

整数論テクニック集 (Ver. 1.0)

夕叢霧香 (Kirika Yuumura, @kirika_comp)

2018 年 3 月 12 日

はじめに

これは、競技プログラミングで使える整数論のテクニックをまとめた文章です。AtCoder の水色から赤下位程度を対象読者にしています。

整数論の問題は得てして「地頭ゲー」などと呼ばれやすいですが、実はそうではなく、知識を持っていることで解ける問題が多いです。しかし、その知識については、日本の競技プログラマーたちには、あまり知られていないように見えます。そのため、この文章を書くことにしました。

基本的に、実際に出題されている問題で利用されるテクニックを、実際の問題と共に紹介するという方式で書いていきます。そのため、ネタバレは非常に多いです。ご注意ください。

各セクションにはレベルを振ってあります。筆者が感じたおよその難易度を表しています。

コメント等は Twitter (@kirika_comp) までお願いします。

目次

1	基本: アルゴリズムについて	2
1.1	アルゴリズムにおける不変量	2
2	$\text{mod } p$ 上の計算 (モジュラー演算)	3
2.1	基本: 整数の加減乗除 (Lv. 1)	3
2.2	基本: 逆元 (Lv. 1)	3
2.3	基本: 分数の加減乗除 (Lv. 2)	4
3	二分累乗法 (Lv. 1)	5
3.1	モノイド的構造を見つけて二分累乗する (Lv. 2)	6
3.2	うまい変形で除算を回避する (Lv. 2)	7
4	Abundance で殴る (Lv. 2)	9
4.1	素数の abundance で殴る (Lv. 2)	9
5	$\text{mod } p$ のアルゴリズム	9
5.1	基礎知識	9

5.2	mod_sqrt, Tonelli-Shanks のアルゴリズム (Lv. 3)	10
6	群論 (Lv. 4)	14
6.1	群 (Lv. 4)	14
7	平方剰余の相互法則と 2 次体 (Lv. 4)	15
7.1	ルジャンドル記号	15
7.2	平方剰余の相互法則	16
7.3	2 次体	16
7.4	有限体	17
7.5	フロベニウス写像	17
7.6	応用例	17
8	mod p のアルゴリズム その 2	18
8.1	mod_sqrt その 2, Lehmer のアルゴリズム (Lv. 3)	18
8.2	mod_sqrt その 3, Cipolla のアルゴリズム (Lv. 4)	21
9	多項式を使うテク (Lv. 4)	22
9.1	高速フーリエ変換 (FFT) (Lv. 3)	22
9.2	フェルマーの小定理 (Lv. 3)	24
9.3	巡回群構造を用いた特殊な畳み込み (Lv. 4)	29
9.4	1 の 2^k 乗根 mod p を用いた畳み込み	30
10	ペル方程式 (Lv. 4)	31
11	単項イデアル整域 (Lv. 5)	33

1 基本: アルゴリズムについて

1.1 アルゴリズムにおける不変量

アルゴリズムを考える時は、ループや再帰呼び出しのある地点で、どのような条件が常に成り立っているか、どのような値が保たれているかを考察することが、アルゴリズムの理解を助けることがあります。このような条件や値のことを、**不変量 (invariant)** と呼びます。

例として、拡張ユークリッドの互除法と呼ばれるアルゴリズムの実装を与えます。拡張ユークリッドの互除法は、2 個の整数 a, b を入力とし、 $\gcd(a, b)$ に加えて $ax + by = \gcd(a, b)$ を満たす $\langle x, y \rangle$ も返すアルゴリズムです。

以下のコードで、各再帰呼び出しの `return` 直前に $ax + by = \gcd(a, b)$ が成り立っていることに注意してください。この性質によって、最終的な返り値が $ax + by = \gcd(a, b)$ を満たすことが、つまりこのアルゴリズムが正しいことが分かります。

ソースコード 1 extgcd.cpp

```
1 #include <iostream>
2 using namespace std;
```

```

3  typedef long long lint;
4
5  lint ext_gcd(lint a,lint b,lint&x,lint&y){
6      if(b==0){
7          x=1;y=0;return a;
8      }
9      lint q=a/b;
10     lint g=ext_gcd(b,a-q*b,x,y);
11     lint z=x-q*y;
12     x=y;y=z;
13     return g;
14 }
15
16 int main(){
17     lint a,b;
18     cin>>a>>b;
19     lint x,y;
20     lint g=ext_gcd(a,b,x,y);
21     cout<<a<<"□"
22         <<b<<"□"<<x<<"□"<<y
23         <<endl;
24     cout<<g<<endl;
25 }

```

2 mod p 上の計算 (モジュラー演算)

2.1 基本: 整数の加減乗除 (Lv. 1)

計算結果が大きすぎるため、適当な素数 p に対して mod p で出力せよ、という問題が結構あります。このような問題の場合は、最終結果を mod p するのではなく、途中結果も mod p することで、多倍長整数などを使わずに計算することができます。例えば、 $abc \bmod p$ を計算したい時は、代わりに $((ab \bmod p) \times c) \bmod p$ を計算する、というようにすることができます。

素数 p については、入力として与えられる場合と、問題に固有の値が設定される場合があります。後者の場合は、 p として $10^9 + 7$, $10^9 + 9$, 998244353 などの値が設定されることが多いです。これらの値は、有名な素数として覚えてしまいましょう。

なお除算については、次のサブセクションで説明する逆元を掛けることで実装することができます。

2.2 基本: 逆元 (Lv. 1)

p を素数とします。 $1 \leq r < p$ の時、 $rs \equiv 1 \pmod{p}$, $1 \leq s < p$ を満たす s がただ一つ存在します。これを r の mod p における逆元 (inverse element) と呼びます。逆元の計算には、以下の2種類の方法があります:

1. $r^{p-1} \equiv 1 \pmod{p}$ (フェルマーの小定理、定理 5.1) を利用して、 $r^{p-2} \bmod p$ を計算する。
2. 拡張ユークリッドの互除法を使う。

実装が単純なのは1. ですが、多くの場合で2.の方が高速に動作します。そのうえ、2.では p が素数であるという条件すら不要で、 r と p が互いに素であれば十分です。ライブラリを作る場合は、2.の方で作ると良いでしょう。

以下に両方の実装を与えます。

1. の実装を与えます。 $a^{-1} \bmod p$ を返します。

ソースコード 2 invmod-1.cpp

```
1 typedef long long lint;
2
3 lint powmod(lint x,lint e,lint mod){
4     lint prod=1%mod;
5     for(int i=63;i>=0;--i){
6         prod=prod*prod%mod;
7         if(e&1LL<=i)prod=prod*x%mod;
8     }
9     return prod;
10 }
11
12 // a の mod p における逆元を返す。p は素数で、a は p の倍数ではないことが要請される。
13 lint invmod(lint a,lint p){
14     return powmod(a%p,p-2,p);
15 }
```

2. の実装を与えます。 $a^{-1} \bmod m$ を返します。

ソースコード 3 invmod-2.cpp

```
1 typedef long long lint;
2
3 lint ext_gcd(lint a,lint b,lint&x,lint&y){
4     if(b==0){
5         x=1;y=0;return a;
6     }
7     lint q=a/b;
8     lint g=ext_gcd(b,a-q*b,x,y);
9     lint z=x-q*y;
10    x=y;y=z;
11    return g;
12 }
13
14 // a の mod m における逆元を返す。a と m は互いに素であることが要請される。
15 lint invmod(lint a,lint m){
16     lint x,y;
17     ext_gcd(a,m,x,y);
18     x%=m;
19     if(x<0)x+=m;
20     return x;
21 }
```

2.3 基本: 分数の加減乗除 (Lv. 2)

たまに、分数についての言及があることがあります。大抵以下のような形をしています。

答えは分数 A/B という形で表すことができ、 B は $10^9 + 7$ と互いに素であることが証明できる。このとき、 B の $\bmod (10^9 + 7)$ での逆元を B^{-1} として、 $A \times B^{-1} \bmod (10^9 + 7)$ を出力せよ。

これも、特別な配慮などはせずに、途中結果を $\bmod (10^9 + 7)$ で保持しておくだけで、計算が正しく行えます。

問題例

- Codeforces Round #465 (Div. 2) D. Fafa and Ancient Alphabet

COLUMN

専門用語を使うと、これは素数 p について自然に定まる環準同型 $\mathbb{Z} \rightarrow \mathbb{Z}/p\mathbb{Z}$ が、環準同型 $\mathbb{Z}_{(p)} \rightarrow \mathbb{Z}/p\mathbb{Z}$ に拡張できる、ということが出来ます (ただし、 $\mathbb{Z}_{(p)}$ は、 \mathbb{Z} のイデアル (p) による局所化と呼ばれるもので、分子が任意、分母が p の倍数でないような分数全体からなる環)。興味のある人は、「局所化」や「附値環」などの単語で調べてみてください。

3 二分累乗法 (Lv. 1)

二分累乗法とよばれる基本テクニックがあります。

(単位元を持ち、) 結合法則を満たす「掛け算」 $x \cdot y$ が定義されているような代数的構造 (モノイドと呼びます) の上で、 x の e 乗 $x^e = x \cdots x$ は $2\log_2 e$ 回以内の掛け算で計算できます。この方法を二分累乗法 (*exponentiation by squaring*) と呼びます。

二分累乗法には 2 種類の方法があります。一つ目は、 e の 2 進表記の小さい桁の方から計算をする方法で、もう一つは e の 2 進表記の大きい桁の方から計算をする方法です。整数の `mod` つき掛け算を例にして、両方のアルゴリズムを紹介します。

まず一つ目から紹介します。ソースコード 4 に実装を載せました。ループの各イテレーション終了時に $prod * cur^e = (answer) = x^e$ が成立していることに注意しましょう。最終的に $e = 0$ となって $prod$ に答えが入ります。

ソースコード 4 powmod-1.cpp

```
1 typedef long long lint;  
2  
3 lint powmod(lint x, lint e, lint mod){  
4     lint prod=1;  
5     lint cur=x;  
6     while(e>0){  
7         if(e%2==1) prod=prod*cur%mod;  
8         cur=cur*cur%mod;  
9         e/=2;  
10    }  
11    return prod;  
12 }
```

次に二つ目を紹介します。 e の上の桁から見ていくアルゴリズムです。ループの各イテレーション終了時に $prod^{2^i} * x^{e \% 2^i} = (answer) = x^e$ が成り立つことに注意しましょう。

ソースコード 5 powmod-2.cpp

```
1 typedef long long lint;  
2  
3 lint powmod(lint x, lint e, lint mod){  
4     lint prod=1%mod;  
5     for(int i=63; i>=0; --i){  
6         prod=prod*prod%mod;  
7         if(e & 1LL < i) prod=prod*x%mod;  
8     }
```

```

8   }
9   return prod;
10  }

```

3.1 モノイド的構造を見つけて二分累乗する (Lv. 2)

問題

$\cos \theta = \frac{d}{l}$ であるような θ に対して、 $\cos t\theta$ は有理数であることが証明できる。 $l \cos t\theta = \frac{p}{q}$ であるとき、 $pq^{-1} \bmod (10^9 + 7)$ を求めよ。

- 部分点 (15/100 点): t は 2 冪である。つまり、ある 0 以上の整数 p について $t = 2^p$ 。
- 満点 (100 点): $1 \leq t \leq 10^{18}$, t は整数。

(出典: CodeChef February Challenge 2018 (FEB18) » Broken Clock (BROCLK))

部分点解法は、2 倍角の公式 $\cos 2\theta = 2\cos^2 \theta - 1$ を利用して、 p 回の計算を行うことでできます。

問題は満点解法の方で、愚直にやると t 倍角の公式が必要になってきて、実質的に不可能です。そこで、ド・モアブルの公式 (de Moivre's formula)

$$\cos t\theta + i \sin t\theta = (\cos \theta + i \sin \theta)^t$$

を利用して、強引に冪乗公式に持っていくことを考えましょう。式の形から、 $\cos \theta + i \sin \theta$ なる数の計算、およびその累乗の計算ができれば良いことになります。ここで、以下のようにペアを用いて数を表現することになります:

$$\langle a, b \rangle \mapsto a + ib \sin \theta$$

これにより掛け算、それゆえ冪乗が、整数ペアの上の演算として実装できます。どのようにするかみていきましょう。

まず、 $\cos \theta + i \sin \theta$ はもちろん、 $\langle \cos \theta, 1 \rangle = \langle d/l, 1 \rangle$ として表現されます。掛け算についてですが、

$$\langle a, b \rangle \times \langle c, d \rangle = (a + ib \sin \theta)(c + id \sin \theta) = (ac - bd \sin^2 \theta) + i(ad + bc) \sin \theta = \langle ac + bd(\cos^2 \theta - 1), ad + bc \rangle$$

