整数論テクニック集 DRAFT

夕叢霧香 (Kirika Yuumura, @kirika_comp)

2018 年 3 月 6 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 の青から赤下位程度をターゲットにしています。

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

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の実装を与えます。ここで、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。

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;
```

```
}

int main(){
  lint a,b;
  cin>>a>>b;
  lint x,y;
  lint g=ext_gcd(a,b,x,y);
  cout<<a<<" "
      <<b<<" "<<x<<" "<<y
      <<endl;
  cout<<g<<endl;
}
```

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、 mod (10^9+7) 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 (10^9+7) するのではなく、途中結果も mod (10^9+7) することで、途中結果が大きくなりすぎるのを防ぐことができます。

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

```
typedef long long lint;

lint powmod(lint x,lint e,lint mod){
  lint prod=1%mod;
  for(int i=63;i>=0;--i){
    prod=prod*prod%mod;
    if(e&1LL<<i)prod=prod*x%mod;
  }
  return prod;
}

// 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。
lint invmod(lint a,lint p){
  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 になる。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

- 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 $\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}$ が、環準同型 $\mathbb{Z}_{(p)} \to \mathbb{Z}/p\mathbb{Z}$ に拡張できる、ということができます (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。

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 $x^e = x \cdot \cdots \cdot x$ 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    }
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): $1 \le t \le 10^{18}$, 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、およびそれ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します:

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

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

```
#include<iostream>
using namespace std;
typedef long long lint;
typedef pair<lint,lint>pll;
const lint mod=1e9+7;

lint powmod(lint x,lint e){
  lint c=1;
  for(int i=63;i>=0;--i){
    c=c*c%mod;
    if(e&1LL<<i)c=c*x%mod;
  }
  return c;
}

pll mul_pll(pll a,pll b,lint c){
  lint x=a.first*b.first%mod;
  lint nx=a.second*b.second%mod;
  x=(x+nx*c)%mod;
  lint y=(a.first*b.second+a.second*b.first)%mod;
  return pll(x,y);
}

pll pow_pll(pll a,lint e,lint c){
  pll p(1,0);
  for(int i=63;i>=0;--i){
    p=mul_pll(p,p,c);
    if(e&1LL<<i)p=mul_pll(p,a,c);
  }
  return p;
}

int main(){
  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。

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

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。

$$A = a_N f(10^{b_N}, l_N) + 10^{b_N \times l_N}(a_{N-1} f(10^{b_{N-1}}, l_{N-1}) + 10^{b_{N-1} \times l_{N-1}}(\ldots)), f(y,z) = 1 + y + \cdots + y^{z-1} = \frac{y^z - 1}{y - 1}$$

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:

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

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

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>
```

```cpp
#include<vector>
using namespace std;
typedef long long lint;
#define rep(i,n)for(int i=0;i<(int)(n);++i)

lint powmod(lint x,lint e,lint mod){
  lint prod=1%mod;
  for(int i=63;i>=0;--i){
    prod=prod*prod%mod;
    if(e&1LL<<i)prod=prod*x%mod;
  }
  return prod;
}

lint f(lint y,lint z,lint mod){
  if(z==0)return 0;
  if(z%2==0){
    lint fac=1+powmod(y,z/2,mod);
    return fac*f(y,z/2,mod)%mod;
  }
  return (1+y*f(y,z-1,mod))%mod;
}

int main(){
  int n;
  cin >> n;
  vector<lint> a(n),l(n);
  rep(i,n)cin>>a[i]>>l[i];
  lint mod;
  cin>>mod;
  lint ans=0;
  rep(i,n){
    lint enc=1;
    while(a[i]>=enc)enc*=10; // enc=10^{b_i}
    ans=ans*powmod(enc%mod,l[i],mod)%mod;
    ans=(ans+a[i]*f(enc%mod,l[i],mod))%mod;
  }
  cout<<ans<<endl;
}
```

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、a を $\bmod p$ 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、0 を除くと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 $\bmod p$ で 1 か -1 かのどちらかですが、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、平方剰余は $0 = 0^2, 1 = 1^2, 3 \equiv 4^2, 4 = 2^2, 9 = 3^2, 10 \equiv 6^2, 12 \equiv 5^2 \pmod{13}$ の 7 個です。0 を除外すると 1,3,4,9,10,12 の 6 個で、mod 13 の 0 以外の同値類 12 個のうち、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 $\bmod 13$ したもの (2,6,8,5,7,11) はすべて平方非剰余です。

