> **写在最前面：本文部分内容来自网上各大博客或是各类图书，由我个人整理，增加些许见解，仅做学习交流使用，无任何商业用途。因个人实力时间等原因，本文并非完全原创，请大家见谅。**

《算法竞赛中的初等数论》正文 0x00整除、0x10 整除相关（ACM / OI / MO）（十五万字符数论书）

《算法竞赛中的初等数论》（信奥 / 数竞 / ACM）前言、后记、目录索引（十五万字符的数论书）

全文目录索引链接： https://fanfansann.blog.csdn.net/article/details/113765056

0x20 同余

0x21 整数的取余运算

0x21.1 整数的取余运算（模运算）

定义：带余除法，设 a,b 是整数，且 $b>0$，使得 $a=bq+r$，且 $0\leq r<b$，称 q 为商， r 为余数。

显然带余除法中的商和余数都是唯一的，在下文中将商记为 a/b，将余数记为 $a\ \%\ b$，`/` 与 `%` 的运算优先级与乘除法相同。

定义以下运算：

- **取模运算**：$a\%p$ (或$a\ mod\ p$)，表示 a 除以 p 的余数。
- **模 p 加法**：$(a+b)\%p$，其结果是$a+b$算术和除以p的余数，也就是说，$(a+b)=kp+r$，则$(a+b)\%p=r$。
- **模 p 减法**：$(a-b)\%p$，其结果是$a-b$算术差除以p的余数。
- **模 p 乘法**：$(a*b)\%p$，其结果是$a*b$算术乘法除以p的余数。

模运算有如下简单性质：

- $a\%b = a - b \times \lfloor \frac{a}{b} \rfloor$，即若 $a\%b = c$，则 $a = bx + c$，$x = \lfloor \frac{a}{b} \rfloor$。
- $n\%p$ 得到结果的正负由被除数 n 决定,与 p 无关。

> 例如：$7\%4 = 3$，$-7\%4 = -3$，$7\%-4 = 3$，$-7\%-4 = -3$

- **若 $p \mid (a-b)$，则 $a \equiv b \pmod{p}$。** 例如 $11 \equiv 4 \pmod{7}$，$18 \equiv 4 \pmod{7}$（符号 `|` 指整除）
- **结合率：** $((a+b)\%p + c)\%p = (a + (b+c)\%p)\%p$，$((a \times b)\%p \times c)\%p = (a \times (b \times c)\%p)\%p$
- **交换率：** $(a+b)\%p = (b+a)\%p$ ，$(a \times b)\%p = (b \times a)\%p$
- **分配率：** $((a+b)\%p \times c)\%p = ((a \times c)\%p + (b \times c)\%p)\%p$
- 若$a\%p = x, a\%q = x, \gcd(p,q) = 1$，则$a\%(p \times q) = x$

更多关于模运算性质以及同余详见本文 **0x22同余**。

0x21.2 整数模意义下的加减乘乘方运算

- $(a+b)\%c = (a\%c + b\%c)\%c$
- $(a-b)\%c = (a\%c - b\%c + c)\%c$
- $(a \times b)\%c = (a\%c) \times (b\%c)\%c$
- $(a^b)\%p = ((a\%p)^b)\%p$

计算减法的时候，通常需要加上模数 c，防止出现负数。

- **经典快速幂**

给定整数 a，正整数 n，以及非零整数 p，求 $a^n\%p$。利用模 p 乘法，这个问题可以递归求解，即令 $f(n) = a^n\%p$，那么$f(n-1) = a^{n-1}\%p$，$f(n) = a \times f(n-1)\%p$，这样就转化成了递归式。但是递归求解的时间复杂度为 $O(n)$，往往当 n 很大的时候就很难在规定时间内得到解了。

当 n 为偶数时，我们可以将 $a^n\%p$ 拆成两部分，令$b = a^{\frac{n}{2}}\%p$，则 $a^n\%p = b \times b\%p$。

当 n 为奇数时，可以拆成三部分，令，$b = a^{\frac{n}{2}}\%p$，则$a^n\%p = a \times b \times b\%p$;

上述两个等式中的 b 可以通过递归计算，由于每次都是除 2 ，所以时间复杂度是 $O(logn)$。

```
ll qpow(ll a, ll b, ll q)
{
    ll res = 1;//因为是用乘法模拟乘方，所以res要是1
    while(b) {
        if(b & 1)res = (res * a) % q;
        a = (a * a) % q;//视情况将 * 换成Mul(龟速乘)
        b >>= 1;
    }
    return res % q;
}
```

- **龟速乘**

在模意义下计算乘法，如果 c 较大（但是不超过 $long\ long$ 范围），进行乘法的两个数同样很大，直接**乘会爆掉**（例如快速幂里的乘法），我们可以用类似快速幂的快速乘计算，时间复杂度为 $O(logb)$。因为慢于 $O(1)$ 的乘法运算符，所以我们常常把这个叫做龟速乘。经常用与快速幂中代替普通乘法。

```
ll Mul(ll a, ll b, ll p)
{
    if(b < 0) a = - a, b = - b;
    ll res = 0;//因为是加法模拟乘法，所以res开始为0
    while(b) {
        if(b & 1) res = (res + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return res;
}
```

这样普通的快速幂会变成 $O(log^2 n)$ 。如果需要更快的快速乘，可以用 `long double` 数据类型进行计算，复杂度$O(1)$。（ `long double` 有15/12 bit，可以处理范围在$1.7 \times 10^{-308} \sim 1.7 \times 10^{308}$ 的数据 ）

```cpp
ll Mul(ll a, ll b, ll p) {
    if(b < 0) a = - a, b = - b;
    if (p ≤ 1000000000) return a * b % p;
    else if ( p ≤ 1000000000000ll) return (((a * (b >> 20) % p) << 20) + (a * (b & ((1 << 20) - 1)))) % p;
    else {
        ll d = (ll)floor(a * (long double)b / p + 0.5);
        ll res = (a * b - d * p) % p;
        if (res < 0) res += p;
        return res;
    }
}
```

或者换一种好写的方法：

```cpp
inline ll Mul(ll x,ll y,ll p)
{
    if(y < 0) x = - x, y = - y;
    ll z = (long double)x / p * y;
    ll res = (unsigned long long)x * y - (unsigned long long)z * p;
    return (res + p) % p;
}
```

$O(n \log n)$ 的大数快速幂：

```cpp
#include <bits/stdc++.h>
using namespace std;
const int mod = 1e9 + 7;
long long quick_mod(long long a, long long b) {
    long long ans = 1;
    while (b) {
        if (b & 1) {
            ans = (ans * a) % mod;
            b--;
        }
        b /= 2;
        a = a * a % mod;
    }
    return ans;
}//内部快速幂
long long quickmod(long long a, char *b, int len) {
    long long ans = 1;
    while (len > 0) {
        if (b[len - 1] != '0') {
            int s = b[len - 1] - '0';
            ans = ans * quick_mod(a, s) % mod;
        }
        a = quick_mod(a, 10) % mod;
        len--;
    }
    return ans;
}

int main() {
    char s[100050];
    int a;
    while (~scanf("%d", &a)) {      //a ^ s % mod
        scanf("%s", s);
        int len = strlen(s);
        printf("%I64d\n", quickmod(a, s, len));
    }
    return 0;
}
```

0x22 同余

若正整数 a 和 b 除以 m 的**余数相等**，则称 a，b 模 m 同余，记作 $a \equiv b \pmod m$。

即 $a\%m = b\%m$。

- **竞赛例题选讲**

Problem A 循环节

$f[1] = a$，$f[2] = b$，$f[3] = c$，当 $n > 3$ 时 $f[n] = (A \times f[n-1] + B \times f[n-2] + C \times f[n-3])\%53$，给定$a, b, c, A, B, C$，求$f[n]$ （$n < 2^{31}$）。

Solution

由于 n 非常大，循环模拟求解肯定是不现实的，仔细观察可以发现当 $n > 3$ 时，$f[n]$ 的值域为 $[0, 53)$，并且连续三个数 $f[n-1]$、$f[n-2]$、$f[n-3]$ 一旦确定，那么 $f[n]$ 也就确定了，而 $f[n-1]$、$f[n-2]$、$f[n-3]$ 这三个数的组合数为 $53 \times 53 \times 53$ 种情况，那么对于一个下标 $k < n$，假设 $f[k]$ 已经求出，并且满足 $f[k-1] == f[n-1]$ 且 $f[k-2] == f[n-2]$ 且 $f[k-3] == f[n-3]$，则 $f[n]$ 必定等于 $f[k]$，这里的 $f[k \cdots n-1]$ 就被称为这个数列的循环节。