より、問題なく実装することができます。これによりべき乗も問題なく実装でき、問題が解けます。

ソースコード 6 BROCLK.cpp

```

1  #include<iostream>
2  using namespace std;
3  typedef long long lint;
4  typedef pair<lint,lint> pll;
5  const lint mod=1e9+7;
6
7  lint powmod(lint x,lint e){
8      lint c=1;
9      for(int i=63;i>=0;--i){
10         c=c*c%mod;
11         if(e&1LL<<i)c=c*x%mod;
12     }
13     return c;
14 }

```

```

15
16 pll mul_pll(pll a,pll b,lint c){
17     lint x=a.first*b.first%mod;
18     lint nx=a.second*b.second%mod;
19     x=(x+nx*c)%mod;
20     lint y=(a.first*b.second+a.second*b.first)%mod;
21     return pll(x,y);
22 }
23
24 pll pow_pll(pll a,lint e,lint c){
25     pll p(1,0);
26     for(int i=63;i>=0;--i){
27         p=mul_pll(p,p,c);
28         if(e&1LL<<i)p=mul_pll(p,a,c);
29     }
30     return p;
31 }
32
33 int main(){
34     int tt;
35     cin>>tt;
36     while(tt--){
37         lint l,d,t;
38         cin>>l>>d>>t;
39         lint cos=d*powmod(l,mod-2)%mod;
40         lint s2=(cos*cos%mod)+mod-1;
41         s2%=mod;
42         lint ans=pow_pll(pll(cos,1),t,s2).first;
43         cout<<ans*l%mod<<endl;
44     }
45 }

```

3.2 うまい変形で除算を回避する (Lv. 2)

問題

整数 A が、次のような 10 進表記で与えられる。

$$(A)_{10} = a_1^{L_1} \cdot a_2^{L_2} \cdot \dots \cdot a_N^{L_N}$$

ここで、 a_i は 10 進表記で与えられた整数、 L_i は整数である。また $a_i^{L_i}$ は、 a_i を文字列としてみなして、 L_i 個連結したものを表し、 $s \cdot t$ は文字列 s, t の連結を表す。このとき、 A を B で割った余りを求めよ。

制約: $1 \leq N \leq 10000, 1 \leq a_i \leq 10^9, 1 \leq L_i \leq 10^9$

- 部分点 (99/100 点): $B = 10^9 + 7$ 。
- 満点 (100 点): $1 \leq B \leq 10^9 + 7$ 。 B は素数とは限らない。

(出典: ARC020 C - A mod B Problem)

B が素数の場合、 A は等比数列の総和の公式を使って、以下のような閉じた式の形で書けるので、計算するこ

とは簡単です。ここで、 b_i は a_i の桁数で、

$$f(y, z) := 1 + y + \dots + y^{z-1} = \frac{y^z - 1}{y - 1}$$

と定義します。

$$A = a_N f(10^{b_N}, l_N) + 10^{b_N \times l_N} (a_{N-1} f(10^{b_{N-1}}, l_{N-1}) + 10^{b_{N-1} \times l_{N-1}} (\dots))$$

問題は B が素数でない場合です。 $f(y, z)$ の計算の中で、 $y - 1$ による除算を行っていますが、 $y - 1$ と B が互いに素でない場合に、これは失敗します。これを回避するために、 y^z を二分累乗法で計算するのを諦め、 $f(y, z)$ を直接二分累乗法に似た方法で計算することを考えましょう。以下の等式が成立します:

$$f(y, 2z) = 1 + y + \dots + y^{2z-1} = (1 + y + \dots + y^{z-1})(1 + y^z) = (1 + y^z)f(y, z)$$

$$f(y, z+1) = 1 + y + \dots + y^z = 1 + y(1 + y + \dots + y^{z-1}) = 1 + yf(y, z)$$

これによって、 z の偶奇に応じて場合分けしながら再帰を行うことで、 $f(y, z)$ の値を除算なしで計算することができます。

ソースコードは以下のようになります。<https://arc020.contest.atcoder.jp/submissions/2141049> です。

ソースコード 7 ARC020C.cpp

```

1 #include<iostream>
2 #include<vector>
3 using namespace std;
4 typedef long long lint;
5 #define rep(i,n)for(int i=0;i<(int)(n);++i)
6
7 lint powmod(lint x,lint e,lint mod){
8     lint prod=1%mod;
9     for(int i=63;i>=0;--i){
10         prod=prod*prod%mod;
11         if(e&1LL<<i)prod=prod*x%mod;
12     }
13     return prod;
14 }
15
16 lint f(lint y,lint z,lint mod){
17     if(z==0)return 0;
18     if(z%2==0){
19         lint fac=1+powmod(y,z/2,mod);
20         return fac*f(y,z/2,mod)%mod;
21     }
22     return (1+y*f(y,z-1,mod))%mod;
23 }
24
25 int main(){
26     int n;
27     cin >> n;
28     vector<lint> a(n),l(n);
29     rep(i,n)cin>>a[i]>>l[i];
30     lint mod;
31     cin>>mod;
32     lint ans=0;

```



```

33 rep(i,n){
34     lint enc=1;
35     while(a[i]>=enc)enc*=10; // enc=10^{b_i}
36     ans=ans*powmod(enc%mod,l[i],mod)%mod;
37     ans=(ans+a[i]*f(enc%mod,l[i],mod))%mod;
38 }
39 cout<<ans<<endl;
40 }

```

4 Abundance で殴る (Lv. 2)

4.1 素数の abundance で殴る (Lv. 2)

問題

正の整数 p, y が与えられる。2 以上 p 以下のどのような整数 i についても $z = ki$ ($k \geq 2$) と表されないような、 y 以下の最大の整数 z を求めよ。

制約: $2 \leq p \leq y \leq 10^9$

(出典: Codeforces Round #467 (Div. 2) B. Vile Grasshoppers)

まず、 z が素数なら問題の条件は確実に満たされます。 y 付近の素数の密度は $\Theta(1/\log y)$ なので、 $O(\log y)$ 個程度調べれば、一つは素数が見つかる計算です。与えられた z が問題の条件を満たすかどうかの判定は $O(\min(p, \sqrt{y}))$ でできるので、 y から大きい順に調べていくことで、時間計算量 $O(\min(p, \sqrt{y}) \log y)$ で解くことができます。

5 mod p のアルゴリズム

5.1 基礎知識

5.1.1 フェルマーの小定理 (Lv. 1)

定理 5.1. p が素数で $a \not\equiv 0 \pmod{p}$ のとき、 $a^{p-1} \equiv 1 \pmod{p}$ が成立する。

5.1.2 平方剰余 (Lv. 3)

$a \equiv x^2 \pmod{p}$ となる x が存在する場合、整数 a を mod p における 平方剰余 (quadratic residue)、そうでない場合 a を 平方非剰余 (quadratic non-residue) と呼びます。 p が奇素数の時、1 から $p-1$ までの整数の中の、平方剰余と平方非剰余の割合は 1:1 です。また、 $a \not\equiv 0 \pmod{p}$ のとき $a^{(p-1)/2}$ は mod p で 1 か -1 かのどちらかと合同ですが、 a が平方剰余のとき 1、平方非剰余のとき -1 です。

例 5.2. $p = 13$ の場合を考えます。このとき、mod 13 における平方剰余のうち、0 以上 12 以下のものは $0 = 0^2, 1 = 1^2, 3 = 4^2, 4 = 2^2, 9 = 3^2, 10 = 6^2, 12 = 5^2 \pmod{13}$ の 7 個です。0 を除外すると 1,3,4,9,10,12 の 6 個で、1 から 12 までの整数のうち、ちょうど半分が平方剰余、もう半分が平方非剰余です。なお、適当な平方非剰余 z をとると、0 以外の平方剰余に z を掛けたものはすべて平方非剰余です。例えば、 $z = 2$ とすると、1,3,4,9,10,12 に 2 を掛けて mod 13 したもの (2,6,8,5,7,11) はすべて平方非剰余です。

5.2 mod_sqrt, Tonelli-Shanks のアルゴリズム (Lv. 3)

5.2.1 問題

ある x について $a \equiv x^2 \pmod{p}$ が成り立つ a が与えられる。この時、 x を求めよ。

5.2.2 解法

まず、簡単のため $p = 2$ の場合を除外します (このときは $a^2 \equiv a \pmod{2}$ なので簡単)。また $a \equiv 0 \pmod{p}$ の場合も除外します ($0^2 = 0$ なので簡単)。 p が mod 4 で 3 の時は簡単です。 $x \equiv a^{(p+1)/4}$ とすると、 $x^2 \equiv a^{(p+1)/2} = a^{(p-1)/2} a \pmod{p}$ です。ここで、 $a \equiv y^2$ となる y が存在するので、 $a^{(p-1)/2} \equiv y^{p-1} \equiv 1 \pmod{p}$ です。だから、 $x^2 \equiv a$ が成り立ちます。

p が mod 4 で 1 の時は結構複雑なことをします。ここでは Tonelli-Shanks の方法と呼ばれるアルゴリズムを説明します。

5.2.3 Tonelli-Shanks (トネリ-シャンクス) のアルゴリズム

Reference: https://en.wikipedia.org/wiki/Tonelli%E2%80%93Shanks_algorithm

注意: 以下の疑似コードでは代入は全部同時に行います。特に 5. で、 t に代入する値は前の c によって決まります。

注意 2: 本来の Tonelli-Shanks とは違いますが、不変量を考えることで筆者が復元できたのが以下のアルゴリズムなので、こちらの方が理解しやすいと思います。(効率が悪い)

Algorithm 1 単純化された Tonelli-Shanks のアルゴリズム

入力: $p(\geq 3)$: 奇素数, $a(\not\equiv 0 \pmod{p})$: 平方剰余

出力: $r^2 \equiv a \pmod{p}, 1 \leq r \leq p-1$ を満たす整数 r

$p = q \times 2^s + 1$ とします。 ($s \geq 1, q$ は奇数)

1. $z^{(p-1)/2} \equiv -1 \pmod{p}$ となるような z を選ぶ。このような z は確率 $1/2$ でヒットするため、何個か試せば必ず見つかる。
 2. $m := s, c := z^q, t := a^q, r := a^{(q+1)/2}$ とする。以降不変量 $r^2 \equiv at \pmod{p}, t^{2^{m-1}} \equiv 1 \pmod{p}, c^{2^{m-1}} \equiv -1 \pmod{p}$ を崩さないように注意して操作する。
 3. 以降 m を減らしていく。 m が 1 なら終了し、 $r \bmod p$ を返す。そうでなければ、 $t^{2^{m-2}} \equiv 1 \pmod{p}$ なら
 4. へ、そうでなければ 5. へ行く。
 4. $c := c^2 \pmod{p}, m := m-1, 6.$ へ行く。
 5. $c := c^2 \pmod{p}, t := c^2 t \pmod{p}, r := cr \pmod{p}, m := m-1$ を全て同時に代入する、 $6.$ へ行く。
 6. 3. へ行く。
-

終了時には $m = 1$ なので、 $t \equiv 1 \pmod{p}$ になっているはずで、そのときの r が求める値です。(不変量 $r^2 \equiv at \pmod{p}$ に注意。)

C++ での実装はソースコード 8 のようになります。

ソースコード 8 tonelli-shanks.cpp

```
1 #include<random>
2 using namespace std;
```

```

3 typedef long long lint;
4
5 lint powmod(lint a,lint e,lint p){
6     lint r=1;
7     for(int i=63;i>=0;--i){
8         r=r*r%p;
9         if(e&1LL<<i)r=r*a%p;
10    }
11    return r;
12 }
13
14 //p:素数, a は 0 でなく、平方剰余
15 lint simplified_tonelli_shanks(lint p,lint a){
16     mt19937 mt;
17     if(powmod(a,(p-1)/2,p)!=1)return -1;
18     lint q=p-1;
19     lint m=0;
20     while(q%2==0)q/=2,m++;
21     lint z;
22     do{
23         z=mt()%p;
24     }while(powmod(z,(p-1)/2,p)!=p-1);
25     lint c=powmod(z,q,p);
26     lint t=powmod(a,q,p);
27     lint r=powmod(a,(q+1)/2,p);
28     for(;m>1;--m){
29         lint tmp=powmod(t,1<<(m-2),p);
30         if(tmp!=1)
31             r=r*c%p,t=t*(c*c%p)%p;
32         c=c*c%p;
33     }
34     return r;
35 }