5.1.3 加法群 (Lv. 4)

$\mathbb{Z}/p\mathbb{Z}$ の話 TODO 素数 p に対して、p で割った余り全体の集合を $\mathbb{Z}/p\mathbb{Z}$ と書きます。$\mathbb{Z}/p\mathbb{Z}$ は、加法を演算として、群の構造を持ちます。この群は、真の部分群が存在しない群です。

5.1.4 乗法群 (Lv. 4)

$\mathbb{Z}/p\mathbb{Z}$ のうち、0 以外の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 $(\mathbb{Z}/p\mathbb{Z})^*$ と表記し、$\mathbb{Z}/p\mathbb{Z}$ の**乗法群** (*multiplicative group*) と呼びます。

平方剰余全体の集合をここでは H と書くことにします。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 $H \cup zH = (\mathbb{Z}/p\mathbb{Z})^*$ 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、$x^2 \equiv a^{(p+1)/2}$ 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。(効率は悪い)

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: $r^2 \equiv a \pmod p$ 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. $m := s, c :\equiv z^q, t :\equiv a^q, r :\equiv a^{(q+1)/2}$ とする。以降不変量 $\boldsymbol{r^2 \equiv at \pmod p, t^{2^{m-1}} \equiv 1 \pmod p, c^{2^{m-1}} \equiv -1 \pmod p}$ 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、$t^{2^{m-2}} \equiv 1 \pmod p$ なら 4. へ、そうでなければ 5. へ行く。

4. $c :\equiv c^2 \pmod p, m := m - 1$, 6. へ行く。

5. $c :\equiv c^2 \pmod p, t :\equiv c^2 t \pmod p, r :\equiv cr \pmod p, m := m - 1$ 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。

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

$m := 3, c :\equiv 7^5 = 16807 \equiv 38, t :\equiv 8^5 = 32768 \equiv 9, r \equiv 8^3 = 512 \equiv 20$ 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、$x \equiv \pm 34 (= \mp 7)$ が答えになります。

以上のアルゴリズムで、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 のアルゴリズムです。

(TODO Wikipedia の Tonelli-Shanks の説明)

以上を使うと、次の問題が解けます。

問題

相異なることが保証されている n 個の整数 $a_1, \ldots, a_n$ が与えられる。これを並べ替えて、$x \bmod m, (x+d) \bmod m, \ldots, (x+(n-1)d) \bmod m$ 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 remoduling)

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。($n \neq 1, m-1, m, m \neq 2, 3$ 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 $(x, d) = (0, 1)$ 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("%lld%d",&m,&n);
49    vector<lint>a(n);
50    rep(i,n)scanf("%lld",&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 squsum=0;
74    rep(i,n){
75      add(sum,a[i],m);
76      add(squsum,a[i]*a[i],m);
77    }
78    lint avg=sum*powmod(n,m-2,m)%m;
79    lint vari=squsum*powmod(n,m-2,m)%m;
80    add(vari,avg*(m-avg),m);
81    lint d;
82    if(2<=n&&n<=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("%lld␣%lld\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。

6 平方剰余の相互法則 (Lv. 4)

6.1 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。

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

命題 **6.1.**

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

命題 **6.2.** 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 **6.3.** 対応 $a \mapsto \left(\frac{a}{p}\right)$ は群準同型 $(\mathbb{Z}/p\mathbb{Z})^* \to \{+1, -1\}$ を定める。

6.2 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 **6.4** (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 **6.5** (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 **6.6.** $p \neq 2, 3$ 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。

6.3 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**するといい、そうしてできた体を $K(\theta)$ と表記します。有理数体 $\mathbb{Q}$ に有理数の平方根 $\sqrt{d}$ 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。この記事では、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。

6.4 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 $\mathrm{GF}(p^e)$ と表記することにします。

$\mathrm{GF}(p^e)$ は、$\mathrm{GF}(p)$ 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、$\mathrm{GF}(p^e) \cong \mathrm{GF}(p)(\theta)$ です。