显然，在 $53 \times 53 \times 53$ 次计算之内必定能够找到循环节。打表找规律即可。

0x21.1 同余的性质

同余的基本性质：

性质21.1.1： (自反性)：$a \equiv a(\mod m)$

性质21.1.2： (对称性)：若 $a \equiv b(\mod m)$，则 $b \equiv a(\mod m)$

性质21.1.3： (传递性)：若 $a \equiv b(\mod m), b \equiv c(\mod m)$，则 $a \equiv c(\mod m)$

性质21.1.4： (同加性)：若 $a \equiv b(\mod m)$，则 $a \pm c \equiv b \pm c(\mod m)$

性质21.1.5： (同乘性)：若 $a \equiv b(\mod m)$，则 $a \times c \equiv b \times c(\mod m)$，若 $a \equiv b(\mod m), c \equiv d(\mod m)$，则 $a \times c \equiv b \times d(\mod m)$

性质21.1.6： (同幂性)：若 $a \equiv b(\mod m)$，则 $a^c \equiv b^c(\mod m)$

性质21.1.7： (不满足同除性)：若 $a \equiv b(\mod m)$不满足$a \div c \equiv b \div c(\mod m)$

性质21.1.8： (满足同除性)：若 $a \equiv b(\mod m)$，$c \mid a$，$c \mid b$，则 $\frac{a}{c} \equiv \frac{b}{c} \pmod{\frac{m}{\gcd(m,c)}}$。

或者可以换一种表述方式：若 $ca \equiv cb(\mod m)$ 则 $a \equiv b \pmod{\frac{m}{\gcd(m,c)}}$

例如：$ca \equiv cb \pmod m, \gcd(c, m) = 1 \Longrightarrow a \equiv b \pmod m$

该性质会在取遍剩余系会用到。

推论21.1.9： 若 $a \equiv b(\mod m)$，$m' \mid m$，$a \equiv b \pmod{m'}$

> 大致证明（随便写的，不严谨，大致是这么个意思）
> 若 $a \equiv b \pmod m$ 显然有：$a = k_1 m + b$ 若 $m' \mid m$ 则 $m = k_2 m'$，则 $a = k_1 k_2 m' + b$
> 若 $b \leq m'$ $a \mod m' = k_1 k_2 m' + b \mod m' = b$ 即：$a \equiv b \pmod m'$
>
> 若 $b > m'$ 设 $b = k_3 m' + r$ $a \pmod m' = k_1 k_2 m' + k_3 m' + r = r$ $b \pmod m' = k_3 m' + r = r$ 即：$a \equiv b \pmod m'$
>
> 综上所诉，若 $a \equiv b \pmod m$，$m' \mid m$，则 $a \equiv b \pmod{m'}$

推论21.1.10： $a \equiv b \pmod{m_i}(i = 1..k)$ 等价于 $a \equiv b \pmod M = \operatorname{lcm}(m1, m2, ..mk)$

推论21.1.12： $\begin{cases} a \equiv b \pmod m \\ c \equiv d \pmod m \end{cases} \Longrightarrow a + c \equiv b + d \pmod m$

推论21.1.13： $\begin{cases} a \equiv b \pmod m \\ c \equiv d \pmod m \end{cases} \Longrightarrow a - c \equiv b - d \pmod m$

注意：`-4 % 5 = -4, -6 % 5 = -1`，所以如果题目要求的是最小正整数那么我们就需要对答案x：`(x % b + b) % b`

同余类与剩余系

对于 $\forall a \in [0, m-1]$，集合 $\{a + km\}(k \in \mathbb{Z})$ 的所有数模 m 同余，余数都是 a，该集合称为一个模 m 的**同余类**，简记为 $\bar{a}$。

模 m 的同余类显然一共有 m 个，分别为 $\bar{0}, \bar{1}, \bar{2}, \cdots, \overline{m-1}$，他们构成了 m 的**完全剩余系**

$1 \sim m$ 中与 m 互质的数代表的同余类共有 $\varphi(m)$ ，他们构成了 m 的**简化剩余系**。

例如模 10 的完全剩余系为 $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$，模 10 的简化剩余系为 $\{\bar{1}, \bar{3}, \bar{7}, \bar{9}\}$。

简化剩余系关于模 m 乘法封闭，因为若 $a, b(1 \leq a, b \leq m)$ 与 m 互质，则 $a \times b$ 显然也不会含有与 m 相同的质因子，即 $a \times b$ 与 m 互质。由余数的定义可得到 $a \times b \mod m$ 也与 m 互质，即 $a \times b \mod m$ 也属于 m 的简化剩余系。

即模 m 的简化剩余系关于模 m 乘法封闭。

- **重要性质定理**

性质21.11： 模 p 的剩余系$\{rk\}$，若 $\gcd(p, d) = 1$ 那么 $\{d \times rk\}$ 也是模 p 的剩余系

很重要的性质，如果求 $ax \mod p$ 的既约剩余系的大小，我们只要利用同余式的性质1

转化为 $x \times \frac{a}{\gcd(a,p)} \mod (\frac{p}{\gcd(a,p)})$，根据这个理论 x 取值个数就是 $\frac{p}{\gcd(a,p)}$

性质21.12： 设模 $m1$ 的既约剩余系为 R_1, m_2 的既约剩余系为 R_2，m_1, m_2 互质，**则模 m_1m_2 的既约剩余系为** $R = \{x = m_2x_1 + m_1x_2 (\mod m_1m_2) | x_1 \in R_1, x_2 \in R_2\}$

这个结论证明可以考虑任意两个 x 互不同余即可（作差判断）

这就告诉我们若 $S(M)$ 表示 M 既约剩余系的大小的话，$S(M) = S(m1) \times S(m2)$

这个结论推广到 k 维就是,设模 m_k 的既约剩余系为 R_k, m_k 两两互质，$M = \prod m_i$ 则模 M 的既约剩余系为 $R = \{\sum M \times m_i^{-1} \times a_i (\mod M) | x_i \in R_i\}$（$m_i^{-1}$ 指 m_i 的逆元）

因此以后我们求模 M 的剩余系大小，可以对 M 质因数分解分别求出后再相乘。

0x21.2 费马小定理

若 p 是质数，则对于任意的整数 a 都有 $a^p \equiv a \pmod p$。若 $\gcd(a, p) = 1$，即 a 不是 p 的倍数，则有 $a^{p-1} \equiv 1 \pmod p$。

> 证明
> 因为 p 是质数，且$(a, p) = 1$，所以 $\varphi(p) = p - 1$。
> 由欧拉定理可得 $a^{p-1} \equiv 1 \pmod p$。证毕。（欧拉定理证明见下一小节）
> 对于该式又有 $a^p \equiv a \pmod p$，而且此式不需要 $(a, p) = 1$，
> 所以，费马小定理的另一种表述为：假如 p 是质数， a 是整数，那么$a^p \equiv a \pmod p$。

费马小定理降幂： $a^k \equiv a^{k \mod (p-1)} (\mod p)$（$a$ 与 p 互质）

费马大定理：

- $m > 2$时， $x^m + y^m = z^m$ 无正整数解
- 当$m = 2$，对于式子$a^2 + b^2 = c^2$（n为任意正整数）：
  - 当 a 为奇数时： $a = 2n + 1, c = n^2 + (n + 1)^2, b = c - 1$
  - 当 a 为偶数时：$a = 2n + 2, c = 1 + (n - 1)^2, b = c - 2$

性质21.2.1： 对于任意多项式 $F(x) = \sum_{i=0}^{\infty} a_i x^i$， （$a_i$ 对一个质数 P 取模），若满足 $a_0 \equiv 1 (\mod P)$，则 $\forall n \leqslant P, F^P(x) \equiv 1 (\mod x^n)$ (例题)

0x21.3 欧拉定理

- **欧拉定理**

定理21.3.1： 若正整数 a, n 互质，则 $a^{\varphi(n)} \equiv 1 (\mod n)$ 其中 $\varphi(n)$ 是欧拉函数。

> 证明
> 设 $x_1, x_2, \cdots, x_{\varphi(n)}$是一个以 n 为模的简化剩余系， 则 ax_1， $ax_2, \cdots, ax_{\varphi(n)}$ 也是一个以 n 为模的简化剩余系（因为$(a， n) = 1$）。 于是有 ax_1， $ax_2, \cdots, ax_{\varphi(n)} \equiv x_1, x_2, \cdots, x_{\varphi(n)} \pmod n$， 所以 $a^{\varphi(n)} \equiv 1 \pmod n$。
> 证毕。