```

例を挙げて見ていきましょう。 $p = 41, a = 8$ とします。

$p = 5 \cdot 2^3 + 1$ なので、 $q = 5, s = 3$ です。 z として、ここでは 7 をとります。

$m := 3, c := 7^5 = 16807 \equiv 38, t := 8^5 = 32768 \equiv 9, r \equiv 8^3 = 512 \equiv 20$ となります。(mod 41 は適宜省略)

m	c	t	r
3	38	9	20
2	9	40	22
1	40	1	34

よって、 $x \equiv \pm 34 (\equiv \mp 7)$ が答えになります。

以上のアルゴリズムには、4. のパートに無駄があります。4. では c と m しか変更していないので、 $t^{2^i} \not\equiv 1 \pmod{p}$ となる最大の i が見つけれられれば、4. の操作をまとめることができます。このアイデアを使うのが、本来の Tonelli-Shanks のアルゴリズム (Algorithm 2) です。

以上を使うと、次の問題が解けます。

Algorithm 2 Tonelli-Shanks のアルゴリズム

入力: $p(\geq 3)$: 奇素数, $a(\not\equiv 0 \pmod p)$: 平方剰余

出力: $r^2 \equiv a \pmod p$ を満たす r

$p = q \times 2^s + 1$ とします。 ($s \geq 1, q$ は奇数)

1. $z^{(p-1)/2} \equiv -1 \pmod p$ となるような z を選ぶ。このような z は確率 $1/2$ でヒットするため、何個か試せば必ず見つかる。

2. $m := s, c := z^q, t := a^q, r := a^{(q+1)/2}$ とする。以降不変量 $r^2 \equiv at \pmod p, t^{2^{m-1}} \equiv 1 \pmod p, c^{2^{m-1}} \equiv -1 \pmod p$ を崩さないように注意して操作する。

3. 以降 m を減らしていく。 m が 1 なら終了。そうでなければ、 $t^{2^k} \equiv 1 \pmod p$ となるような最小の整数 k を求める。これは二分累乗法でできる。この k は $1 \leq k \leq m-1$ を満たす。

4. $c := c^{2^{m-k}} \pmod p, t := c^{2^{m-k}} t \pmod p, r := c^{2^{m-k-1}} r \pmod p, m := k$ を全て同時に代入する, 3. へ行く。

問題

相異なることが保証されている n 個の整数 a_1, \dots, a_n が与えられる。これを並べ替えて、 $x \bmod m, (x+d) \bmod m, \dots, (x+(n-1)d) \bmod m$ と表せるかどうか判定せよ。表せる場合、 x, d ($0 \leq x < m, 0 \leq d < m$ を満たす) を復元せよ (複数ある場合はどれでもよい)。表せない場合、 -1 を出力せよ。

制約: $2 \leq m \leq 10^9 + 7$, m は素数、 $1 \leq n \leq 10^5, 0 \leq a_i < m$

(出典: Codeforces Round #395 (Div. 1) C. Timofey and remodeling)

並べ替えても平均と分散は変わらないため、これを特徴量とすることができます。平均は $x + d(n-1)/2$, 分散は $(n-1)(n+1)d^2/12$ です。 ($m \neq 2, 3$ かつ $n \neq 1, m-1, m$ ならば) ここから d が復元でき、 x も復元できます。最後に復元した x, d が妥当か調べれば OK です。

$m = 2, 3$ の場合は全探索でよいでしょう。 a_i たちは相異なるので、 $n \leq m$ であることに注意しましょう。また $n = 1, m-1$ の場合はそれぞれ $d = 0, 1$ とするとよく、 $n = m$ の場合は $\langle x, d \rangle = \langle 0, 1 \rangle$ が常に正答を与えます。

ソースコードは以下の通りです。 <http://codeforces.com/contest/763/submission/35830261> です。

ソースコード 9 CF395-1C.cpp

```
1 #include<algorithm>
2 #include<cstdio>
3 #include<vector>
4 #include<random>
5 using namespace std;
6 typedef long long lint;
7 #define rep(i,n)for(int i=0;i<(int)(n);++i)
8
9 lint powmod(lint x,lint e,lint mod){
10     lint prod=1%mod;
11     for(int i=63;i>=0;--i){
12         prod=prod*prod%mod;
13         if(e&1LL<<i)prod=prod*x%mod;
14     }
15     return prod;
16 }
17
18 //p:素数,aは0でなく、平方剰余
19 lint simplified_tonelli_shanks(lint p,lint a){
```

```

20  mt19937 mt;
21  if(powmod(a,(p-1)/2,p)!=1)return -1;
22  lint q=p-1;
23  lint m=0;
24  while(q%2==0)q/=2,m++;
25  lint z;
26  do{
27      z=mt()%p;
28  }while(powmod(z,(p-1)/2,p)!=p-1);
29  lint c=powmod(z,q,p);
30  lint t=powmod(a,q,p);
31  lint r=powmod(a,(q+1)/2,p);
32  for(;m>1;--m){
33      lint tmp=powmod(t,1<<(m-2),p);
34      if(tmp!=1)
35          r=r*c%p,t=t*(c*c%p)%p;
36      c=c*c%p;
37  }
38  return r;
39 }
40
41 void add(lint &x,lint y,lint m){
42     x=(x+y)%m;
43 }
44
45 int main(){
46     lint m;
47     int n;
48     scanf("%11d",&m,&n);
49     vector<lint>a(n);
50     rep(i,n)scanf("%11d",&a[i]);
51     if(m<=3){
52         // brute force
53         sort(a.begin(),a.end());
54         rep(x,m){
55             rep(d,m){
56                 vector<lint> b(n);
57                 rep(i,n)b[i]=(x+d*i)%m;
58                 sort(b.begin(),b.end());
59                 if(a==b){
60                     printf("%d_%d\n",x,d);
61                     return 0;
62                 }
63             }
64         }
65         puts("-1");
66         return 0;
67     }
68     if(n==m){
69         puts("0_1");
70         return 0;
71     }
72     lint sum=0;
73     lint sqsum=0;
74     rep(i,n){
75         add(sum,a[i],m);
76         add(sqsum,a[i]*a[i],m);

```

```

77 }
78 lint avg=sum*powmod(n,m-2,m)%m;
79 lint vari=sqsum*powmod(n,m-2,m)%m;
80 add(vari,avg*(m-avg),m);
81 lint d;
82 if(2<=n&&2<=m-2){
83     lint d2=vari*12%m;
84     d2=d2*powmod(((lint)n*n-1)%m,m-2,m)%m;
85
86     d=simplified_tonelli_shanks(m,d2);
87     if(d==1){
88         puts("-1");
89         return 0;
90     }
91 } else if(n==1){
92     d=0;
93 } else{
94     d=1;
95 }
96 lint x=avg;
97 lint tmp=(n-1)*d%m;
98 tmp=(tmp*(m+1)/2)%m;
99 add(x,m-tmp,m);
100 vector<lint> b(n);
101 rep(i,n)b[i]=(x+d*i)%m;
102 sort(a.begin(),a.end());
103 sort(b.begin(),b.end());
104 if(a==b){
105     printf("%11d_1%11d\n",x,d);
106 } else{
107     puts("-1");
108 }
109 }

```

COLUMN

群論的なアプローチをすると、Algorithm 1 についてもっと綺麗な見方が得られます。 $(\mathbb{Z}/p\mathbb{Z})^* \cong \mathbb{Z}/(p-1)\mathbb{Z} \cong (\mathbb{Z}/2^s\mathbb{Z}) \times (\mathbb{Z}/q\mathbb{Z})$ の 2 冪成分 (2-Sylow 部分群) $H \cong \mathbb{Z}/2^s\mathbb{Z}$ を考えます。このとき、 z と t は H の元であることがわかります。 t が 1 になるように、うまく $H^2 := \{h^2 \mid h \in H\}$ の元で調整しているわけです。

このあたりの話題は、この後のセクション 6 で詳しく紹介します。

6 群論 (Lv. 4)

6.1 群 (Lv. 4)

結合法則を満たす乗算と、乗算の逆元と、乗算の単位元が定義された集合のことを群 (group) と呼びます。例えば、整数全体の集合 \mathbb{Z} や、有理数全体の集合 \mathbb{Q} は、加法を演算とした群です。 n 要素の置換全体の集合は $n!$ 個の要素を持ちますが、この集合は置換の合成を演算として群をなします。この群を対称群 (symmetric group) と呼び、 S_n と表記します。また、 n を正の整数として、可逆な n 次実数係数行列全体は、行列積という演算で群をなします。

群の演算は、交換法則 $a * b = b * a$ を満たさなくても構いません。満たす場合、その群はアーベル群 (可換群) (abelian group (commutative group)) と呼ばれます。 \mathbb{Z} や \mathbb{Q} はアーベル群の例です。3 次以上の対称群や、 $n \geq 2$ のときの n 次可逆実数係数行列全体の群は、アーベル群でない群の例です。

群 G の部分集合であって、 G の演算でまた群となるものを、 G の部分群 (subgroup) と呼びます。たとえば、偶数だけを集めた集合 $2\mathbb{Z} := \{2x \mid x \in \mathbb{Z}\}$ に加法という演算を入れた群は、 \mathbb{Z} の部分群です。

6.1.1 巡回群 (Lv. 4)

G が巡回群 (cyclic group) であるとは、ある要素 $g \in G$ が存在して、任意の要素 $x \in G$ が $x = g^i = g \cdot g \cdots g$ の形で表せることをいいます。この g のことを生成元 (generator) と呼びます。整数 m に対して、整数を m で割った余り全体の集合を $\mathbb{Z}/m\mathbb{Z}$ と書きます。 $\mathbb{Z}/m\mathbb{Z}$ は、加法を演算として、群の構造を持ちます。これは巡回群の一例となっています。これ以降、 $\mathbb{Z}/m\mathbb{Z}$ の元の演算については、合同式の \equiv と $(\text{mod } m)$ を適宜省略します。たとえば、

$$x = 4 \in \mathbb{Z}/10\mathbb{Z}, y = 7 \in \mathbb{Z}/10\mathbb{Z} \text{ のとき、} x + y = 1$$

などという数式を書くことがあります。

6.1.2 乗法群 (Lv. 4)

$\mathbb{Z}/m\mathbb{Z}$ の要素のうち、 m と互いに素な元には乗法の逆元が存在することは、2.2 で確かめました。これらの元からなる集合を $(\mathbb{Z}/m\mathbb{Z})^*$ と表記し、 $\mathbb{Z}/m\mathbb{Z}$ の乗法群 (multiplicative group) と呼びます。例えば、 $(\mathbb{Z}/7\mathbb{Z})^* = \{1, 2, 3, 4, 5, 6\}$, $(\mathbb{Z}/10\mathbb{Z})^* = \{1, 3, 7, 9\}$ です。この演算についても、今後は \equiv と $(\text{mod } m)$ は適宜省略します。たとえば、 $(\mathbb{Z}/7\mathbb{Z})^*$ において $3^4 = 81 = 4$ である、というようにです。

m が素数 p のとき、 $(\mathbb{Z}/p\mathbb{Z})^*$ は巡回群であることが知られています。^{*1} $(\mathbb{Z}/p\mathbb{Z})^*$ の生成元のことを、 $\text{mod } p$ における原始根 (primitive root) と呼びます。例えば、 $(\mathbb{Z}/7\mathbb{Z})^* = \{1, 2, 3, 4, 5, 6\}$ について、 $3^0 = 1, 3^1 = 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5$ であるため、3 は $\text{mod } 7$ における原始根です。

ここで、1 から $p-1$ までの整数の中の $\text{mod } p$ の平方剰余全体の集合を、 H と書くことにして、 H の群論的性質を見てみましょう。 H は $(\mathbb{Z}/p\mathbb{Z})^*$ の部分群です。5.1.2 でみたように、平方非剰余 $z \notin H$ を適当にとると、 $zH := \{zh \mid h \in H\}$ と H は共通部分を持たず、また $|H| = |zH|$, $H \cup zH = (\mathbb{Z}/p\mathbb{Z})^*$ です。このように、群論では、部分群を用いた類別が綺麗に成功し、要素を性質によって等分することができます。この場合、1 から $p-1$ までの整数の中に、平方剰余と平方非剰余が同じ個数だけ存在することが、このように表現できます。

7 平方剰余の相互法則と 2 次体 (Lv. 4)

7.1 ルジャンドル記号

ルジャンドル記号とは、以下で定義されるものです。

$$\left(\frac{a}{p}\right) := \begin{cases} 1 & a \text{ が平方剰余のとき} \\ -1 & a \text{ が平方非剰余のとき} \\ 0 & a \equiv 0 \pmod{p} \text{ のとき} \end{cases}$$

^{*1} $(\mathbb{Z}/m\mathbb{Z})^*$ が巡回群となる 2 以上の整数 m は、 $m = 2, 4, p^e, 2p^e$ (p は奇素数、 e は正の整数) の形に限られることが知られています。

命題 7.1.