6.5 フロベニウス写像

$\mathrm{Frob}\colon \mathrm{GF}(p^e) \to \mathrm{GF}(p^e), \mathrm{Frob}(x) := x^p$ を**フロベニウス写像**と呼びます。 TODO

命題 **6.7.** Frob は e 回適用すると元に戻る。つまり、$\mathrm{Frob}^e(x) = x$。

命題 **6.8.** $x, \mathrm{Frob}(x), \mathrm{Frob}^2(x), \ldots, \mathrm{Frob}^{e-1}(x)$ は全て共役 (TODO definition)。つまり、TODO。

6.6 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 $\left(\frac{3}{M}\right) = 1$ なら)、議論は $\mathrm{GF}(M)$ の中で完結できます。3 が $\bmod M$ で平方非剰余 (つまり $\left(\frac{3}{M}\right) = -1$) の場合を考えます。このとき、$\mathrm{GF}(M)$ に $\sqrt{3}$ を添加して拡大したものは、$\mathrm{GF}(M^2)$ と同型になります。

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

ここで、$\mathrm{Frob}(2+\sqrt{3}) = (2+\sqrt{3})^M \in \mathrm{GF}(M^2)$ 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、$\mathrm{Frob}(2+\sqrt{3}) = 2-\sqrt{3}$ 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。)

ソースコード 10　yukicoder-613.java

```
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;
```

```
    }
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        long n=scan.nextLong();
        long m=scan.nextLong();
        long p=m==2||m%12==1||m%12==11?m-1:m+1;
        long e=powerMod(2,n,p);
        long x=1,y=0;
        for(int i=31;i>=0;--i){
            long z=(x*x+3*y*y)%m,w=2*x*y%m;
            x=z;y=w;
            if((e&1L<<i)!=0){
                z=(2*x+3*y)%m;w=(x+2*y)%m;
                x=z;y=w;
            }
        }
        System.out.println((2*x+m-2)%m);
    }
}
```

7 mod p のアルゴリズム その 2

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

定理 **7.1.** p を奇素数とする。 mod p の 0 以外の平方剰余を $a_1, \ldots, a_{\frac{p-1}{2}}$ 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 8.1 と同じ論法が使えます。 □

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

Algorithm 2 [1, Algorithm 3.1]

入力: p: 奇素数, a: $1 \le a < p$ 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 (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。

1. での選択にかかる時間の期待値が定数であることは、以下の定理で証明できます。

定理 **7.2** ([1, Theorem 3.2]). $a \in (\mathbb{Z}/p\mathbb{Z})^*$ を平方剰余とする。このとき、$c \in (\mathbb{Z}/p\mathbb{Z})^*$ がランダムに選ばれるならば、確率 $(p-3)/2(p-1)$ 以上で $\left(\frac{c^2-a}{p}\right) = -1$ が成立する。

証明. むずかしいノエ TODO □

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 6.2 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。$x^{\frac{p-1}{2}}-1$ は全ての平方剰余 a に対する $x-a$ の積なので、最大公約数をとると平方剰余の方だけが残ります。

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。これは行列累乗などを用いることで $O(\log p)$ 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 TODO

7.2 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 3 Cipolla のアルゴリズム

入力: p: 奇素数, a: $1 \le a < p$ 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 $\mathrm{GF}(p)(\theta)$ を考える。$(c+\theta)^{\frac{p+1}{2}}$ は $\mathrm{GF}(p)$ の元であり、これが答え (平方根の一つ) を与える。

命題 **7.3.** Algorithm 3 は正しい解を与える。

証明. 6.6 でみたように、$(c+\theta)^p=c-\theta$ となるのは自明です。よって $(c+\theta)^{p+1}=c^2-\theta^2=a$ です。よって $(c+\theta)^{\frac{p+1}{2}} \in \mathrm{GF}(p)$ が示せれば良いです。

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 $\mathrm{GF}(p)$ 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 **7.4.** あるいは、これは単に、$\mathrm{GF}(p)(\theta)$ が体であるため、方程式 $x^2-a=0$ は $\mathrm{GF}(p)(\theta)$ 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 $\mathrm{GF}(p)$ 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 11　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
```