推论21.3.2： $\exists x \in N^*, a^x \equiv 1 (\mod m) \iff \gcd(a, m) = 1$ (证明) (例题)

- **竞赛例题选讲**

Problem A 好大好大好大好大

整数 a 和 n 互素，求 a 的 k 次幂模 n ，其中$k = X^Y$，正整数a, n, X, Y（$X, Y <= 10^9$）为给定值。

Solution

好大好大，问题要求的是 $a^k \% n, k = X^Y$，指数上还是存在指数，需要将指数化简，注意到 a 和 n 互素，所以可以利用欧拉定理，令 $X^Y = k\varphi(n) + r$，那么 $k\varphi(n)$ 部分并不需要考虑，因为他们都与 1 膜 n 同余。问题转化成求 $r = X^Y \% \varphi(n)$，可以采用快速幂取模，得到 r 后再采用快速幂取模求解 $a^r \% n$。这种思想其实就是拓展欧拉定理，也叫欧拉降幂。

- **欧拉降幂（拓展欧拉定理）**

若 a 与 m 互质： $a^b \equiv a^{b \mod \varphi(m)} \pmod m$

> **证明**：
>
> 设 $b = q \times \varphi(m) + r$，其中 $0 \leq r \leq \varphi(m)$，即 $r = b \mod \varphi(m)$。
> $a^b \equiv a^{q \times \varphi(m) + r} \equiv (a^{\varphi(m)})^q \times a^r = 1^q \times a^r \equiv a^r \equiv a^{b \mod \varphi(m)} (\mod m)$
>
> 定理得证 □

若不保证 a 与 m 互质： $b > \varphi(m)$ 时： $a^b \equiv a^{b \mod \varphi(m) + \varphi(m)} \pmod m$

> 太长不证，证明详见： https://www.cnblogs.com/1024th/p/11349355.html

在一些计数的问题中，常常要求对结果取模，但是在计算非常庞大的次幂的时候，无法直接取模，可以先把底数对 p 取模，指数对 $\varphi(p)$ 取模，再计算次幂，有效地降低时间复杂度。

（模板）

计算 $a^b \bmod m$，$1 \leq a \leq 10^9$ ，$1 \leq b \leq 10^{20000000}$，$1 \leq m \leq 10^8$

Code

```cpp
int main()
{
    scanf("%d%d", &a, &m);
    bool flag = false;
    int phi_m = getphi(m);
    cin >> s;
    for(int i = 0; i < s.length(); ++ i) {
        b = (b * 10 + s[i] - '0');
        if(b ≥ phi_m)
            flag = 1, b %= phi_m;
    }
    if(flag)
        b += phi_m;
    int ans = qpow(a, b, m);
    printf("%d\n", ans);
    return 0;
}
```

竞赛例题选讲

Problem A 上帝与集合的正确用法（P4139 ）

$2^{2^{2^{\cdots}}} \bmod p$

Solution

根据拓展欧拉定理有：

$$2^{2^{2^{\cdots}}} \bmod p = 2^{((2^{2^{\cdots}}) \bmod \varphi(p))+\varphi(p)} \bmod p \tag{1}$$

然后递归求解即可。

Code

```cpp
int n, m, mod = 100, p, t;
int primes[N];
int cnt, phi[N];
bool vis[N];
void init(int n)
{
    phi[1] = 1;
    for(int i = 2; i ≤ n; ++ i) {
        if(vis[i] == 0) primes[ ++ cnt] = i, phi[i] = i - 1;
        for(int j = 1; j ≤ cnt && i * primes[j] ≤ n; ++ j) {
            vis[i * primes[j]] = true;
            if(i % primes[j] == 0) {
                phi[i * primes[j]] = phi[i] * primes[j];
                break;
            }
            phi[i * primes[j]] = phi[i] * (primes[j] - 1);
        }
    }
}

int qpow(int a, int b, int mod)
{
    int res = 1;

    while(b) {
        if(b & 1) res = 1ll * res * a % mod;
        a = 1ll * a * a % mod;
        b >>= 1;
    }
    return res;
```

```
}
int tower(int a, itn p)
{
    if(p == 1) return 0;
    return qpow(a, tower(a, phi[p]) + phi[p], p);
}

void solve()
{
    scanf("%d", &p);
    mod = p;
    int ans = tower(2, p);
    printf("%d\n", ans);
}
int main()
{
    init(N - 7);
    scanf("%d", &t);
    while(t -- ) {
        solve();
    }
    return 0;
}
```

0x21.4 威尔逊定理

威尔逊定理

定理21.4.1： 当 p 为质数时有：$(p-1)! \equiv p-1 \equiv -1(\mod p)$，$(p-2)! \equiv 1 \pmod p$

其中 **定理21.4.1** 实际上就等价于：若 p 是质数，则 $(p-1)!+1$ 能够被 p 整除。
n 为素数时：$(n-1)! \mod n = 1$
n 为合数时：除 $n=4$ 以外，$(n-1)! \mod n = 0$

威尔逊定理的逆命题

定理21.4.2： 若一个数 p，满足条件 $(p-1)!+1$可以被 p 整除，那么 p 是素数。

即：p **可整除** $(p-1)!+1$ **是** p **为质数的充要条件。**

Problem A Faulty Factorial（CERC2017 F）

给定三个数，n, p, r（$p > r$，r 是余数）,在 n 的阶乘 $1 \times 2 \times 3 \times 4 \times \cdots \times n$，这个连乘式中的 n 个数里，找到一个数 k（$2 \le k \le n$），将 k 换成比 k 小的数 v（$1 \le v < k$），使得换完之后的连乘式的结果 $res \equiv r \pmod p$。请你输出任意一组 k, v，若找不到答案则输出 `-1 -1`

Input

```
4 5 1
```

Output

```
3 2
```

样例解释：$n=4$，$n! = 1 \times 2 \times 3 \times 4$，将其中的 3 换成 2，则连乘式的结果变为 $1 \times 2 \times 2 \times 4 = 16 \equiv 1 \pmod 5$，故输出 `3 2`

Solution

看上去像是一个同余方程，要求输出一种方案 -> 拓展欧几里德解同余方程得到一组解 / 暴力枚举判断得到一组解...

因为 $n \le 10^{18}, p \le 10^7$， n 很大，经验告诉我们，当 n 超过一定的限制之后一定只有一种答案即一定是特解（~~盲猜特别大的时候无解~~），不然就真的没法玩了。

所以先从分析数据开始入手。显然可以发现：

若 $n \ge 2p$ 时， $n!$ 一定含有两个及以上的 p，而我们只能修改换掉一个 p，故 $n!$ 中一定含有 p，即 $n!$ 是 p 的倍数， 则 $n! \equiv 0 \pmod p$，也就是若 $r! = 0$ 就无解，若 $r = 0$，任意修改一个数即可。

有了这个分析之后，我们就有了一个分类讨论的解题思路。

显然剩余的情况还有两种：

若 $p \le n < 2p$，显然当 r 等于 0 的时，我们任意修改一个，只要不修改 p 即可。当 r 不等于 0 的时，我们必须修改 p，不然 $\mod p$ 一定等于 0。所以我们可以直接暴力枚举 p 修改成的数（$p \le 10^7$，$O(p)$ 的复杂度没问题），若膜 p 之后能和 r 匹配，则输出修改方案，否则输出 `-1 -1` 。

若 $n = p$ 我们可以使用威尔逊定理，但没必要，上面枚举判断就够了。

若 $n < p$，因为 $n \le p \le 10^7$，所以我们可以直接暴力枚举 k，即题目中要求的修改的点的下标显然其权值就是其下标。对于每一个位置，其修改的值 v，显然有

$$r \equiv \frac{n! \times v}{k} \pmod{p}$$

$$v \equiv \frac{r \times k}{n!}$$

因为有两个自变量，却只有一个同余方程，故不能通过拓展欧几里得算法求出特解，所以只能暴力，好在数据小，只有 10^7，我们是可以承受得了直接暴力枚举判断的。

那么暴力跑，我们可以直接预处理 $n!$ 的逆元，然后乘起来计算 v，然后判断当前的这个 v 是否满足题意即可。即判断当前得到的 v 合法，即若 $v < k$ 且 $v \ge 1$ 则合法，直接输出即可。

最后注意多个数相乘的时候记得多取模数，不然你会因爆 `long long` 而 wa 71...