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}$$

命題 7.2. ルジャンドル記号は乗法的である。つまり、

$$\left(\frac{a}{p}\right)\left(\frac{b}{p}\right) = \left(\frac{ab}{p}\right)$$

系 7.3. 対応 $a \mapsto \left(\frac{a}{p}\right)$ は群準同型 $(\mathbb{Z}/p\mathbb{Z})^* \rightarrow \{+1, -1\}$ を定める。

7.2 平方剰余の相互法則

以下の定理が成り立つことが知られています。

定理 7.4 (平方剰余の相互法則). $p, q \geq 3$ を奇素数とする。このとき、以下が成立する。

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2} \times \frac{q-1}{2}}$$

定理 7.5 (補充法則). $p \geq 3$ を奇素数とする。このとき、以下が成立する。

$$\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}, \left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$$

これを利用することで、ルジャンドル記号を計算できます。

例 7.6. $p \neq 2, 3$ を、2, 3 以外の素数とします。このとき、 $\left(\frac{3}{p}\right)$ は、 p を 12 で割った余りで完全に決まります。

$$\left(\frac{3}{p}\right) = (-1)^{\frac{p-1}{2}} \left(\frac{p}{3}\right)$$

ここで、

$$\left(\frac{p}{3}\right) = \begin{cases} 1 & \text{if } p \equiv 1 \pmod{3} \\ -1 & \text{if } p \equiv 2 \pmod{3} \end{cases}$$

なので、

$$(-1)^{\frac{p-1}{2}} = \begin{cases} 1 & \text{if } p \equiv 1 \pmod{4} \\ -1 & \text{if } p \equiv 3 \pmod{4} \end{cases}$$

と合わせ、

$$\left(\frac{3}{p}\right) = \begin{cases} 1 & \text{if } p \equiv 1, 11 \pmod{12} \\ -1 & \text{if } p \equiv 5, 7 \pmod{12} \end{cases}$$

が得られます。

7.3 2 次体

体 K に、 $\theta \notin K$ なる要素 θ を追加することを、 θ を添加するといひ、そうしてできた体を $K(\theta)$ と表記します。有理数体 \mathbb{Q} に、平方数でない整数 d の平方根 \sqrt{d} を添加した体 $\mathbb{Q}(\sqrt{d})$ を、2 次体 (quadratic field) と呼びます。2 次体は古くから性質が研究されていて、理論も深いものがあります。このセクションおよびこの記事の後の方では、競技プログラミングに関連する部分を、主に取り上げたいと思います。

7.4 有限体

任意の素数 p と正の整数 e に対して、 p^e 要素の有限体が存在します。逆に、有限体の要素数は、必ず p^e の形で表せます。このような有限体は、一意に存在します。これを $\text{GF}(p^e)$ と表記することにします。

$\text{GF}(p^e)$ は、 $\text{GF}(p)$ にある要素を添加することで得ることができます。つまり、ある要素 θ に対して、 $\text{GF}(p^e) \cong \text{GF}(p)(\theta)$ です。

7.5 フロベニウス写像

$\text{Frob}: \text{GF}(p^e) \rightarrow \text{GF}(p^e), \text{Frob}(x) := x^p$ をフロベニウス写像と呼びます。

命題 7.7. Frob は e 回適用すると元に戻る。つまり、 $\text{Frob}^e(x) = x$ 。

証明. $\text{GF}(p^e)$ の元 $x \in \text{GF}(p^e)$ は、すべて $x^{p^e} = x$ を満たすことから、明らかです。 \square

命題 7.8. $x, \text{Frob}(x), \text{Frob}^2(x), \dots, \text{Frob}^{e-1}(x)$ は全て共役。つまり、ある $\text{GF}(p)$ 係数の e 次多項式 $f(t)$ が存在して、 $x, \text{Frob}(x), \text{Frob}^2(x), \dots, \text{Frob}^{e-1}(x)$ はすべて $f(t) = 0$ の根である。

7.6 応用例

問題

数列 $a_0 = 2, a_{n+1} = a_n(a_n + 4)$ がある。このとき、素数 M に対して、 $a_N \bmod M$ を求めよ。

(出典: yukicoder No.613 Solitude by the window)

この問題は、一般項を求めるところが一番難しく、一般項を求めた後は数論的な考察を進めるだけで解けます。ここでは、 $a_n = (2 + \sqrt{3})^{2^n} + (2 - \sqrt{3})^{2^n} - 2$ であることがわかっているとして、この状態から問題を解いてみましょう。 $a_n \bmod M$ が計算できれば良いです。

$(2 + \sqrt{3})^{2^n} \bmod M$ が計算できれば万事解決です。簡単のため、 M が 2 でも 3 でもないとしましょう。3 が $\bmod M$ で平方剰余なら (つまり $(\frac{3}{M}) = 1$ なら)、議論は $\text{GF}(M)$ の中で完結できます。3 が $\bmod M$ で平方非剰余 (つまり $(\frac{3}{M}) = -1$) の場合を考えます。このとき、 $\text{GF}(M)$ に $\sqrt{3}$ を添加して拡大したものは、 $\text{GF}(M^2)$ と同型になります。

$$\text{GF}(M)(\sqrt{3}) \cong \text{GF}(M^2)$$

ここで、 $\text{Frob}(2 + \sqrt{3}) = (2 + \sqrt{3})^M \in \text{GF}(M^2)$ がどのような元になるかを考えてみましょう。 $2 + \sqrt{3}$ の共役は自分自身と $2 - \sqrt{3}$ のみなので、 $\text{Frob}(2 + \sqrt{3}) = 2 - \sqrt{3}$ でなければなりません。これから、 $(2 + \sqrt{3})^{M+1} = (2 - \sqrt{3})(2 + \sqrt{3}) = 1$ であることが分かります。

よって、 $M \equiv 1, 11 \pmod{12}$ なら周期 $M - 1$ 、それ以外なら周期 $M + 1$ だと思って実装することができます。注意すべきなのは $M = 2, 3$ のケースで、今回は周期が $M \pm 1$ のどちらだと思っても偶然うまく動作しますが、うまく行かない問題もあるので、注意が必要です。

この問題は筆者が writer でした。ソースコードを以下に載せます。<https://yukicoder.me/submissions/239504> からアクセスすることもできます。(今回限り Java です。)

```

1  import java.util.*;
2
3  class Main {
4      static long powerMod(long x, long exponent, long m) {
5          long prod = 1;
6          for (int i = 63; i >= 0; --i) {
7              prod = (prod * prod) % m;
8              if ((exponent & 1L << i) != 0) {
9                  prod = (prod * x) % m;
10             }
11         }
12         return prod;
13     }
14     public static void main(String[] args) {
15         Scanner scan = new Scanner(System.in);
16         long n=scan.nextLong();
17         long m=scan.nextLong();
18         long p=m==2||m%12==1||m%12==11?m-1:m+1;
19         long e=powerMod(2,n,p);
20         long x=1,y=0;
21         for(int i=31;i>=0;--i){
22             long z=(x*x+3*y*y)%m,w=2*x*y%m;
23             x=z;y=w;
24             if((e&1L<<i)!=0){
25                 z=(2*x+3*y)%m;w=(x+2*y)%m;
26                 x=z;y=w;
27             }
28         }
29         System.out.println((2*x+m-2)%m);
30     }
31 }

```

8 mod p のアルゴリズム その 2

8.1 mod_sqrt その 2, Lehmer のアルゴリズム (Lv. 3)

定理 8.1. p を奇素数とする。mod p の 0 以外の平方剰余を $a_1, \dots, a_{\frac{p-1}{2}}$ とする。このとき

$$(x - a_1) \times (x - a_2) \times \cdots \times (x - a_{\frac{p-1}{2}}) \equiv x^{\frac{p-1}{2}} - 1 \pmod{p}$$

が成り立つ。

証明. 定理 9.1 と同じ論法が使えます。 □

これを利用したアルゴリズム (Algorithm 3) を紹介します。このアルゴリズムは Lehmer によるものです。

1. d の選択にかかる時間の期待値が定数で抑えられることは、以下の定理で証明できます。

定理 8.2 ([1, Theorem 3.2]). p を奇素数、 $a \in (\mathbb{Z}/p\mathbb{Z})^*$ を平方剰余とする。このとき、 $c \in \mathbb{Z}/p\mathbb{Z}$ が一様にランダムに選ばれるならば、確率 $(p-1)/2p$ で $\left(\frac{c^2-a}{p}\right) = -1$ が成立する。

証明. まず、 $x^2 - y^2 = a$ を満たす $\langle x, y \rangle \in (\mathbb{Z}/p\mathbb{Z})^2$ の個数を計算しましょう。以下の事実に着目します。

Algorithm 3 [1, Algorithm 3.1]

入力: p : 奇素数, a : $1 \leq a < p$ を満たす平方剰余

出力: $r^2 = a$ を満たす r

1. $c \in \mathbb{Z}/p\mathbb{Z}$ をランダムに選び、 $\left(\frac{c^2-a}{p}\right) = -1$ となるようにする。
 2. 多項式 $(\mathbb{GF}(p)[x]$ の元) の最大公約数 $\gcd(x^{\frac{p-1}{2}} - 1, (x-c)^2 - a)$ を計算する。答えは 1 次式 $ux + v$ となる。 a の平方根は $\pm(c + u^{-1}v)$ なので、 $+$ か $-$ のうちどちらかを出力する。
-

$G := \{\langle x, y, a \rangle \in (\mathbb{Z}/p\mathbb{Z})^2 \times (\mathbb{Z}/p\mathbb{Z})^* \mid x^2 - y^2 = a\}$ としたとき、 G には以下により群構造が入る。

$$\begin{aligned} e &:= \langle 1, 0, 1 \rangle, \\ \langle x, y, a \rangle * \langle z, w, b \rangle &:= \langle xz + yw, xw + yz, ab \rangle, \\ \langle x, y, a \rangle^{-1} &:= \langle xa^{-1}, -ya^{-1}, a^{-1} \rangle \end{aligned}$$

任意の $a \in (\mathbb{Z}/p\mathbb{Z})^*$ に対して、 $x^2 - y^2 = a$ となる $\langle x, y \rangle \in (\mathbb{Z}/p\mathbb{Z})^2$ が存在します。これは、 a が平方剰余の場合は自明で、 a が最大の平方非剰余の場合は $a + 1$ が平方剰余であることから $(\sqrt{a+1})^2 - 1^2 = a$ とすればよく、 a がそれ以外の平方非剰余の場合は、 G の群構造を利用すれば良いことからわかります。

また、このような $\langle x, y \rangle$ の個数は a に依存しません。適当に $\langle u, v, a \rangle \in G$ となるような u, v を選ぶことで、全単射 $\{\langle x, y \rangle \mid \langle x, y, 1 \rangle \in G\} \rightarrow \{\langle x, y \rangle \mid \langle x, y, a \rangle \in G\}$ が掛け算として構成できるからです。以上から、各 $a \in (\mathbb{Z}/p\mathbb{Z})^*$ に対して、 $x^2 - y^2 = a$ を満たす $\langle x, y \rangle \in (\mathbb{Z}/p\mathbb{Z})^2$ の個数が $|G|/(p-1)$ であることが分かりました。 $|G|$ を計算します。

$x^2 - y^2 = 0$ を満たす $\langle x, y \rangle$ は何個存在するのでしょうか? $x = 0$ ならば $y = 0$ で、 $x \neq 0$ ならば、 y は $y = \pm x$ の 2 個存在するので、合計 $(2p-1)$ 個です。よって、 $|G| = p^2 - (2p-1) = (p-1)^2$ が成り立ちます。