```
lint powmod(lint a,lint e,lint p){
  a%=p;
  lint ans=1;
  for(int i=63;i>=0;--i){
    ans=ans*ans%p;
    if(e&1LL<<i)ans=ans*a%p;
  }
  return ans;
}

// 2次体上の掛け算
pll mul_quad(pll a,pll b,lint theta,lint p){
  lint x=a.first*b.first+(theta*a.second%p)*b.second;
  x%=p;
  lint y=a.first*b.second+a.second*b.first;
  y%=p;
  return pll(x,y);
}

// 2次体上の冪乗
pll pow_quad(pll a,lint e,lint theta,lint p){
  pll ans(1,0);
  for(int i=63;i>=0;--i){
    ans=mul_quad(ans,ans,theta,p);
    if(e&1LL<<i)ans=mul_quad(ans,a,theta,p);
  }
  return ans;
}

// p:奇素数, a:mod p の平方剰余であることが要請される。
lint cipolla(lint p,lint a){
  a%=p;
  if(a==0)return 0;
  lint c,t;
  mt19937 mt;
  do{
    c=mt()%p;
    t=(c*c+p-a)%p;
  }while(powmod(t,(p-1)/2,p)!=p-1);
  pll ans=pow_quad(pll(c,1),(p+1)/2,t,p);// 証明中のx+y*theta
  return ans.first;
}
```

注意 **7.5.** 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 7.3 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 11 の実装では、常に 0 が返ることになります。

8 多項式を使うテク (Lv. 4)

8.1 高速フーリエ変換 (FFT) (Lv. 3)

数列 $a_0, a_1, \ldots, a_{n-1}$ と $b_0, b_1, \ldots, b_{n-1}$ に対して、a と b の**畳み込み** (*convolution*) とは、数列 $c_i := \sum_{j=0}^{i} a_j b_{i-j}$ のことです。多項式の積の係数も、畳み込みだと思うことができます。

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 $du^2 < 2^{53}$ でないといけないからです。$d \leq 10^5, u \leq 10^5$ くらいでギリギリでしょう。

一応実装を載せておきます。

ソースコード 12　fft.cpp

```
#include<cmath>
#include<complex>
#include<iostream>
#include<vector>
using namespace std;
typedef long long lint;
#define rep(i,n)for(int i=0;i<(int)(n);++i)

typedef complex<double> comp;

const double pi=acos(-1);

// n は 2冪で、a.size()==n
void fft(int n,vector<comp> &a,double dir) {
  // ビット反転は http://math314.hateblo.jp/entry/2015/05/07/014908 を参考にしている。
  int i = 0;
  for (int j = 1; j < n - 1; ++j) {
    for (int k = n >> 1; k >(i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
  vector<comp> zeta_pow(n);
  rep(i,n){
    double theta=pi/n*i*dir;
    zeta_pow[i]=comp(cos(theta),sin(theta));// 毎回計算することで、誤差を回避する。
  }
  // ここも http://math314.hateblo.jp/entry/2015/05/07/014908 を参考にしている。
  for(int m=1;m<n;m*=2){
    for(int y=0;y<m;++y){
      comp fac=zeta_pow[n/m*y];
      for(int x=0;x<n;x+=2*m){
        int u=x+y;
        int v=x+y+m;
        comp s=a[u]+fac*a[v];
        comp t=a[u]-fac*a[v];
        a[u]=s;a[v]=t;
      }
    }
  }
}