Code

```
ll n, p, r;
ll inv[N];

void init()
{
    inv[1] = 1;
    for(ll i = 2; i ≤ p; ++ i) {
        inv[i] = (p - (p / i)) * inv[p % i] % p;
    }
}

void solve()
{

    scanf("%lld%lld%lld", &n, &p, &r);
    init();
    if(n ≥ 2 * p) {
        if(r == 0) {
            printf("2 1\n");
        }
        else puts("-1 -1");
    }
    else if(n ≥ p) {
        if(r == 0) {
            bool ok = 0;
            //if(p == 2)
            for(ll i = 2; i ≤ n; ++ i) {
                if(i ≠ p) {
                    printf("%lld 1", i);
                    return ;
                }
            }
            puts("-1 -1");
            return ;
        }
        else {
            ll fact = 1;
            for(int i = 1; i ≤ n; ++ i) {
                if(i == p) continue;
                fact = (fact * i) % p;
            }
            for(ll i = 1; i < p; ++ i) {
                if((fact * i) % p == r) {
                    printf("%lld %lld", p, i);
                    return ;
                }
            }
            puts("-1 -1");
            return ;
        }
    }
    else if(n < p) {
```

```
        ll fact_inv = 1;
        for(int i = 1; i ≤ n; ++ i) {
            fact_inv = (fact_inv * inv[i]) % p;
        }
        for(ll k = 1; k ≤ n; ++ k) {
            ll v = r * k % p * fact_inv % p;
            if(v ≥ 1 && v < k) {
                printf("%lld %lld", k, v);
                return ;
            }
        }
        puts("-1 -1");
    }
    return ;
}

itn main()
{
    solve();
}
```

Problem B. Fansblog（HDU 6608 19多校）

给定一个质数 $P(10^9 \leq P \leq 10^{14})$， Q是 最大的那个小于 P 的质数，求 $Q!\%P$。

Solution

根据威尔逊定理 $(P-1)! \equiv P-1 \equiv -1(\mod P)$，所以显然可以构造答案 $Q!$

$Q! = \frac{(P-1)!}{(P-1)\times(P-2)\times\cdots\times(P-Q)}$

上面就是 -1 或者 $P-1$ ，因为题目中求的是正整数，所以取 $P-1$即可。下面直接求逆元即可。

由素数分布定理得，两个素数之间的距离最大不超过 300 ，所以直接暴力找上一个素数即可。因为数据范围 $P \leq 10^{14}$ ，我们每次暴力判断素数需要 $O(\sqrt{n})$ ，我们可以先预处理小于 $\sqrt{10^{14}} = 10^7$ 的质数，只有 $6e5$ 个，这样我们可以直接用质数试除法判断即可。并且因为数据较大，所以求逆元乘的时候需要用到快速乘。

注意如果传参数多（多传一个 `mod` ）的话会 T...改了就A了。不然就只能用 Miller-Rabin 判定法了。

Code

```
ll mod;
int n, m;
ll p, cnt;
ll q;
bool vis[N];
int primes[N];

void init(itn n)
{
    for(int i = 2;i ≤ n; ++ i) {
        if(vis[i] == 0) primes[ ++ cnt] = i;
        for(int j = 1; j ≤ cnt && i * primes[j] ≤ n; ++ j) {
            vis[i * primes[j]] = true;
            if(i % primes[j] == 0) break;
        }
    }
}

bool is_primes(ll x)
{
    for(int i = 1;i ≤ cnt && 1ll * primes[i] * primes[i] ≤ x; ++ i) {
        if(x % primes[i] == 0) return false;
    }
    return true;
}

ll mul(ll a, ll b)
{
    if(b < 0) a = - a, b = - b;
    ll res = 0;
    while(b) {
        if(b & 1) res = (res + a) % mod;
```

```
        a = (a + a) % mod;
        b >>= 1;
    }
    return res;
}

ll qpow(ll a, ll b)
{
    ll res = 1;
    while(b) {
        if(b & 1) res = mul(res, a);
        a = mul(a, a);
        b >>= 1;
    }
    return res;
}

ll inv(ll x)
{
    return qpow(x, mod - 2);
}

void solve()
{
    scanf("%lld", &p);
    mod = p;
    q = p - 1;
    ll ans = p - 1;
    while(is_primes(q) == false)
        q -- ;
    q ++ ;
    while(q < p)
        ans = mul(ans, inv(q)), q ++ ;
    printf("%lld\n", ans);
    return ;
}

itn main()
{
    itn t;
    init(N - 7);
    scanf("%d", &t);
    while(t -- ) {
        solve();
    }
}
```

0x22 拓展欧几里德

0x22.1 裴蜀（Bézout）定理

定理22.1.1：设 a,b 是不全为零的整数，存在无穷多组整数对 (x,y)，满足不定方程 $ax+by=d$，其中 $d=\gcd(a,b)$ 即：$ax+by=\gcd(a,b)$。

推论22.1.2：$\gcd(a,b)\mid c\Longleftrightarrow \exists x,y\in Z,ax+by=c$

> 方程 $ax+by=d(d=\gcd(a,b))$ 即为丢番图方程。

推论21.1.3：$\forall a,b,z\in \mathbb{N}^*,\gcd(a,b)=1,\exists x,y\in \mathbb{N},ax+by=ab-a-b+z$，即两互质的数 a,b，表示不出的最大的数为$ab-a-b$。

> **推论21.1.3证明**：不妨设 $a<b$，假设答案为 x。
> 若$x\equiv ma\pmod b(1\le m\le b-1)$（若$m\ge b$，$m\equiv 0\pmod b$）
> 即$x=ma+nb(1\le m\le b-1)$
> 显然当 $n\ge 0$ 时 x 可以用 a,b 表示出来，不合题意。
> 因此当 $n=-1$ 时 x 取得最大值，此时$x=ma-b$。
> 显然当 m 取得最大值 $b-1$ 时 x 最大，此时 $x=(b-1)a-b=ab-a-b$
> 即 a,b 所表示不出的最大的数是 $ab-a-b$

- **竞赛例题选讲**

Problem A Fox And Jumping（CF510D）

给出 n 张卡片，分别有 l_i 和 c_i 。在一条无限长的纸带上，你可以选择花 c_i 的钱来购买卡片 i，从此以后可以向左或向右跳 l_i 个单位。问你至少花多少元钱才能够跳到纸带上全部位置（整数数轴）。若不行，输出 -1。$1 \leq n \leq 300, 1 \leq l_i, c_i \leq 10^9$。

Solution （思路较长，比较详细，建议看完）

首先分析子问题，先考虑两个数的情况，因为纸带是无限长的，没有循环，我们发现，要想能够跳到每一个格子上，就必须使得我们选择的数通过数次加减得到的数的绝对值为 1，进而想到了裴蜀定理。

我们知道裴蜀定理的内容是：**设 a, b 是不全为零的整数，则存在整数x, y，使得$ax + by = \gcd(a, b)$。** 我们想要解这个二元一次方程得到答案 1，也就是使得 $\gcd(a, b) = 1$，我们可以推出：如果 a 与 b 互质，则一定存在两个整数 x, y ，使得$ax + by = 1$。

由此我们想到本题的解题思路：选择最便宜的两个或者多个**互质**的数（多个数互质是指$\gcd(\gcd(\gcd(a, b)c\ldots) = 1$）。考虑动态规划。但是我们发现数据的大小达到了$10^9$，我们的动态规划开不了这么大的数组，并且时间复杂度上也会很糟糕（尽管动规实际上是可以通过本题的hhh，如果可以把数据范围中的 $n = 300$ 上调至 500000 ，可以卡掉除本思路以外的所有做法，那么这道题我将绝杀，可惜调不得 ~ ）。那么考虑有没有其他的做法。

我们知道动态规划问题实际上就是 `DAG` 上的递推，动态规划对于状态空间的遍历构成了一张有向无环图 `DAG` ，遍历顺序就是该 `DAG` 的一个拓扑序。考虑用图论的做法解决这个问题。因为我们想要找到的是最小代价的两个互质的数，从 0 开始出发，选取最小的代价，最后的终点是 $\gcd(a, b) = 1$ 或者是 $\gcd(\gcd(\gcd(a, b)c\ldots) = 1$ ，我们发现有一点最短路的感觉。我们发现多个数互质，实际上就是一步步的 $\gcd$ 最后推到 1，就类似我们最短路的一步一步的结点转移。最后考虑从 0 出发会不会对我们求解 $\gcd$ 造成影响呢？由于我们知道 $\gcd(0, x) = x$ ，所以不会有任何的影响。