以上から、 $a \in (\mathbb{Z}/p\mathbb{Z})^*$ を固定したとき、 $x^2 - y^2 = a$ を満たす $\langle x, y \rangle$ はちょうど $(p-1)$ 個存在することが分かりました。求めたいものは、 a が平方剰余であることを仮定した場合の、 $x^2 - y^2 = a$ を満たす y が存在しないような x の個数です。 $x = \pm\sqrt{a}$ のときは $y = 0$ が条件を満たします。 x がそれ以外の場合は、条件を満たす y は存在しないか 2 個存在するかのいずれかです。よって、 y が存在しないような x の個数は、 $p-2 - (p-1-2)/2 = (p-1)/2$ 個であることが分かりました。

よって、問題の確率は $(p-1)/2p$ です。 □

また、このアルゴリズムの正当性については、以下のような観察により証明できます。

まず、 a は平方剰余なので、 $r^2 \equiv a \pmod{p}$ となるような r が存在します。このような r について、 $(x-c)^2 - a = (x-c-r)(x-c+r)$ です。先ほどみたように $\left(\frac{c^2-a}{p}\right) = -1$ で、命題 7.2 より $\left(\frac{c+r}{p}\right)\left(\frac{c-r}{p}\right) = -1$ です。そのため、 $\left(\frac{c+r}{p}\right)$ か $\left(\frac{c-r}{p}\right)$ のどちらかは 1 でどちらかは -1 となります。定理 8.1 より、 $x^{\frac{p-1}{2}} - 1$ は全ての平方剰余 a に対する $x-a$ の積です。よって、 $(x-c-r)(x-c+r)$ と $x^{\frac{p-1}{2}} - 1$ の最大公約数をとると、 $c \pm r$ のうち平方剰余の方だけが残ります。残った方が $-u^{-1}v$ なので、 $c \pm r = -u^{-1}v$ を解いて $r = \mp(c + u^{-1}v)$ です。

このアルゴリズムの計算量についてですが、 $r(x) = (x^{\frac{p-1}{2}} - 1) \bmod ((x-c)^2 - a)$ と置くと、 $r(x)$ が求めたい最大公約数であることは明らかです。これは二分累乘法を用いることで $O(\log p)$ で計算できます。よって、このアルゴリズムの計算量は、 $O(\log p)$ です。

実装を以下に載せます。

```

1 #include<utility>
2 #include<random>
3 using namespace std;
4 typedef long long lint;
5 typedef pair<lint,lint> pll; // <a,b> は  $a*x+b$  を表す。
6
7 lint powmod(lint a,lint e,lint p){
8     a%=p;
9     lint ans=1;
10    for(int i=63;i>=0;--i){
11        ans=ans*ans%p;
12        if(e&1LL<<i)ans=ans*a%p;
13    }
14    return ans;
15 }
16
17 // 多項式の掛け算  $\text{mod } x^2 - ca.first*x - ca.second$ 
18 pll mul_poly(pll a,pll b,pll ca,lint p){
19     lint s=a.first*b.first%p;
20     lint t=(a.first*b.second+a.second*b.first)%p;
21     lint u=a.second*b.second%p;
22     t=(t+ca.first*s)%p;
23     u=(u+ca.second*s)%p;
24     return pll(t,u);
25 }
26
27 // 2次体上の冪乗
28 pll pow_poly(pll a,lint e,pll ca,lint p){
29     pll ans(0,1);
30     for(int i=63;i>=0;--i){
31         ans=mul_poly(ans,ans,ca,p);
32         if(e&1LL<<i)ans=mul_poly(ans,a,ca,p);
33     }
34     return ans;
35 }
36
37 // p: 奇素数, a: mod p の平方剰余であることが要請される。
38 lint lehmer_mod_sqrt(lint p,lint a){
39     a%=p;
40     if(a==0)return 0;
41     lint c,t;
42     mt19937 mt;
43     do{
44         c=mt()%p;
45         t=(c*c+p-a)%p;
46     }while(powmod(t,(p-1)/2,p)!=p-1);
47     pll ans=pow_poly(pll(1,0),(p-1)/2,pll((2*c)%p,(p-t)%p),p);
48     ans.second=(ans.second+p-1)%p; // 最大公約数
49     return (c+powmod(ans.first,p-2,p)*ans.second+p)%p;
50 }

```

8.2 mod_sqrt その 3, Cipolla のアルゴリズム (Lv. 4)

Cipolla のアルゴリズムと呼ばれるものを紹介します。<http://pekempey.hatenablog.com/entry/2017/02/03/220150> でも紹介されています。

Algorithm 4 Cipolla のアルゴリズム

入力: p : 奇素数, a : $1 \leq a < p$ を満たす平方剰余

出力: $r^2 = a$ を満たす r

1. $c \in \mathbb{Z}/p\mathbb{Z}$ をランダムに選び、 $\left(\frac{c^2-a}{p}\right) = -1$ となるようにする。
 2. $\theta = \sqrt{c^2-a}$ として体拡大 $\text{GF}(p)(\theta)$ を考える。 $(c+\theta)^{\frac{p+1}{2}}$ は $\text{GF}(p)$ の元であり、これが答え (平方根の一つ) を与える。
-

命題 8.3. Algorithm 4 は正しい解を与える。

証明. Algorithm 3 と同様に、1. にかかる時間の期待値は、定数で抑えることができます。

2. について調べましょう。7.6 でみたように、 $(c+\theta)^p = c-\theta$ となることが示せます。よって $(c+\theta)^{p+1} = c^2 - \theta^2 = a$ です。よって $(c+\theta)^{\frac{p+1}{2}} \in \text{GF}(p)$ が示せば良いです。

より一般に $(x+y\theta)^2 = a$ ならば $y=0$ を示します。 $(x+y\theta)^2 = (x^2 + (c^2-a)y^2) + 2xy\theta$ であり、1 と θ が $\text{GF}(p)$ 上線形独立であることから、 $2xy=0$ であることがわかります。 p は奇素数なので、 $x=0$ あるいは $y=0$ が成り立ちます。ここで $x=0, y \neq 0$ だとしましょう。このとき $a = (y\theta)^2 = (c^2-a)y^2$ ですが、 c^2-a が平方非剰余となるように c を選んだので、矛盾します。□

注意 8.4. あるいは、これは単に、 $\text{GF}(p)(\theta)$ が体であるため、方程式 $x^2 - a = 0$ は $\text{GF}(p)(\theta)$ 上に根を最大 2 個しか持たず、 a が平方剰余であるためそれらは $\text{GF}(p)$ の元であるような 2 個だけである、ということもできます。

このアルゴリズムの計算量は、冪乗を何回か計算するだけなので、 $O(\log p)$ です。

実装を与えます。

ソースコード 12 cipolla.cpp

```
1 #include<utility>
2 #include<random>
3 using namespace std;
4 typedef long long lint;
5 typedef pair<lint,lint> pll; // 2次体の数をエンコードするためのもの。<x,y> は  $x+y\theta$  を表す。
6
7 lint powmod(lint a,lint e,lint p){
8     a%=p;
9     lint ans=1;
10    for(int i=63;i>=0;--i){
11        ans=ans*ans%p;
12        if(e&1LL<<i)ans=ans*a%p;
13    }
14    return ans;
15 }
16
```

```

17 // 2次体上の掛け算
18 pll mul_quad(pll a, pll b, lint theta, lint p){
19     lint x=a.first*b.first+(theta*a.second%p)*b.second;
20     x%=p;
21     lint y=a.first*b.second+a.second*b.first;
22     y%=p;
23     return pll(x,y);
24 }
25
26 // 2次体上の冪乗
27 pll pow_quad(pll a, lint e, lint theta, lint p){
28     pll ans(1,0);
29     for(int i=63;i>=0;--i){
30         ans=mul_quad(ans,ans,theta,p);
31         if(e&1LL<<i)ans=mul_quad(ans,a,theta,p);
32     }
33     return ans;
34 }
35
36 // p:奇素数, a:mod p の平方剰余であることが要請される。
37 lint cipolla(lint p, lint a){
38     a%=p;
39     if(a==0)return 0;
40     lint c,t;
41     mt19937 mt;
42     do{
43         c=mt()%p;
44         t=(c*c+p-a)%p;
45     }while(powmod(t,(p-1)/2,p)!=p-1);
46     pll ans=pow_quad(pll(c,1),(p+1)/2,t,p); // 証明中の  $x+y*theta$ 
47     return ans.first;
48 }

```

注意 8.5. このアルゴリズムに、 a として平方非剰余を与えると、命題 8.3 の証明中の議論で、 $y = 0$ ではなく $x = 0$ が成立します。よって、ソースコード 12 の実装では、常に 0 が返ることになります。

9 多項式を使うテク (Lv. 4)

9.1 高速フーリエ変換 (FFT) (Lv. 3)

数列 a_0, a_1, \dots, a_{n-1} と b_0, b_1, \dots, b_{n-1} に対して、 a と b の畳み込み (convolution) とは、数列 $c_i := \sum_{(j+k)\%n=i} a_j b_k$ ($0 \leq i \leq n-1$) のことです。多項式の積の係数も、畳み込みだと思えます。

高速フーリエ変換 (fast Fourier transform, FFT) という手法を使うことで、 d 項の数列の畳み込みの計算量が $O(d^2)$ から $O(d \log d)$ になります。よって、 d 次多項式の乗算も、同じ計算量でできます。詳しくは、https://atc001.contest.atcoder.jp/tasks/fft_c を参考にしてください。

`double` で計算を行うと誤差が出るので、次数 d , 係数の最大値 u として、 $du^2 \leq 10^{15}$ くらいでないと整数演算のためには使えません。理由は、畳み込みの結果の係数は最大 du^2 程度で、整数として扱うため `double` の精度の関係で $du^2 < 2^{53}$ でないといけなからです。 $d \leq 10^5, u \leq 10^5$ くらいでギリギリでしょう。

実装を載せます。

```

1  #include<cmath>
2  #include<complex>
3  #include<iostream>
4  #include<vector>
5  using namespace std;
6  typedef long long lint;
7  #define rep(i,n)for(int i=0;i<(int)(n);++i)
8
9  typedef complex<double> comp;
10
11 const double pi=acos(-1);
12
13 // n は 2 幂で、a.size()==n
14 void fft(int n,vector<comp> &a,double dir) {
15     // ビット反転は http://math314.hateblo.jp/entry/2015/05/07/014908 を参考にしている。
16     int i = 0;
17     for (int j = 1; j < n - 1; ++j) {
18         for (int k = n >> 1; k > (i ^ k); k >>= 1);
19         if (j < i) swap(a[i], a[j]);
20     }
21     vector<comp> zeta.pow(n);
22     rep(i,n){
23         double theta=pi/n*i*dir;
24         zeta.pow[i]=comp(cos(theta),sin(theta));// 毎回計算することで、誤差を回避する。
25     }
26     // ここも http://math314.hateblo.jp/entry/2015/05/07/014908 を参考にしている。
27     for(int m=1;m<n;m*=2){
28         for(int y=0;y<m;++y){
29             comp fac=zeta.pow[n/m*y];
30             for(int x=0;x<n;x+=2*m){
31                 int u=x+y;
32                 int v=x+y+m;
33                 comp s=a[u]+fac*a[v];
34                 comp t=a[u]-fac*a[v];
35                 a[u]=s;a[v]=t;
36             }
37         }
38     }
39 }
40
41 template<class T>
42 vector<comp> convolution(const vector<T> &a,const vector<T> &b){
43     int n=1;
44     while(n<(int)a.size()+(int)b.size())n*=2;
45     vector<comp>a_(n),b_(n);
46     rep(i,a.size())a_[i]=a[i];
47     rep(i,b.size())b_[i]=b[i];
48     fft(n,a_,1);fft(n,b_,1);
49     rep(i,n)a_[i]*=b_[i];
50     fft(n,a_,-1);
51     rep(i,n)a_[i]/=n;
52     return a_;
53 }
54
55
56 // 要素数n, 要素の最大値をc としたとき、 $n * c^2 \leq 10^{15}$  でなければ精度が保証されない。