template<class T>
vector<comp> convolution(vector<T> a,vector<T> b){
  int n=1;
  while(n<(int)a.size()+(int)b.size())n*=2;
  vector<comp>a_(n),b_(n);
  rep(i,a.size())a_[i]=a[i];
  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 <= 10^15 でなければ精度が保証されない。
57  vector<lint> integral_convolution(vector<lint> a,vector<lint> b){
58    vector<comp>a_(a.size()),b_(b.size());
59    rep(i,a.size())a_[i]=a[i];
60    rep(i,b.size())b_[i]=b[i];
61    vector<comp>ans=convolution(a_,b_);
62    vector<lint>ret(ans.size());
63    rep(i,ans.size())ret[i]=floor(ans[i].real()+0.5);
64    return ret;
65  }
66
67  int main(){
68    vector<lint> a(3);
69    a[0]=1,a[1]=2,a[2]=3;
70    vector<lint> b(5);
71    b[0]=1,b[1]=10,b[2]=15,b[3]=11,b[4]=1;
72    rep(i,5)b[i]=-b[i];
73    vector<lint> ab=integral_convolution(a,b);
74    // -1 -12 -38 -71 -68 -35 -3 0
75    rep(i,ab.size())cout<<" "<<ab[i];
76    cout<<endl;
77  }
```

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。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` を見てください。

8.2 フェルマーの小定理 (Lv. 3)

定理 **8.1.** 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、$x+k \mid x^{p-1}-1$ が成り立ちます (整除は $\mathrm{GF}(p)[x]$ の上のもの)。以上より、$(x+1)\times(x+2)\times\cdots\times(x+(p-1)) \mid x^{p-1}-1$ 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。 □

注意として、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、$(x+1)(x+3)(x+5)(x+7) \equiv x^4 + 6x^2 + 1 \not\equiv x^{\varphi(8)} - 1 \pmod{8}$ です。これは $\mathbb{Z}/p\mathbb{Z}$ が体にならず、$(\mathbb{Z}/p\mathbb{Z})[x]$

において (0 以外の) 0 次多項式が必ずしも可逆元ではないことに由来します。

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。

問題

正の整数 n と素数 p が与えられる。$[n] := \{1, \ldots, n\}$ 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、$\bmod 10^9 + 7$ 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、

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

であることがわかります。$P(n) := (t+1) \times (2t+1) \times \cdots \times (nt+1)$ 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:

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

$(kt+1) \bmod p$ は周期 p なので、$P(pk) \equiv P(p)^k = (-t^{p-1}+1)^k \pmod{p}$ 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 の倍数かどうかにだけ関心があるので、$(t^{p-1}+1)^q$ 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 $a = (d_{e-1}d_{e-2}\cdots d_1 d_0)_p$ とすると、$(x+1)^a \bmod p$ の 0 でない係数は、ちょうど
$(d_{e-1}+1) \times (d_{e-2}+1) \times \cdots \times (d_0+1)$ 個ある。

これに $x = t^{p-1}$ を代入すると、$q = (d_{e-1}d_{e-2}\cdots d_1 d_0)_p$ として、$(-t^{p-1}+1)^q \bmod p$ の 0 でない係数は $(d_{e-1}+1) \times (d_{e-2}+1) \times \cdots \times (d_0+1)$ 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 $(d_{e-1}+1) \times (d_{e-2}+1) \times \cdots \times (d_0+1) \times (P(r) \bmod p \text{ の } 0 \text{ でない係数 })$ 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/17576449` 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 12 と同じです。と 25 行目において、1 の n 乗根 ζ の i 乗を愚直に計算していることに注意しましょう。(これをサボろうとすると誤差死します。ていうかしました)

ソースコード 13　FEB18-LUCASTH.cpp

```
#include<cmath>
#include<complex>
#include<iostream>
#include<string>
#include<vector>
using namespace std;
typedef long long lint;
#define rep(i,n)for(int i=0;i<(int)(n);++i)

typedef complex<double> comp;

const double pi=acos(-1);

// n は 2冪で、a.size()==n
void fft(int n,vector<comp> &a,double dir) {
  // ビット反転は http://math314.hateblo.jp/entry/2015/05/07/014908 を参考にしている。
  int i = 0;
  for (int j = 1; j < n - 1; ++j) {
    for (int k = n >> 1; k >(i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
  vector<comp> zeta_pow(n);
  rep(i,n){
    double theta=pi/n*i*dir;
    zeta_pow[i]=comp(cos(theta),sin(theta));// 毎回計算することで、誤差を回避する。
  }
  // ここも http://math314.hateblo.jp/entry/2015/05/07/014908 を参考にしている。
  for(int m=1;m<n;m*=2){
    for(int y=0;y<m;++y){
      comp fac=zeta_pow[n/m*y];
      for(int x=0;x<n;x+=2*m){
        int u=x+y;
        int v=x+y+m;
        comp s=a[u]+fac*a[v];
        comp t=a[u]-fac*a[v];
        a[u]=s;a[v]=t;
      }
    }
  }
}