至此整体的思路已经非常清晰了：我们从 0 号节点开始，每一步走到的结点编号为 $\gcd(x, y)$ ，其中 x 为当前的结点编号，也就是当前已选择的数的总 $\gcd$，y 为我们枚举到的下一个结点 ，也就是下一个待选的数的值 $l[i]$，与此同时更新代价，利用 `dijkstra` 算法求得最小的总代价。

最后如果能够到达结点 1，也就意味着我们能够找到若干个数使得他们的总$\gcd$ 为 1 ，输出 `dist[1]` 即可。反之说明无法得到，即无法遍历所有的格点。

最后的一点小细节：由于我们的最短路过程中，需要使用一个 `vis` 数组记录每一个结点是否已经被遍历过了，由于数据过大，开不下这么大的数组，但是数据的数量不大，所以我们可以手写一个 `hash` 或者使用 `STL` 里自带的 `hash` 表 `unordered_map` 快速地（$O(1)$）访问每一个元素。

时间复杂度 $O(n \log n)$

Code

```cpp
const int N = 50007, M = 500007, INF = 0x3f3f3f3f;
typedef long long ll;
typedef pair<int, int> PII;
unordered_map<int, bool> vis;
unordered_map<int, ll> dist;

int n, m;
int head[N], ver[N], nex[N], edge[M], tot;
int a[N], l[N], c[N];

int gcd(int a, int b)
{
    if(b == 0) return a;
    return gcd(b, a % b);
}
//gcd(0,x) = x;
void dijkstra()
{
    priority_queue<PII, vector<PII>, greater<PII> > q;
    q.push({0, 0});
    dist[0] = 0;

    while(q.size()) {
        int x = q.top().second;
        q.pop();
        if(x == 1) break;

        if(vis.find(x) != vis.end()) continue;
        vis[x] = true;
        for(int i = 1; i <= n; ++ i) {
            int y = __gcd(x, l[i]), z = c[i];
            if(dist.find(y) == dist.end()) dist[y] = INF;
            if(dist[y] > dist[x] + z) {
                dist[y] = dist[x] + z;
                q.push({dist[y], y});
            }
        }
    }
}

int main()
{
```

```
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i)
        scanf("%d", &l[i]);

    for(int i = 1; i ≤ n; ++ i)
        scanf("%d", &c[i]);
    dijkstra();

    if(dist.find(1) == dist.end()) puts("-1");
    else printf("%lld\n", dist[1]);
    return 0;
}
```

0x22.2 扩展欧几里德算法

我们可以利用欧几里得算法求解 $ax+by=\gcd(a,b)$ 中的 x 和 y 。

我们知道欧几里得算法利用的核心性质为 $\gcd(a,b)=\gcd(b,a \bmod b)=\gcd(b,a-b\lfloor\frac{a}{b}\rfloor)$

根据裴蜀定理，我们一定可以找到四个整数 x,y,x',y'，使得 $ax+by=\gcd(a,b)$ 且 $bx'+(a-b\lfloor\frac{a}{b}\rfloor)y'=\gcd(b,a-b\lfloor\frac{a}{b}\rfloor)$

由于 $\gcd(a,b)=\gcd(b,a \bmod b)=\gcd(b,a-b\lfloor\frac{a}{b}\rfloor)$

有:

$$
\begin{aligned}
ax+by&=bx'+(a-b\lfloor\frac{a}{b}\rfloor)y' \\
a(x-y')+b(y-(x'-\lfloor\frac{a}{b}\rfloor y'))&=0
\end{aligned}
\tag{2}
$$

我们希望这个等式对一切 a，b 都成立，于是 $x=y'$ 且 $y=x'-\lfloor\frac{a}{b}\rfloor y'$ 。

显然，我们想求 x 和 y，只需要求出 x'，y'，由于 x'，y' 对应的问题相同且规模更小，所以可以进行递归的运算。边界条件为：当 $b=0$ 时，$x=1,y=0$， $a\times 1+0\times 0=\gcd(a,0)=a$ 。

由于算法思想与欧几里得相同，我们称之为拓展欧几里得算法。

实现：

在用欧几里德算法求 $d=\gcd(a,b)$ 的过程中求方程 $ax+by=d$ 的一组整数解 (x,y)

若 $d\mid c$,不妨设 $c=kd$，则有$a(kx)+b(ky)=c$，否则原方程无整数解。

```
//在gcd的过程上增加了拓展
inline int exgcd(int a, int b, int &x, int &y)
{
    if(b == 0){
        x = 1, y = 0, return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;x = y, y = z - y * (a / b);
    return d;
}
```

`exgcd` 可得到 $ax+by=\gcd(a,b)$ 的解，对于 $ax+by=c$ 的解，我们只需要根据 **定理22.3.3** 构造即可。

0x22.3 解二元模线性方程

二元模线性方程（二元一次不定方程）：形如 $ax\equiv c\pmod b$ 或 $ax+by=c$ 。其中 a,b,c,x,y 均为整数。

定理22.3.1：上述方程有解的充要条件是 $\gcd(a,b)\mid c$

可以理解为 $\gcd(a,b)$ 是 $ax+by$ 可以表示出来的最小的正整数。

定理22.3.2：方程$ax+by=d,d=\gcd(a,b)$的所有解为：

$$
\begin{cases}
x=x_0+k\frac{b}{d} \\
y=y_0-k\frac{a}{d}
\end{cases}
$$

其中 x_0，y_0 是一组特解，$k\in\mathbb{Z}$ 。

> **证明：**
> 方程的特解 (x_0, y_0)，任取另一组解 (x, y)，则 $ax_0 + by_0 = ax + by = \gcd(a, b)$。变形得 $a(x_0 - x) = b(y - y_0)$ 。设 $\gcd(a, b) = d$，方程左右两边同时除以 d（如果 $d = 0$，说明 a 或 b 等于 0），得 $a'(x - x_0) = b'(y_0 - y)$，其中 $a' = \frac{a}{d}$，$b' = \frac{b}{d}$ 。显然此时 a' 和 b' 互质，因此 $x - x_0$ 一定是 b' 的整数倍（因为 a' 中不包含 b'，所以 $x_0 - x$ 一定包含 b'）。设它为 kb'，即 $x - x_0 = k\frac{b}{d}$，同理，有 $y_0 - y = k\frac{a}{d}$ 。

定理22.3.3：方程 $ax + by = c, \gcd(a, b) \mid c$ 的所有解为

$$\begin{cases} x = \frac{c}{d}\ x_0 + k\frac{b}{d} \\ y = \frac{c}{d}\ y_0 - k\frac{a}{d} \end{cases}$$

其中 x_0，y_0 是方程$ax + by = d, d = \gcd(a, b)$ 的一组特解，$k \in \mathbb{Z}$。

Z 是整数集，也就意味着k 可以为负数，即最小正整数解：

若 $x > 0 \to x \mod \frac{b}{d}$

- **竞赛例题选讲**

Problem A　A + B（UVALive6428）

给出 a, b, S 三个数，两个格子，第一个格子上的权值是 a，第二个格子上的权值是 b，每次可以使得第一个格子上的权值变成原权值加上第二个格子上的权值，也可以，问a使得第二个格子上的权值变成原权值加上第一个格子上的权值。问两个格子上的权值能否等于 S。

Solution

手动模拟一下：

```
                (a      ,      b)
          (a+b , b)            (a ,  a+b)
    (a+2b,b)  (a+b,a+2b) (2a+b,a+b)  (a,2a+b)
```

