(X1, X2, …, Xk)) + 1 (2) Số lượng các phần tử trong biểu thức kết quả (1) ở trên chính bằng 2^12 -1 tương ứng với 2^12 trạng thái 0/1 của 12 tập hợp, ko tính trạng thái toàn số 0. Bây giờ xét từng trạng thái, với trạng thái t, xét những tập hợp được chọn (bit thứ i (với i=0..11) của trạng thái t = 1 tức là tập A(i+1) được chọn), tính BCNN của các số hoàn hảo đó. Tính số lượng phần tử của tập hợp này dựa vào công thức (2) ta tính được số lượng phần tử tương ứng, đếm số lượng số 1 của trạng thái t, nếu số lượng số 1 chẵn thì lấy lấy kết hiện tại trừ đi, nếu số lượng số 1 lẻ thì lấy kết quả hiện tại cộng vào hiện tại cộng vào. Code tham khảo: https://ideone.com/rcMDMi