```

```

57 vector<lint> integral_convolution(const vector<lint> &a,const vector<lint> &b){
58     vector<comp>ans=convolution(a,b);
59     vector<lint>ret(ans.size());
60     rep(i,ans.size())ret[i]=floor(ans[i].real()+0.5);
61     return ret;
62 }
63
64 int main(){
65     vector<lint> a(3);
66     a[0]=1,a[1]=2,a[2]=3;
67     vector<lint> b(5);
68     b[0]=1,b[1]=10,b[2]=15,b[3]=11,b[4]=1;
69     rep(i,5)b[i]=-b[i];
70     vector<lint> ab=integral_convolution(a,b);
71     // -1 -12 -38 -71 -68 -35 -3 0
72     rep(i,ab.size())cout<<" ";<<ab[i];
73     cout<<endl;
74 }

```

COLUMN

NTT (number theoretic transformation) (FMT (fast modulo transformation) ともいう) という手法があります。これは、double の代わりに Proth prime $p = k \times 2^n + 1$ を mod とした環の上で FFT をするという手法です。整数演算なので誤差が出ないのが嬉しいですね。その上、double の上で行う FFT と比べてもパフォーマンスに大きな違いはありません。詳しくは <http://math314.hateblo.jp/entry/2015/05/07/014908> をご覧ください。サブセクション 9.4 でも少し触れます。

9.2 フェルマーの小定理 (Lv. 3)

定理 9.1. 素数 p に対して、以下の等式が成り立つ:

$$(x+1) \times (x+2) \times \cdots \times (x+(p-1)) \equiv x^{p-1} - 1 \pmod{p}$$

証明. まず、 $(x+k)$ 同士は互いに素です。フェルマーの小定理から、 $1 \leq k \leq p-1$ なる各 k について、 $x+k | x^{p-1}-1$ が成り立ちます (整除は $\text{GF}(p)[x]$ の上のもので)。以上より、 $(x+1) \times (x+2) \times \cdots \times (x+(p-1)) | x^{p-1}-1$ です。この整除関係の左辺も右辺も $(p-1)$ 次なので、両者は定数倍の違いしかありません。 x^{p-1} の係数を比べることで、その定数倍は 1 倍である、つまり両者が等しいことがわかります。□

注意 9.2. p が素数でない場合にこれを拡張しようとしても、失敗します。例えば $p = 8$ の時、 $(x+1)(x+3)(x+5)(x+7) \equiv x^4 + 6x^2 + 1 \not\equiv x^{p(8)} - 1 \pmod{8}$ です。これは、 $x+k$ が $(\mathbb{Z}/p\mathbb{Z})[x]$ の既約元であっても素元ではないことに由来します。^{*2}

この事実を応用して解ける問題を紹介します。以下の問題を考えましょう。

^{*2} 可換環 R があるとき、 $a \in R$ が既約元 (irreducible element) であるとは、 $b \in R, c \in R, a = bc$ となる場合 b または c のどちらかが可逆元 (invertible element) (逆数が R の元である) となることです。また、 a が素元 (prime element) であるとは、 $a | xy$ となる場合、 $a | x$ または $a | y$ のどちらかが成立することです。可換環 R が体なら $R[x]$ は既約元が全部素元になりますが、 R が体でない場合それは保証されません。 p が素数でない場合、 $\mathbb{Z}/p\mathbb{Z}$ は体ではありません。

問題

正の整数 n と素数 p が与えられる。 $[n] := \{1, \dots, n\}$ として、整数 k に対して $f(n, k)$ を以下で定める:

$$f(n, k) := \sum_{S \subseteq [n], |S|=k} \prod_{x \in S} x$$

このとき、 $p \nmid f(n, k)$ となるような $0 \leq k \leq n$ の個数を、 $\text{mod } 10^9 + 7$ で求めよ。(テストケースは T ケース与えられる。)

制約: $1 \leq T \leq 4, 2 \leq p \leq 100000, p$ は素数

- 部分点 (10/100 点): $n \leq 5000$
- 部分点 (50/100 点): $n \leq 100000$
- 満点 (100 点): $n < 10^{501}$

(出典: CodeChef February Challenge 2018 (FEB18) » Lucas Theorem (LUCASTH))

簡単な式変形、DP、あるいは実験などで、

$$f(n, k) = ((t+1) \times (2t+1) \times \dots \times (nt+1) \text{ の } t^k \text{ の係数})$$

であることがわかります。 $P(n) := (t+1) \times (2t+1) \times \dots \times (nt+1)$ と置きましょう。 $P(n)$ の係数を愚直に計算することで、 $O(n^2)$ 解法が得られます (10/100 点)。

これを高速化することを考えましょう。多項式の乗算を高速化したいので、FFT を使うことが思いつきます。分割統治で計算すれば、 $O(n(\log n)^2)$ 解法が得られます (50/100 点)。

満点解法について考えましょう。先ほど紹介したフェルマーの小定理を少し修正することで、以下が分かります:

$$P(p-1) = (t+1) \times (2t+1) \times \dots \times ((p-1)t+1) \equiv -t^{p-1} + 1 \pmod{p}$$

$(kt+1) \pmod{p}$ は周期 p なので、 $P(pk) \equiv P(p)^k \equiv (-t^{p-1} + 1)^k \pmod{p}$ が分かります。

よって、 $n = qp + r$ ($0 \leq r < p$) としたとき、 $P(n) \equiv P(qp)P(r) \equiv (-t^{p-1} + 1)^q P(r) \pmod{p}$ が計算できればよいです。

$(-t^{p-1} + 1)^q$ について考えましょう。今は p の倍数かどうかに関心があるので、 $(t^{p-1} + 1)^q$ を考えても同じです。ここで、次のような事実が実験によって分かります:

非負整数 a の p 進表記を $a = (d_{e-1}d_{e-2} \dots d_1d_0)_p$ とすると、 $(x+1)^a \pmod{p}$ の 0 でない係数は、ちょうど $(d_{e-1}+1) \times (d_{e-2}+1) \times \dots \times (d_0+1)$ 個ある。

これに $x = t^{p-1}$ を代入すると、 $q = (d_{e-1}d_{e-2} \dots d_1d_0)_p$ として、 $(-t^{p-1} + 1)^q \pmod{p}$ の 0 でない係数は $(d_{e-1}+1) \times (d_{e-2}+1) \times \dots \times (d_0+1)$ 個あって、しかもそれぞれの次数は $p-1$ 次以上離れていることが分かります。 p 進表記を計算する方法は色々ありますが、 n の桁数が小さいので、 $O((\log n)^2)$ 時間かけて愚直に多倍長整数演算をすれば良いでしょう。

次に $P(r)$ ですが、 $P(r)$ は r 次なので、 $r < p-1$ ならば $(-t^{p-1} + 1)^q$ と干渉しないことが分かります。よって答えは $(d_{e-1}+1) \times (d_{e-2}+1) \times \dots \times (d_0+1) \times (P(r) \pmod{p})$ の 0 でない係数) であることが分かります。 $r = p-1$ の時は、 $P(r) \equiv P(p) \pmod{p}$ より、 $n = (q+1)p$ の時と同じ答えになることが分かります。

以上から $O(p(\log p)^2 + (\log n)^2)$ 時間の解法が得られました。以下にソースコードを載せます。https://www.codechef.com/viewsolution/17683322 です。実装には double 型の演算を用いた FFT を用いまし

た。FFT の部分はソースコード 13 と同じです。25 行目において、1 の n 乗根 $\zeta = \cos(2\pi/n) + \sqrt{-1} \sin(2\pi/n)$ の i 乗 $\zeta^i = \cos(2\pi i/n) + \sqrt{-1} \sin(2\pi i/n)$ を愚直に計算していることに注意しましょう。(気を利かせて、毎回 ζ を掛けることで ζ^i を計算する、などすると誤差死します。ていうかしました)

ソースコード 14 FEB18-LUCASTH.cpp

```

1  #include<cmath>
2  #include<complex>
3  #include<iostream>
4  #include<string>
5  #include<vector>
6  using namespace std;
7  typedef long long lint;
8  #define rep(i,n)for(int i=0;i<(int)(n);++i)
9
10 typedef complex<double> comp;
11
12 const double pi=acos(-1);
13
14 // n は 2 幂で、a.size()==n
15 void fft(int n,vector<comp> &a,double dir) {
16     // ビット反転は http://math314.hateblo.jp/entry/2015/05/07/014908 を参考にしている。
17     int i = 0;
18     for (int j = 1; j < n - 1; ++j) {
19         for (int k = n >> 1; k > (i ^ k); k >>= 1);
20         if (j < i) swap(a[i], a[j]);
21     }
22     vector<comp> zeta_pow(n);
23     rep(i,n){
24         double theta=pi/n*i*dir;
25         zeta_pow[i]=comp(cos(theta),sin(theta));// 毎回計算することで、誤差を回避する。
26     }
27     // ここも http://math314.hateblo.jp/entry/2015/05/07/014908 を参考にしている。
28     for(int m=1;m<n;m*=2){
29         for(int y=0;y<m;++y){
30             comp fac=zeta_pow[n/m*y];
31             for(int x=0;x<n;x+=2*m){
32                 int u=x+y;
33                 int v=x+y+m;
34                 comp s=a[u]+fac*a[v];
35                 comp t=a[u]-fac*a[v];
36                 a[u]=s;a[v]=t;
37             }
38         }
39     }
40 }
41
42 template<class T>
43 vector<comp> convolution(const vector<T> &a,const vector<T> &b){
44     int n=1;
45     while(n<(int)a.size()+(int)b.size())n*=2;
46     vector<comp> a_(n),b_(n);
47     rep(i,a.size())a_[i]=a[i];
48     rep(i,b.size())b_[i]=b[i];
49     fft(n,a_,1);fft(n,b_,1);
50     rep(i,n)a_[i]*=b_[i];
51     fft(n,a_,-1);

```

```

52 rep(i,n)a_[i]/=n;
53 return a_;
54 }
55
56
57 // 要素数 $n$ , 要素の最大値を $c$ としたとき、 $n * c^2 \leq 10^{15}$  でなければ精度が保証されない。
58 vector<lint> integral_convolution(const vector<lint> &a,const vector<lint> &b){
59     vector<comp>ans=convolution(a,b);
60     vector<lint>ret(ans.size());
61     rep(i,ans.size())ret[i]=floor(ans[i].real()+0.5);
62     return ret;
63 }
64
65 pair<string,lint> divide(const string &n,lint r){
66     string res;
67     lint rem=0;
68     bool cont_zero=1;
69     rep(i,n.size()){
70         rem=10*rem+(n[i]-'0');
71         lint q=rem/r;
72         rem%=r;
73         if(cont_zero&&q==0)continue;
74         if(q!=0)cont_zero=0;
75         res+='0'+q;
76     }
77     return make_pair(res,rem);
78 }
79
80
81 vector<lint> parse(const string &n,lint r){
82     vector<lint> dig;
83     string cur(n);
84     while(1){
85         pair<string,lint>sub=divide(cur,r);
86         dig.push_back(sub.second);
87         cur=sub.first;
88         if(cur=="")break;
89     }
90     return dig;
91 }
92
93 vector<lint> g(lint p,lint l,lint r){
94     if(l>r){
95         return vector<lint>(1,1);
96     }
97     if(l==r){
98         vector<lint> ret(2);
99         ret[0]=1;
100         ret[1]=l%p;
101         return ret;
102     }
103     lint mid=(l+r+1)/2;
104     vector<lint>fst=g(p,l,mid-1);
105     vector<lint>snd=g(p,mid,r);
106     vector<lint>ans=integral_convolution(fst,snd);
107     rep(i,ans.size())ans[i]%=p;
108     return ans;

```

```

109 }
110
111
112 lint f(string n,lint p){
113     const lint mod=1e9+7;
114     vector<lint> dig=parse(n,p);
115     if(dig[0]==p-1){
116         //propagate
117         dig[0]=0;
118         int car=1;
119         for(int pos=1;pos<(int)dig.size();++pos){
120             dig[pos]+=car;
121             car=dig[pos]/p;
122             dig[pos]%=p;
123         }
124         if(car>0)dig.push_back(car);
125     }
126     vector<lint> ans=g(p,1,dig[0]);
127     lint ret=0;
128     rep(i,dig[0]+1)
129         if(ans[i]!=0)ret++;
130     rep(i,dig.size()-1)
131         ret=ret*(dig[i+1]+1)%mod;
132     return ret;
133 }
134
135 int main(){
136     int t;
137     cin>>t;
138     while(t--){
139         string n;
140         lint p;
141         cin>>n>>p;
142         cout<<f(n,p)<<endl;
143     }
144 }