可以发现实际上就是问有没有两个**非负整数**解 x, y，满足 $ax + by = S$ 且 $\gcd(x, y) = 1$。

直接用拓展欧几里德算法求解系，看解系中是否存在满足题意的 x, y 。注意一些细节上的特判即可。

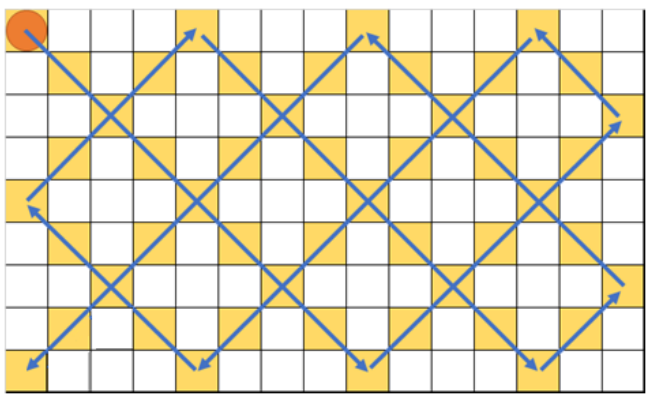
Code

```
int n, m, t;
int a, b, s, kcase;
bool ans;

int exgcd(int a, int b, int &x, int &y)
{
    if(b == 0) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;
    x = y;
    y = z - y * (a / b);
    return d;
}

void solve()
{
    if(a == 0 && b == 0) {
        ans = (s == 0);
        return ;
    }
    if(a == 0) {
        ans = (s % b == 0);
        return ;
    }
    if(b == 0) {
        ans = (s % a == 0);
        return ;
    }
    itn x, y;
    int d = exgcd(a, b, x, y);
    if(s % d =/= 0) {
        ans = false;
```

```
        return ;
    }
    ll x0 = b / d;
    ll y0 = a / d;
    x = ((s / d % x0) * (x % x0) % x0 + x0) % x0;
    y = (s - x * a) / b;
    ans = false;
    while(y > 0) {
        if(__gcd(x, y) == 1) {
            ans = true;
            return ;
        }
        else {
            x += x0;
            y -= y0;
        }
    }
}

signed main()
{
    while(scanf("%lld%lld%lld", &a, &b, &s) != EOF) {
        ans = false;
        solve();
        if(ans)
            puts("YES");
        else puts("NO");
    }
    return 0;
}
```

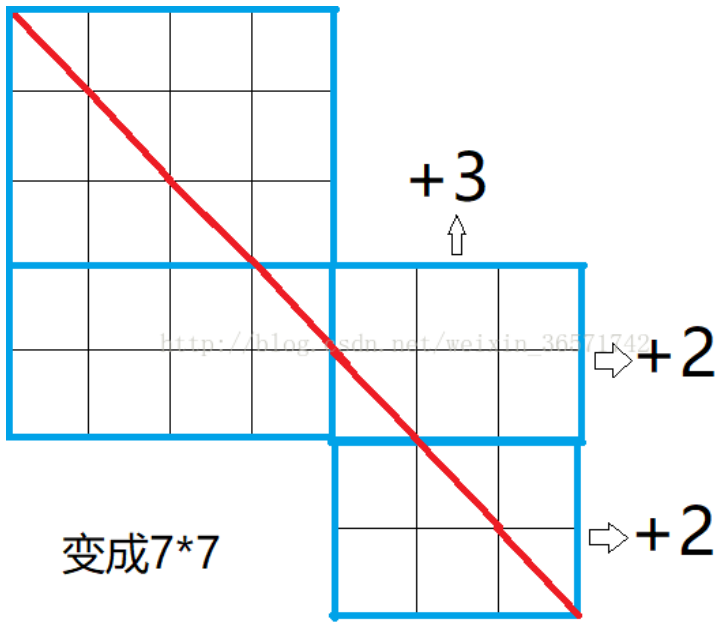
Problem B Bounce（hihocoder1584 2017ICPC beijing网络赛）

在 $n \times m$ 的网格上，一颗小球左上方出发往右下的格子走，碰到边界就反弹，当小球到达角落就停止运动。求小球只经过一次的格子的数量。



Solution

对于这种反弹的问题，我们可以把每次反弹后的线路展开：



每次反弹就增加 $m-1$ （碰到上下边界）或者 $n-1$ （碰到左右边界）。因为小球是沿着 45 度方向移动，所以最后展开之后走的整个图形一定是一个正方形。于是可以列出不定方程 $(m-1) \times x + m = (n-1) \times y + n$。其中 x 和 y 可以通过扩展欧几里德求出最小正数解。根据图像可以看出 $(m-1) \times x + m$ 就是小球经过的格子总数（重复的也算）。

最后减去重复经过两次的格子即可。之前求得的方程的解中 x 就是左右反弹次数之和，y 就是上下反弹次数之和。那么相交的就是经过两次的次数，要减去。显然每次反弹的时候，向上下反弹一次，就一定会跟着向左右反弹一次。每两个相反的路线都会相交一次。或者分析其实际的意义，对于一个上下碰撞，会产生一个‘V‘字形，而左右碰撞会产生一个’ >’形 。每个’V’ 和 ‘>’必会产生一个有且只有一个交点，故根据乘法原理，一共有 $x \times y$ 个格子重复两次经过。即答案就是 $(m-1) \times x + m - x \times y$。

Code

```
int n, m, t;
int a, b, s, kcase;
int ans;

int exgcd(int a, int b, int &x, int &y)
{
    if(b == 0) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;
    x = y;
    y = z - y * (a / b);
    return d;
}

void solve()
{
    int x, y;
    int a = m - 1, b = 1 - n, c = n - m;
    int d = exgcd(a, b, x, y);
    int x0 = abs(b / d);
    int y0 = abs(a / d);
    x *= c / d;
    if(x < 0) {
        x = x + (-x / x0 + 1) * x0;
    }
    else x %= x0;
    y = (c - a * x) / b;
    printf("%lld\n", (m - 1) * x + m - x * y);
}

signed main()
{
    while( ~ scanf("%lld%lld", &n, &m)) {
        solve();
    }
    return 0;
}
```

Problem C Integer Sequences（SGU 140）

给出一个长度为 n 的非负整数序列 A 和两个数 P, B，要求找出同样的非负整数序列 X 满足 $A_1 * X_1 + A_2 * X_2 + \ldots + A_n * X_n = B \pmod P$

Solution

$A_1 * X_1 + A_2 * X_2 + \ldots + A_n * X_n = B \pmod P$

显然有
$A_1 * X_1 + A_2 * X_2 + \ldots + A_n * X_n + PQ = B, Q \in \mathbb{Z}$

看起来是一个多元一次方程，我们只需要找到一个合法的非负整数序列 X 作为解即可，因此我们可以构造答案。我们可以求出二元一次方程的解，因此我们可以从前往后，两个数就可以找到一组解，这样两两合并即可得到一组合法的解。

即：先考虑 $A_1X_1 + A_2X_2$，我们可以求出方程 $A_1x + A_2y = \gcd(A_1, A_2)$ 的解 x, y，此时 x 就是满足当前条件的 X 序列的第一项的解，$X_1 = x, X_2 = y$。我们把 $\gcd(A_1, A_2)$ 当作新的元素，于是得到新的方程：

$\gcd(A_1, A_2)x + A_3 * X_3 + \ldots + A_n * X_n + PQ = B, Q \in \mathbb{Z}$

我们再合并 $gcd(A1, A2)$ 和 $A3$ ，解出 $\gcd(A_1, A_2)x + A_3y = \gcd(\gcd(A_1, A_2), A_3)$ 的 x, y，把之前求出的所有的 X_i乘上 x（因为 $A_1 * X_1 + A_2 * X_2 = \gcd(A_1, A_2)$，gcd 的系数也要乘到前面的所有项上），$X_3 = y$ ，不断重复，直到合并只剩两项为止。

最后求解 $\gcd(A_1, A_2, A_3 \ldots A_n)x + Py$ 的 x, y，判断 $\gcd(A_1, A_2, A_3 \ldots A_n)$ 能否整除 B ，若不能整除，显然该丢番图方程无解，输出 NO 即可。

若能整除，所有的解乘上 $\frac{B}{\gcd(A_1, A_2 \ldots A_n, P)}$ 即为一组合法的解，输出即可。

注意我们在求解 x, y 的过程中可能得到负数解，而题目要求输出整数解，最后输出的时候将 X_i 置于 $[0, P]$ 之间即可。

Code

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 100 + 7;
int n, m, s, t, k, ans;
int X[maxn];
int A[maxn], p, b;

int exgcd(int a, int b, int &x, int &y)
{
    if(b == 0) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;
    x = y, y = z - y * (a / b);
    return d;
}

int main()
{
    scanf("%d%d%d", &n, &p, &b);
    for (int i = 1; i ≤ n; ++ i)
        scanf("%d", &A[i]), A[i] %= p;
    int gcd = A[1];
    X[1] = 1;
    for (int i = 2; i ≤ n; ++ i) {
        int x, y;
        gcd = exgcd(gcd, A[i], x, y);
        for (int j = 1; j < i; ++ j)
            X[j] = X[j] * x % p;
        X[i] = y;
    }
    int x, y;
    gcd = exgcd(gcd, p, x, y);
    for (int i = 1; i ≤ n; ++ i)
        X[i] = X[i] * x % p;
    if(b % gcd ≠ 0) {
        puts("NO");
        return 0;
    }
    else {
        puts("YES");
        for (int i = 1; i ≤ n; ++ i) {
            X[i] = X[i] * b / gcd % p;
            printf("%d ", (X[i] + p) % p);
        }
    }
    puts("");
    return 0;
}
```

0x22.4 类欧几里德算法（一个求和技巧）

- **竞赛例题选讲**

Problem A 类欧几里德算法 1

求$\sum_{i=0}^{n}[\frac{ai+b}{c}]$

其中$[\ ]$表示取整。

Solution