template<class T>
vector<comp> convolution(vector<T> a,vector<T> b){
  int n=1;
  while(n<(int)a.size()+(int)b.size())n*=2;
  vector<comp>a_(n),b_(n);
  rep(i,a.size())a_[i]=a[i];
  rep(i,b.size())b_[i]=b[i];
  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 <= 10^15 でなければ精度が保証されない。
58  vector<lint> integral_convolution(vector<lint> a,vector<lint> b){
59    vector<comp>a_(a.size()),b_(b.size());
60    rep(i,a.size())a_[i]=a[i];
61    rep(i,b.size())b_[i]=b[i];
62    vector<comp>ans=convolution(a_,b_);
63    vector<lint>ret(ans.size());
64    rep(i,ans.size())ret[i]=floor(ans[i].real()+0.5);
65    return ret;
66  }
67
68  pair<string,lint> divide(const string &n,lint r){
69    string res;
70    lint rem=0;
71    bool cont_zero=1;
72    rep(i,n.size()){
73      rem=10*rem+(n[i]-'0');
74      lint q=rem/r;
75      rem%=r;
76      if(cont_zero&&q==0)continue;
77      if(q!=0)cont_zero=0;
78      res+='0'+q;
79    }
80    return make_pair(res,rem);
81  }
82
83
84  vector<lint> parse(const string &n,lint r){
85    vector<lint> dig;
86    string cur(n);
87    while(1){
88      pair<string,lint>sub=divide(cur,r);
89      dig.push_back(sub.second);
90      cur=sub.first;
91      if(cur=="")break;
92    }
93    return dig;
94  }
95
96  vector<lint> g(lint p,lint l,lint r){
97    if(l>r){
98      return vector<lint>(1,1);
99    }
100   if(l==r){
101     vector<lint> ret(2);
102     ret[0]=1;
103     ret[1]=l%p;
104     return ret;
105   }
106   lint mid=(l+r+1)/2;
```

```
107     vector<lint>fst=g(p,l,mid-1);
108     vector<lint>snd=g(p,mid,r);
109     vector<lint>ans=integral_convolution(fst,snd);
110     rep(i,ans.size())ans[i]%=p;
111     return ans;
112   }
113
114
115   lint f(string n,lint p){
116     const lint mod=1e9+7;
117     vector<lint> dig=parse(n,p);
118     if(dig[0]==p-1){
119       //propagate
120       dig[0]=0;
121       int car=1;
122       for(int pos=1;pos<(int)dig.size();++pos){
123         dig[pos]+=car;
124         car=dig[pos]/p;
125         dig[pos]%=p;
126       }
127       if(car>0)dig.push_back(car);
128     }
129     vector<lint> ans=g(p,1,dig[0]);
130     lint ret=0;
131     rep(i,dig[0]+1)
132       if(ans[i]!=0)ret++;
133     rep(i,dig.size()-1)
134       ret=ret*(dig[i+1]+1)%mod;
135     return ret;
136   }
137
138   int main(){
139     int t;
140     cin>>t;
141     while(t--){
142       string n;
143       lint p;
144       cin>>n>>p;
145       cout<<f(n,p)<<endl;
146     }
147   }
```

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 2.22sec 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。)

8.3 原始根 (Lv. 4)

http://techtipshoge.blogspot.jp/2012/04/facebook-hacker-cup-2011-round2-scott.html
まだ理解していないちょく

9　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 **9.1.** 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 (x, y) 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。

Algorithm 4 ペル方程式

入力: d: 正の整数、平方数でない
出力: x, y: 正の整数, $x^2 - dy^2 = 1$ 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): $q_0, q_1, \ldots, q_{m-1}, q_m, q_1, \ldots$ となる。このような最小の m をとり、

$$b := [q_0; q_1, \ldots, q_{m-1}] = q_0 + \cfrac{1}{q_1 + \cfrac{1}{\cdots + \cfrac{1}{q_{m-1}}}}$$

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、x, y を出力する。$x^2 - dy^2 = -1$ の場合、$x^2 + dy^2, 2xy$ を出力する。

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見ましょう。数列 α_i と q_i は以下のようになります:

$$\{a_i\} : \sqrt{14}, \frac{3+\sqrt{14}}{5}, \frac{2+\sqrt{14}}{2}, \frac{2+\sqrt{14}}{5}, 3+\sqrt{14}, \frac{3+\sqrt{14}}{5}, \frac{2+\sqrt{14}}{2}, \ldots$$

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

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、

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

が成立します。$15^2 - 14 \times 4^2 = 1$ であるため、出力は $15, 4$ 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 14　aoj-2116.cpp