```

また、<http://math314.hateblo.jp/entry/2015/05/07/014908> を参考にした NTT による実装も行いました。ソースコードは割愛しますが、<https://www.codechef.com/viewsolution/17544576> で公開されています。double 型 FFT は 2.16sec で、NTT は 2.72sec で終了したので、両者に大きなパフォーマンスの違いはありません。(NTT は 3 並列でやったので、2 並列にしたら NTT の方が早くなる可能性もあります。)

9.3 巡回群構造を用いた特殊な畳み込み (Lv. 4)

問題

素数 p と、長さ n の整数列 A と、長さ m の整数列 B が与えられる。このとき、 $(A_i B_j) \bmod p < l$ となるような組 (i, j) の個数を求めよ。

制約: $2 \leq p \leq 250000, 1 \leq l \leq p, 2 \leq n, m \leq 10000000 = 10^7, A$ と B は、種が与えられた乱数生成器によって生成される。

(出典: Facebook Hacker Cup 2011 Round2 - Scott's New Trick <http://techtipshoge.blogspot.jp/2012/04/facebook-hacker-cup-2011-round2-scott.html>)

p の大きさが比較的小さめなので、 $A_i \bmod p$ と $B_j \bmod p$ の出現回数を記録しておいて、あとでまとめて加算するという手法が思いつきます。つまり、以下のような操作を行うということです:

```
ac=[],bc=[];
for i in 0..n{
    ac[a[i]%p]++;
}
for i in 0..m{
    bc[b[i]%p]++;
}
ans=[];
for i in 0..p{
    for j in 0..p{
        ans[(i*j)%p]+=ac[i]*bc[j];
    }
}
```

この手法だと計算量は $O(p^2)$ です。

これではうまく行かないので、高速化をしたいところです。ここで、上の擬似コードに畳み込みのようなものが出現していることに着目します。畳み込みで加算になっているべきところが $(i*j)\%p$ になっています。そのため、 $(\mathbb{Z}/p\mathbb{Z})^*$ の群構造に着目して、この問題を解きたいと思います。

セクション 6 でみたように、 $(\mathbb{Z}/p\mathbb{Z})^*$ は巡回群であることが知られています。つまり、ある要素 (生成元) $g \in (\mathbb{Z}/p\mathbb{Z})^*$ が存在して、 $(\mathbb{Z}/p\mathbb{Z})^*$ の要素 $x \in (\mathbb{Z}/p\mathbb{Z})^*$ はすべて、 $x = g^i$ の形で表せるということです。このような g のことを、 $\bmod p$ における原始根と呼ぶのでした。ここで、 $x = g^i, y = g^j$ とすると、 $xy \bmod p = g^i g^j \bmod p = g^{i+j}$ です。つまり、 ac や bc の添字を写像 $i \mapsto g^i$ で写せば、普通の畳み込みに帰着できます。

原始根の計算方法について説明します。ある要素 $a \in (\mathbb{Z}/p\mathbb{Z})^*$ が原始根であるかどうかは、brute force により $O(p)$ 時間で検査できます。それでは、原始根はどのくらいの割合で存在するのでしょうか? ある原始根 g に対して $a = g^i$ となっている場合、 $\gcd(i, p-1) = 1$ であることと a が原始根であることは同値です。 $p-1 = q_1^{e_1} \cdots q_k^{e_k}$ を $p-1$ の素因数分解とすると、上の条件を満たす i を選ぶ確率は $(q_1-1) \cdots (q_k-1)/q_1 \cdots q_k$

です。 $p \leq 250000$ より、 k は最大でも 5 程度であり、 a が原始根である確率は十分大きいと言えるでしょう。以上より、ランダムに a を選び、原始根かどうか調べることによって、原始根を得ることができます。(もっと高速な方法もあります。考えてみましょう。答えはソースコード 15 の実装を読むと分かります。)

実装するときは、0 番目の要素の扱いに注意しましょう。0 は g^i の形で表せません。(この問題は公式のページが消えていて、実装してもジャッジできませんが…。)

9.4 1 の 2^k 乗根 mod p を用いた畳み込み

素数 p が特殊な形 $p = u2^e + 1$ (u が小さく、 e が大きい) の場合、 $k \leq e$ なる k に対して、 2^k 要素の FFT が(つまり結果として畳み込みが)できます。このような素数には、以下のようなものがあります。

p	$u \times 2^e + 1$	16 進表記	最小の原始根
998244353	$119 \times 2^{23} + 1$	0x3b800001	3
163577857	$39 \times 2^{22} + 1$	0x9c00001	23
167772161	$5 \times 2^{25} + 1$	0xa000001	3
469762049	$7 \times 2^{26} + 1$	0x1c000001	3

mod p の原始根を g とすると、 $g^u \bmod p$ が 2^e 乗根を与えます。mod の値が $10^9 + 7$ 以外の場合は、このような特殊な素数ではないかと疑ってみて、16 進表示を求めるなどすると良いでしょう。

有名な素数についてはもう原始根が計算されているので、それも表に書きました。表にない素数が出題された時のために、一応原始根を計算するスクリプトを載せておきます。Python 3 で書かれています。

ソースコード 15 generator.py

```

1 p = int(input())
2
3 fac = []
4 pp = 2
5 v = p - 1
6 while v >= pp * pp:
7     e = 0
8     while v % pp == 0:
9         e += 1
10        v //= pp
11    if e > 0:
12        fac.append(pp)
13    pp += 1
14 if v > 1:
15    fac.append(v)
16
17 g = 2
18 while g < p:
19     if pow(g, p - 1, p) != 1:
20         print(str(p) + 'is not a prime')
21         exit()
22     ok = True
23     for pp in fac:
24         if pow(g, (p - 1) // pp, p) == 1:
25             ok = False
26             break

```

```

27     if ok:
28         print('The smallest generator is ' + str(g))
29         exit()
30     g += 1

```

mod p における 1 の 2^k 乗根を利用した FFT (NTT) については、<http://math314.hateblo.jp/entry/2015/05/07/014908> を参考にしてください。

10 ペル方程式 (Lv. 4)

問題

一辺 a メートルの正方形がある。この正方形から、一辺 b メートルの正方形を n 個切り出すことを考える。ただし、切り出されなかった部分の面積は、元の正方形の面積の 50% 以上でなければならない。つまり、 $a^2 - nb^2 \geq a^2/2$ が必要である。

ここで切り出されなかった部分のうち、 a^2 平方メートルの 50% を越える部分の面積 (つまり $(a^2/2 - nb^2)$ 平方メートル) を最小化したい。 n が与えられるので、最小値を与える最小の正の整数 a, b を与えよ。

制約: $1 \leq n \leq 10000$

(出典: Aizu Online Judge 2116: Subdividing a Land (ACM-ICPC Japan Alumni Group Practice Contest, for World Finals, Tokyo, Japan, 2008-02-23), <http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=2116>)

まず、 $2n$ が平方数の場合、 $a = \sqrt{2n}, b = 1$ が最小値 0 を与えることが自明です。そうでない場合を考えます。 $a^2 - 2nb^2 \geq 0$ より $b \leq a/\sqrt{2n}$ です。よって、1 辺 b の正方形を、縦横それぞれ $\lfloor \sqrt{2n} \rfloor$ 個並べることができます。簡単な計算により、任意の正の整数 n に対して $n \leq \lfloor \sqrt{2n} \rfloor^2$ であることがわかるので、1 辺 a の正方形の中に 1 辺 b の正方形を確実に n 個詰められることがわかります。つまり、結局 $a^2 - 2nb^2 \geq 0$ の条件のもとで、 $a^2 - 2nb^2$ の最小値を与える最小の a, b を計算すれば良いことが分かります。

ここで、以下の事実が知られています。

定理 10.1. d が平方数でない正の整数のとき、方程式 $x^2 - dy^2 = 1$ は、正の整数解 $\langle x, y \rangle$ を必ず持つ。

このような方程式をペル方程式 (Pell's equation) と呼び、このような (x, y) のうち、 x が最小のものを基本解 (fundamental solution) と呼びます。この問題では、基本解が計算できれば良いです。

ペル方程式の基本解を計算するアルゴリズムを説明します。連分数を使う、以下のアルゴリズムが広く知られています。

具体例として、 $d = 14$ の場合にこのアルゴリズムがどう動くかを見てみましょう。数列 α_i と q_i は以下のようになります:

$$\{\alpha_i\} : \sqrt{14}, \frac{3+\sqrt{14}}{5}, \frac{2+\sqrt{14}}{2}, \frac{2+\sqrt{14}}{5}, 3+\sqrt{14}, \frac{3+\sqrt{14}}{5}, \frac{2+\sqrt{14}}{2}, \dots$$

$$\{q_i\} : 3, 1, 2, 1, 6, 1, 2, \dots$$

循環節は q_1 から q_4 までの長さ 4 の部分列です。 $m = 4$ で、

$$b = [3; 1, 2, 1] = 15/4$$

Algorithm 5 ペル方程式

入力: d : 正の整数、平方数でない

出力: $\langle x, y \rangle$, ただし x, y は正の整数で $x^2 - dy^2 = 1$ を満たす、 x, y はその中で最小

1. $\alpha_0 := \sqrt{d}$ とする。
2. $q_i := \lfloor \alpha_i \rfloor, \alpha_{i+1} := 1/(\alpha_i - q_i)$ として無限数列を作る。この無限数列は、必ず循環する。
3. 上で作った数列は、循環節が q_1 から開始する。つまり、 $(q_i): q_0, q_1, \dots, q_{m-1}, q_m, q_1, \dots$ となる。このような最小の m をとり、

$$b := [q_0; q_1, \dots, q_{m-1}] = q_0 + \frac{1}{q_1 + \frac{1}{\dots + \frac{1}{q_{m-1}}}}$$

とする。

4. $b = x/y$ (x と y は互いに素な正の整数) としたとき、 x, y は $x^2 - dy^2 = \pm 1$ を満たす。 $x^2 - dy^2 = 1$ の場合、 $\langle x, y \rangle$ を出力する。 $x^2 - dy^2 = -1$ の場合、 $\langle x^2 + dy^2, 2xy \rangle$ を出力する。
-

が成立します。 $15^2 - 14 \times 4^2 = 1$ であるため、出力は $\langle 15, 4 \rangle$ となります。

実装するときの注意として、 α_i の表現方法があります。実は、 α_i は $\alpha_i = (x + \sqrt{d})/z$ (x, z は整数) という形で表せることが知られているので、この x, z を保持すれば良いです。

ソースコードを載せます。このソースコードは <http://judge.u-aizu.ac.jp/onlinejudge/review.jsp?rid=2723600> で公開されています。

ソースコード 16 aoj-2116.cpp

```
1 #include<iostream>
2 #include<vector>
3 #include<set>
4 #include<cassert>
5 #include<cmath>
6 using namespace std;
7 typedef long long lint;
8 typedef pair<lint,lint> pll;
9
10 // 連分数を使って、基本単数 (Pell 方程式の基本解) を求める。d は平方数でないことが要請される。
11 pll fundamental_unit(lint d){
12     vector<int> ans;
13     lint x=0,z=1; //(x+sqrt(d))/z
14     lint sqrt_d=floor(sqrt(d));
15     set<pll> seen;
16     // invariants: x>=0, z|x*x-d
17     while(1){
18         if(seen.count(pll(x,z)))break;
19         seen.insert(pll(x,z));
20         lint q=(x+sqrt_d)/z;
21         ans.push_back(q);
22         x=q*z-x;
23         lint norm=x*x-d;
24         z=-norm/z;
25     }
26     // recover
27     lint num=0,den=1;
28     for(int i=(int)ans.size()-2;i>=0;--i){
29         lint z=num+ans[i]*den;
```



```

30     num=den;den=z;
31 }
32 if(den*den-d*num*num==1){
33     lint x=den*den+d*num*num;
34     lint y=2*den*num;
35     den=x;num=y;
36 }
37 assert(den*den-d*num*num==1);
38 return pll(den,num);
39 }
40
41 pll solve(lint n){
42     for(int i=0;i<=200;++i)
43         if(i*i==2*n)
44             return pll(i,1);
45     return fundamental_unit(2*n);
46 }
47
48 int main(){
49     for(int t=1;;++t){
50         lint n;
51         cin>>n;
52         if(n==0)break;
53         pll ans=solve(n);
54         cout<<"Case_"<<t<<" :_"<<ans.first<<"_"<<ans.second<<endl;
55     }
56 }