```cpp
ll sum_pow(ll n, ll k) {
    if (k == 0) return n;
    else if (k == 1) return n * (n + 1) / 2;
    else if (k == 2) return n * (n + 1) * (2 * n + 1) / 6;
```

```
    else if (k == 3) return n * n * (n + 1) * (n + 1) / 4;
    else if (k == 4) return n * (2 * n + 1) * (n + 1) * (3 * n * n + 3 * n - 1) / 30;
    else assert(false);
}
ll EuclidLike1(ll a, ll b, ll c, ll n) {
    if (a == 0) return b / c * (n + 1);
    else if (a ≥ c || b ≥ c)
        return (a / c) * sum_pow(n, 1) + (b / c) * (n + 1) + EuclidLike1(a % c, b % c, c, n);
    else
        return (a * n + b) / c * n - EuclidLike1(c, c - b - 1, a, (a * n + b) / c - 1);
}
```

Problem B 类欧几里德算法 2

给定 n, a, b, c，求
$\sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor, \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2, \sum_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor$

Solution

```
#define Int register int
#define mod 998244353ll
#define int long long
int inv2 = 499122177ll,inv6 = 166374059ll;
struct Ans{int f,g,h;};
Ans Solve (int a,int b,int c,int n)
{
    if (!a) {
        int f = (n + 1) * (b / c) % mod;
        int g = (n + 1) * (b / c) % mod * (b / c) % mod;
        int h = n * (n + 1) % mod * inv2 % mod * (b / c) % mod;
        return Ans {f % mod,g % mod,h % mod};
    }
    else if (a ≥ c || b ≥ c) {
        Ans fucker = Solve (a % c,b % c,c,n);
        int F = fucker.f + n * (n + 1) / 2 % mod * (a / c) % mod + (n + 1) * (b / c) % mod;
        int G = fucker.g + 2 * (a / c) % mod * fucker.h % mod + 2 * (b / c) % mod * fucker.f +
        n % mod * (n + 1) % mod * (2 * n % mod + 1) % mod * inv6 % mod * (a / c) % mod * (a / c) % mod +
        n % mod * (n + 1) % mod * (a / c) % mod * (b / c) % mod + (n + 1) * (b / c) % mod * (b / c) % mod;
        int H = fucker.h + n % mod * (n + 1) % mod * (2 * n + 1) % mod * inv6 % mod * (a / c) % mod + n % mod * (n + 1) % mod *
inv2 % mod * (b / c) % mod;
        return Ans {F % mod,G % mod,H % mod};
    }
    else {
        int M = (a * n + b) / c;
        Ans fucker = Solve (c,c - b - 1,a,M - 1);
        int F = n * M % mod - fucker.f;
        int G = n * M % mod * (M + 1) % mod - 2 * fucker.h % mod + mod - 2 * fucker.f % mod + mod - F % mod;
        int H = (M * n % mod * (n + 1) % mod - fucker.g + mod - fucker.f) % mod * inv2 % mod;
        return Ans {F % mod,G % mod,H % mod};
    }
}
int read ()
{
    int x = 0;char c = getchar();int f = 1;
    while (c < '0' || c > '9'){if (c == '-') f = -f;c = getchar();}
    while (c ≥ '0' && c ≤ '9'){x = (x << 3) + (x << 1) + c - '0';c = getchar();}
    return x * f;
}
void write (int x)
{
    if (x < 0){x = -x;putchar ('-');}
    if (x > 9) write (x / 10);
    putchar (x % 10 + '0');
}
signed main()
{
    int times = read ();
    while (times --)
    {
        int n = read (),a = read (),b = read (),c = read ();
```

```
        Ans Putout = Solve (a,b,c,n);
        write ((Putout.f + mod) % mod),putchar (' '),write ((Putout.g + mod) % mod),putchar (' '),write ((Putout.h + mod) % mod),putchar ('\n');
    }
    return 0;
}
```

0x22.5 整式方程

整式方程就是方程中所有的未知数均在分子上，分母只是常数且无未知数。

通常情况下，常年用字母 x, y, z 来表示未知数，方程中含有几个不同的未知数就叫做几元，未知数的最高次数是几就叫做几次。

例如：$ax + b = c$ 就是一个一元一次整式方程

- **一元一次整式方程**

对于方程 $ax + b = c$，有：$x = \frac{c-b}{a}$

```
double calculate(double a, double b, double c){
    return (c - b) / a;
}
```

- **一元二次整式方程**

```
double x1, x2;
bool calculate(double a, double b, double c){
    double delta = b * b - 4 * a * c;
    if(delta < 0) {
        return false;
    }
    else if(delta == 0) {
        x1 = ( - 1 * b + sqrt(delta) / (2 * a) );
        x2 = x1;
    }
    else {
        x1 = ( - 1 * b + sqrt(delta) / (2 * a) );
        x2 = ( - 1 * b - sqrt(delta) / (2 * a) );
    }
    return true;
}
```

0x23 乘法逆元

0x23.1 乘法逆元定义与性质

定义23.1.1：若$a \times x \equiv 1 \pmod b$，且 a 与 b 互质，那么我们就能定义: x 为 a 的逆元，记为a^{-1} ，所以我们也可以称 x 为 a 在 $\bmod\ b$ 意义下的倒数。

由于除法不能直接取模，但是可以用乘法逆元

对于 $\frac{a}{b} \bmod p$，我们就可以求出 b 在 $\bmod\ p$ 下的逆元，然后乘上 a，再 $\bmod\ p$，即可得到该分数的值。

当然满足条件的逆元不止一个，通常情况下我们只用最小正整数的逆元，并且逆元也不在任何条件下都有逆元

有了乘法逆元，我们在求计数类问题中遇到 $\frac{a}{b}$ 的除法算式的时候，可以先把 a，b 各自对模数 p 取模，然后再计算 $a \times b^{-1} \pmod p$ 作为最终的结果。（前提是保证 b， p 互质，当 p 是质数的时候等价于 b 不是 p 的倍数）

若 b, q 不互质，显然 $bx \equiv 1 \pmod p$ 无解，即不存在 b 模 p 的乘法逆元。

0x23.2 费马小定理求乘法逆元

保证 b， m 互质且 m 是质数。

费马小定理 $a^{p-1} \equiv 1 \mod p$,其中 p 为素数。

根据费马小定理变形得 $a \times a^{p-2} \equiv 1 \pmod p$ 。

则 $a^{p-2} \mod p$ 就是逆元，直接快速幂求得：

```
int inv(int x,int p) {return qpow(x, mod - 2, p) % p;}
```

0x23.3 扩展欧几里得求乘法逆元

若题目仅保证 b，m 互质，根据逆元的定义 $ax \equiv 1 \pmod m$ 可得丢番图方程组： $ax + my = 1$。

解方程得 $x \bmod m$ 得到的就是 a 的乘法逆元，同时可得逆元存在的条件 $\gcd(a, m) = 1$。

```
inline int invers(int a,int mod) {
    register int x,y;
    exgcd(a,mod,x,y);
    return (x % mod + mod) % mod;
}
```

0x23.4 线性递推求乘法逆元

> 给定 n, p 求 $1 \sim n$ 中**所有的整数**在模 p 意义下的乘法逆元。$1 \leq n \leq 3 \times 10^6, n < p < 20000528$，输入保证 p 是质数。

数据较大，我们需要一个复杂度为 $O(n)$ 的算法：线性递推。

首先，很显然的 $1^{-1} \equiv 1 \pmod p$

> 对于 $\forall p \in \mathbb{Z}$，有 $1 \times 1 \equiv 1 \pmod p$ 恒成立，故在 p 下 1 的逆元是 1，而这是推算出其他情况的基础。

其次对于递归情况 i^{-1}，我们令 $k = \left\lfloor \frac{p}{i} \right\rfloor$，$j = p \bmod i$，有 $p = ki + j$。再放到 $\bmod p$ 意义下就会得到：$ki + j \equiv 0 \pmod p$

两边同时乘 $i^{-1} \times j^{-1}$：

$$\begin{aligned} kj^{-1} + i^{-1} &\equiv 0 \pmod p \\ i^{-1} &\equiv -kj^{-1} \pmod p \end{aligned} \tag{3}$$

再将 $k = \left\lfloor \frac{p}{i} \right\rfloor$，$j = p \bmod i$ 带入有：

$$i^{-1} \equiv -\lfloor \frac{p}{i} \rfloor (p \bmod i)^{-1} \pmod p \tag{4}$$

我们注意到 $p \bmod i < i$，显然我们在循环计算的时候，在计算 i 的时候，我们已经得到了所有的 $j < i$ 的模 p 下的逆元 j^{-1}。

即：

$$i^{-1} \equiv \begin{cases} 1, & \text{if } i = 1, \\ -\lfloor \frac{p}{i} \rfloor (p \bmod i)^{-1}, & \text{otherwises.} \end{cases} \pmod p \tag{5}$$