```
#include<iostream>
#include<vector>
#include<set>
#include<cassert>
#include<cmath>
using namespace std;
typedef long long lint;
typedef pair<lint,lint>pll;

// 連分数を使って、基本単数を求める。d は平方数でないことが要請される。
pll fundamental_unit(lint d){
  vector<int> ans;
  lint x=0,z=1;//(x+sqrt(d))/z
  lint sqrtd=floor(sqrt(d));
  set<pll>seen;
  // invariants: x>=0, z|x*x-d
  while(1){
    if(seen.count(pll(x,z)))break;
    seen.insert(pll(x,z));
    lint q=(x+sqrtd)/z;
    ans.push_back(q);
    x=q*z-x;
    lint norm=x*x-d;
    z=-norm/z;
  }
  // recover
  lint num=0,den=1;
  for(int i=(int)ans.size()-2;i>=0;--i){
    lint z=num+ans[i]*den;
    num=den;den=z;
  }
  if(den*den-d*num*num==-1){
    lint x=den*den+d*num*num;
    lint y=2*den*num;
    den=x;num=y;
  }
```

```
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 9.1 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、$U \cong$ (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 (x, y) 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。

10　単項イデアル整域 (Lv. 5)

定義 **10.1.** 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 **10.2.** 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:

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

定義 **10.3.** 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 **10.4.** **単項イデアル整域** (*principal ideal domain*, PID) (独: *Hauptidealbereich*, ハウプトイデアールベライヒ) 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 **10.5.** PID の例として有名なのは、$\mathbb{Z}$, $\mathbb{Z}[i]$ (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) に到達可能として、$x+y\sqrt{-1}$ 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、他でもないイデアル $(P+Q\sqrt{-1}, P-Q\sqrt{-1})$ 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、$\gamma | X_i+Y_i\sqrt{-1}$ かどうかの判定を行うことで、この問題が解けました。

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:

- 黒板にある数を一つ選び、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 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。
(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、GF(2)[x] の元 (つまり、GF(2) 係数の多項式) として表すという発想が自然です:

$$t=b_u2^u+b_{u-1}2^{u-1}+\cdots+b_12+b_0 \mapsto b_ux^u+b_{u-1}x^{u-1}+\cdots b_1x+b_0$$

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

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

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のようになります。`https://arc084.contest.atcoder.jp/submissions/2137652` です。

ソースコード 15　ARC084F.cpp

```cpp
#include<algorithm>
#include<bitset>
#include<cassert>
#include<iostream>
#include<vector>
using namespace std;
typedef long long lint;
#define rep(i,n)for(int i=0;i<(int)(n);++i)

const int N=4000;
typedef bitset<N> bs;
const lint MOD=998244353;

bs read(){
  string s;
  cin>>s;
  int l=s.length();
  bs ret;
  rep(i,l)ret[i]=s[l-i-1]=='1';
  return ret;
}

// a%b
bs rem(bs a,bs b){
  int hi=-1;
  rep(i,N)
    if(b[i])hi=i;
  assert(hi>=0);
  for(int i=N-hi-1;i>=0;--i)
    if(a[i+hi])a^=b<<i;
  return a;
}

bs gcd(bs a,bs b){
  while(b.count()!=0){
    a=rem(a,b);
    swap(a,b);
  }
  return a;
}

int main(){
  int n;
  cin>>n;
  bs x=read();
  vector<bs> s(n);
  rep(i,n)s[i]=read();
  bs g=s[n-1];
  rep(i,n-1)g=gcd(s[i],g);
  lint ans=0;
  lint cur=1;
  int hi=-1;
  rep(i,N)
    if(g[i])hi=i;
  rep(i,N-hi){
    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

謝辞

全ての助けの中で、けんちょん (@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。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)
- かならい (@sugarknri)

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.