```

COLUMN

定理 10.1 は、ディリクレの単数定理 (Dirichlet's unit theorem) [2, Theorem 5.13] の特殊な場合で、 $K = \mathbb{Q}(\sqrt{d})$ (実 2 次体) のオーダー $\mathbb{Z}[\sqrt{d}]$ の単数たちのなす乗法群 U のランクが 1 である (つまり、 $U \cong (\text{有限群}) \times \mathbb{Z}$ と表すことができる) ことを言っています。 \mathbb{Z} 成分の生成元のことを基本単数 (fundamental unit) と呼びますが、上のアルゴリズムで求めた $\langle x, y \rangle$ が基本単数 $x + y\sqrt{d}$ を与えます。興味のある人は調べてみましょう。

11 単項イデアル整域 (Lv. 5)

定義 11.1. 可換環 R が整域 (integral domain) であるとは、 $x, y \in R$ が $xy = 0$ を満たすならば、 x か y の少なくとも一方は 0 であることである。

定義 11.2. 可換環 R のイデアル (ideal) とは、 R の部分集合 I であって、以下の 2 条件を満たすものである：

- $x \in I, y \in I \rightarrow x + y \in I$
- $x \in R, y \in I \rightarrow xy \in I$

定義 11.3. 可換環 R とその要素 $x \in R$ に対して、 x によって生成される単項イデアル (a principal ideal generated by x) とは、 $xR := \{xy \mid y \in R\}$ のことである。これを (x) と表記する。

定義 11.4. 可換環 R とその要素 $x_1, \dots, x_n \in R$ に対して、 x_1, \dots, x_n によって生成されるイデアル (an ideal generated by x_1, \dots, x_n) とは、 $\{\sum_{i=1}^n \alpha_i x_i \mid \alpha_i \in R\}$ のことである。これを (x_1, \dots, x_n) と表記する。

定義 11.5. 単項イデアル整域 (*principal ideal domain*, PID) (独: *Hauptidealbereich*, ハウプトイデアルベライヒ) とは、全てのイデアルが単項生成であるような整域である。

例 11.6. PID の例として有名なものは、 \mathbb{Z} , $\mathbb{Z}[i]$ (i は虚数単位)、 $K[x]$ (係数が K の元であるような多項式全体のなす集合、 K は体) などです。逆に PID でない例として有名なものには、 $\mathbb{Z}[\sqrt{-5}]$ などがあります。

以下の問題を考えてみましょう。

問題

(P, Q) -サンタがいる。 (P, Q) -サンタは最初原点 $(0, 0)$ におり、 (x, y) からは $(x \pm P, y \pm Q)$ または $(x \pm Q, y \pm P)$ の 8 種類の点に移動できる。 N 人の子供の座標 (X_i, Y_i) が与えられるので、 (P, Q) -サンタが到達できる子供の数を求めよ。

(出典: yukicoder No.321 (P, Q) -サンタと街の子供たち)

この問題に限り、添字との混同を防ぐため、虚数単位を $\sqrt{-1}$ と表記します。座標 (x, y) に到達可能なとき、 $x + y\sqrt{-1}$ がどのような条件を満たすべきかを考えてみましょう。問題の移動は、 $x + y\sqrt{-1}$ から $x + y\sqrt{-1} \pm (P + Q\sqrt{-1})$, $x + y\sqrt{-1} \pm (P - Q\sqrt{-1})$, $x + y\sqrt{-1} \pm (Q + P\sqrt{-1})$, $x + y\sqrt{-1} \pm (Q - P\sqrt{-1})$ の 8 種類の点に移動するものと考えることができます。 $Q - P\sqrt{-1} = -\sqrt{-1}(P + Q\sqrt{-1})$, $Q + P\sqrt{-1} = \sqrt{-1}(P - Q\sqrt{-1})$ に注意すると、結局移動できるのは

$$\alpha(P + Q\sqrt{-1}) + \beta(P - Q\sqrt{-1})$$

(ただし、 $\alpha, \beta \in \mathbb{Z}[\sqrt{-1}]$) で表される点ということが分かります。集合

$$\{\alpha(P + Q\sqrt{-1}) + \beta(P - Q\sqrt{-1}) \mid \alpha, \beta \in \mathbb{Z}[\sqrt{-1}]\}$$

は、他にもないイデアル $(P + Q\sqrt{-1}, P - Q\sqrt{-1})$ であり、 $\mathbb{Z}[\sqrt{-1}]$ が PID であるという性質から、 $P + Q\sqrt{-1}$ と $P - Q\sqrt{-1}$ の最大公約数を γ と置くと、このイデアルは γ によって生成される単項イデアル (γ) です。よって、 $\gamma \mid X_i + Y_i\sqrt{-1}$ かどうかの判定を行うことで、この問題が解けました。なお、実装時には $\gamma = 0$ となるコーナーケース ($P = Q = 0$) に気をつけましょう。

ソースコードを以下に載せます。<https://yukicoder.me/submissions/243652> です。

ソースコード 17 yukicoder-321.cpp

```
1 #include<iostream>
2 using namespace std;
3 typedef long long lint;
4 typedef pair<lint,lint> pll; // ガウス整数の表現用。<a,b>はa+biを表す。
5
6 // a/b に一番近い整数を返す。
7 lint quo_nearest(lint a,lint b){
8     lint q=a/b;
9     lint r=a-q*b;
10    if(abs(r)>abs(r-b))return q+1;
11    if(abs(r)>abs(r+b))return q-1;
12    return q;
13 }
14
15 // ガウス整数の剰余を計算する。一番絶対値が小さいものを返す。
16 pll rem_gaussian(pll a,pll b){
```

```

17 lint bnorm=b.first*b.first+b.second*b.second;
18 lint x=a.first*b.first+a.second*b.second;
19 lint y=-a.first*b.second+a.second*b.first;
20 // a/b に一番近いガウス整数を q として、a-q*b を計算する。a/b=(x+yi)/bnorm なので、x/bnorm,y/
    bnorm に一番近い整数がわかればよい。
21 lint qx=quo_nearest(x,bnorm);
22 lint qy=quo_nearest(y,bnorm);
23 return pll(a.first-qx*b.first+qy*b.second,a.second-qx*b.second-qy*b.first);
24 }
25
26 // ガウス整数の最大公約数を計算する。
27 pll gcd_gaussian(pll a,pll b){
28     while(b!=pll(0,0)){
29         pll c=rem_gaussian(a,b);
30         a=b;b=c;
31     }
32     return a;
33 }
34
35 int main(){
36     lint p,q;
37     int n;
38     cin>>p>>q>>n;
39     pll g=gcd_gaussian(pll(p,q),pll(p,-q));
40     int ans=0;
41     while(n--){
42         lint x,y;
43         cin>>x>>y;
44         ans+=g==pll(0,0)?(x==0&&y==0):rem_gaussian(pll(x,y),g)==pll(0,0);
45     }
46     cout<<ans<<endl;
47 }

```

また、以下の問題を考えてみましょう。

問題

N 個の非負整数 A_i が黒板に書かれている。以下の操作を何度でも行える:

- 黒板にある数の一つを選び、それを x とする。 $2x$ を新しく書き込む。
- 黒板にある数を二つ選び、それらを x, y とする (同じ数でも良い)。 $x \text{ xor } y$ を新しく書き込む。

最終的に書き込める数のうち、 X 以下のものは何種類あるか? 998244353 で割った余りを求めよ。

制約: $1 \leq N \leq 6, 1 \leq X < 2^{4000}, 1 \leq A_i < 2^{4000}, X$ と A_i は 2 進数で与えられ、先頭の桁は 1。

(出典: ARC084 F - XorShift)

まず、演算が 2 倍と xor なので、整数を以下のようにして、 $\text{GF}(2)[x]$ の元 (つまり、 $\text{GF}(2)$ 係数の多項式) として表すという発想が自然です:

$$t = b_u 2^u + b_{u-1} 2^{u-1} + \cdots + b_1 2 + b_0 \mapsto b_u x^u + b_{u-1} x^{u-1} + \cdots b_1 x + b_0$$

多項式の上の演算として考えると、2 倍する操作は x を掛ける操作、xor をとる操作は多項式の足し算です。
($\text{GF}(2)$ の上の足し算が xor であることに注意してください。) $\text{GF}(2)[x]$ は PID なので、この問題は、 $\text{GF}(2)[x]$

の上の最大公約数が計算できれば解けます。つまり、整数 t が黒板に書き込めることと、以下が同値です:

$$t \in (A_1, \dots, A_N) = (\gcd(A_1, \dots, A_N))$$

この場合、最大公約数の計算一回には $O(|A_i|^2)$ 時間かかるので、全体の計算量は $O(N|A_i|^2)$ となり、十分間に合います。ビット並列のテクニックを使うことで、 $O(N|A_i|^2/64)$ にしても良いでしょう。

ソースコードは以下ようになります。<https://arc084.contest.atcoder.jp/submissions/2137652> です。

ソースコード 18 ARC084F.cpp

```
1 #include<algorithm>
2 #include<bitset>
3 #include<cassert>
4 #include<iostream>
5 #include<vector>
6 using namespace std;
7 typedef long long lint;
8 #define rep(i,n)for(int i=0;i<(int)(n);++i)
9
10 const int N=4000;
11 typedef bitset<N> bs;
12 const lint MOD=998244353;
13
14 bs read(){
15     string s;
16     cin>>s;
17     int l=s.length();
18     bs ret;
19     rep(i,l)ret[i]=s[l-i-1]=='1';
20     return ret;
21 }
22
23 // a%b
24 bs rem(bs a,bs b){
25     int hi=-1;
26     rep(i,N)
27         if(b[i])hi=i;
28     assert(hi>=0);
29     for(int i=N-hi-1;i>=0;--i)
30         if(a[i+hi])a^=b<<i;
31     return a;
32 }
33
34 bs gcd(bs a,bs b){
35     while(b.count()!=0){
36         a=rem(a,b);
37         swap(a,b);
38     }
39     return a;
40 }
41
42 int main(){
43     int n;
44     cin>>n;
45     bs x=read();
46     vector<bs> s(n);
```

```

47 rep(i,n)s[i]=read();
48 bs g=s[n-1];
49 rep(i,n-1)g=gcd(s[i],g);
50 lint ans=0;
51 lint cur=1;
52 int hi=-1;
53 rep(i,N)
54     if(g[i])hi=i;
55 rep(i,N-hi){
56     if(x[i+hi])ans=(ans+cur)%MOD;
57     cur=cur*2%MOD;
58 }
59 bs y=x^rem(x,g);
60 bool lt=false; //x<y?
61 for(int i=N-1;i>=0;--i){
62     if(x[i]!=y[i]){
63         lt=x[i]<y[i];
64         break;
65     }
66 }
67 if(!lt)ans=(ans+1)%MOD; // y<=x; counts y
68 cout<<ans<<endl;
69 }

```

索引

Sylow 部分群, 14

アーベル群 (可換群), 15

イデアル, 33

可逆元, 24

基本解, 31
基本単数, 33
既約元, 24
逆元, 3
共役, 17

群, 14

原始根, 15

巡回群, 15
乗法群, 15

整域, 33
生成元, 15

素元, 24

対称群, 14
畳み込み, 22
単項イデアル整域, 34

添加, 16

2 次体, 16
二分累乗法, 5

部分群, 15
不変量, 2
フロベニウス写像, 17

平方剰余, 9
平方非剰余, 9
ベル方程式, 31

モノイド, 5

ルジャンドル記号, 15

謝辞

全ての助けの中で、けんちゃん (@drken1215) によるものが最大です。彼の協力により、問題がたくさん集まりました。また、定理 8.2 の証明には、かならい (@sugarknri) による助言が非常に大きな助けとなりました。その他、以下の方々にも助けをいただきました:

- minus3theta (@minus3theta)
- noicwt (@noicwt)
- p 進大好き bot (@non_archimedean)

感謝します。

最後に、これを読んでくださった読者の方に感謝を述べて、筆を置かせていただきます。

参考文献

- [1] Richard M. Karp. An introduction to randomized algorithms. *Discrete Applied Mathematics*, Vol. 34, No. 1-3, pp. 165–201, 1991.
- [2] Peter Stevenhagen. *Number Rings*. 2008.