```
inv[1] = 1;
for (int i = 2; i ≤ n; ++i) {
    inv[i] = (long long)(p - p / i) * inv[p % i] % p;
}
```

其中我们使用 $p - \left\lfloor \frac{p}{i} \right\rfloor$ 来防止出现负数。

显然时间复杂度为 $O(n)$。

另外我们注意到我们没有对 `inv[0]` 进行定义却可能会使用它：当 $i \mid p$ 成立时，我们在代码中会访问 `inv[p % i]`，也就是 `inv[0]`，这是因为当 $i \mid p$ 时不存在 i 的逆元 i^{-1}。显然如果 i 与 p 不互素时不存在相应的逆元（当一般而言我们会使用一个大素数，比如 $10^9 + 7$ 来确保它有着有效的逆元）。因此需要指出的是：如果没有相应的逆元的时候，`inv[i]` 的值是未定义的。

值得一提的是，我们使用上述递归式求解单个数的逆元，理论时间复杂度的上界为 $O(n^{\frac{1}{3}})$，高于拓展欧几里得的 $O(\log n)$。（详见 知乎回答 ）

```
int n, m;
int inv[N], p;
int main()
{
    scanf("%d%d", &n, &p);
    inv[1] = 1;
    puts("1");
    for(int i = 2; i ≤ n; ++ i) {
        inv[i] = (ll)(p - p / i) * inv[p % i] % p;
        printf("%d\n", inv[i]);
    }
    return 0;
}
```

0x23.5 求阶乘的逆元

定义 $\text{inv}[i]$ 为 $i!$ 的逆元

我们知道 $\text{inv}[i+1] = \frac{1}{i+1!}$

两边同时乘上 $i+1$ 得 $\text{inv}[i+1] \times (i+1) = \frac{1}{i!} = \text{inv}[i]$。

我们只需要先求出 $\text{inv}[n]$然后往回递推即可。

当然我们也可以先 $O(n)$ 求出 $1 \sim n$ 的所有数的逆元，然后求阶乘即可。

0x24 线性同余方程

0x24.1 同余方程

形如 $ax \equiv b \pmod{m}$ 的方程称为同余方程。

根据同余的定义，同余方程等价于 $ax + mt = b(t \in \mathbb{Z})$ ，可以用扩展欧几里得求解

$$ax \equiv b \pmod{m} \Rightarrow ax \bmod m = b \bmod m \Rightarrow ax = b + mt \Rightarrow ax + mt = b(t \in \mathbb{Z})$$

显然同余方程有解的条件是 $\gcd(a, m) \mid b$。

这个方程就是二维空间中的直线方程，但是由于我们的 x 和 y 的取值均为整数，所以这个方程的解是一些排列成直线的点集。

- **竞赛例题选讲**

Problem A 同余方程（NOIP 2012）

输入 a, b，求关于 x 的同余方程 $ax \equiv 1 \pmod{b}$ 的**最小正整数解**。 $(2 \leq a, b \leq 2 \times 10^9)$

Solution

```cpp
inline ll exgcd(ll a, ll b, ll& x, ll& y) {
    if(b == 0){x = 1, y = 0; return a;}
    ll d = exgcd(b , a % b, x, y);
    ll z = x; x = y, y = z - y * (a / b);
    return d;
}
ll x, y, a, b;
int main()
{
    scanf("%lld%lld", &a, &b);
    exgcd(a, b, x, y);
    printf("%lld\n", (x % b + b) % b);
    //通过取模压缩到1~b之间即是最小正整数解
    return 0;
}
```

Problem B 青蛙的约会（AcWing 222）

两只青蛙在网上相识了，它们决定见上一面。我们把这两只青蛙分别叫做青蛙 A 和青蛙 B，它们在一条单位长度 1 米首尾相接的数轴。设青蛙 A 的出发点坐标是 x，青蛙 B 的出发点坐标是 y 。青蛙 A 一次能跳 m 米，青蛙 B 一次能跳 n 米，两只青蛙跳一次所花费的时间相同。纬度线总长 L 米。现在要你求出它们跳了几次以后才会碰面。

Solution

A, B 不在同一位置，设两只青蛙A, B一共跳了 t 次后碰面

那么 A 要追 B $(y - x)$ 米（可能为负）

每跳一次 A 追 B $(m - n)$ 米，一圈 L 米

$mt + x \equiv nt + y \pmod{L}$

$(m - n)t \equiv y - x \pmod{L}$

设 $a = m - n$，$c = y - x$

得：$at + Ly = c$

我们直接使用扩展欧几里得求解即可。

Code

```cpp
ll n, m, x, y;
ll a, b, L;
ll exgcd(ll a, ll b, ll &x, ll &y){
    if(b == 0){x = 1;y = 0;return a;}
    ll d = exgcd(b, a % b, x, y);
    ll z = x;x = y;y = z - (a / b) * x;
    return d;
}
int main()
```

```cpp
{
    scanf("%lld%lld%lld%lld%lld", &a, &b, &m, &n, &L);
    ll d = exgcd(m - n, L, x, y);
    if((b - a) % d != 0){
        puts("Impossible");
    }
    else {
        x *= (b - a) / d;
        ll t = abs(L / d);
        printf("%lld\n", (x % t + t) % t);
    }
    return 0;
}
```

Problem B. Rise of Shadows (2020 ICPC shenyang I)

给定一个时钟，时针 H 小时转一圈，分针 M 分钟转一圈，求有多少时刻两个针的夹角 $\leq \alpha$，其中 $\alpha = \dfrac{2\pi A}{HM}$

Solution

假设当前时间为 t，我们需要求夹角小于 $\alpha = \dfrac{2\pi A}{HM}$ ，当前时间 $t \in [0, HM)$，时针的角度为 $\frac{2\pi t}{HM}$ ，分钟角度为 $\frac{2\pi t}{M}$。

则方程 $|\frac{2\pi t}{HM}\%2\pi - \frac{2\pi t}{M}\%2\pi| \leq \frac{2\pi A}{HM}$ 的所有正整数解 t 的个数即问题的答案。

将方程化简得：

$$\begin{gathered} t(H-1) \mod HM \leq A \\ \text{or} \\ t(H-1) \mod HM \geq HM - A \end{gathered} \tag{6}$$

根据同余的同除性：

若 $ca \equiv cb \pmod m$ 则 $a \equiv b \pmod{\frac{m}{\gcd(m,c)}}$

$t(H-1) \mod HM$ 的取值会重复 $\gcd(H-1, HM)$ 次。

因此我们将不等式同除 $d = \gcd(H-1, HM)$，得：

$$\begin{gathered} \frac{t(H-1)}{d} \mod \frac{HM}{d} \leq \frac{A}{d} \\ \text{or} \\ \frac{t(H-1)}{d} \mod \frac{HM}{d} \geq \frac{HM-A}{d} \end{gathered} \tag{7}$$

显然第一个式子在 $[0, \dfrac{HM}{d}]$ 中解 t 共有 $0 \sim \dfrac{A}{d}$，共 $\dfrac{A}{d} + 1$ 种取值。

第二个式子共有 $\dfrac{HM}{d} - \dfrac{HM-A}{d} = \dfrac{A}{d}$ 种取值，共 $2 \times \dfrac{A}{d} + 1$ 种取值。

乘上除去的 d，原式在 $[0, HM]$ 共有 $d \times (2 \times \dfrac{A}{d} + 1)$ 种取值。

注意当 $A = \dfrac{HM}{2}$ 时，显然答案为 HM，特判即可。

Code

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
const int maxn = 1e5 + 7;
int n, m, s, t, k;
ll ans;
ll H, M, A;
int main()
{
    scanf("%lld%lld%lld", &H, &M, &A);
    if(A == H * M / 2 ) {
        printf("%lld\n", H * M);
        return 0;
    }
    ll d = __gcd(H - 1, H * M);
    ans = d * (2ll * (A / d) + 1);
```

```
    cout << ans << endl;
    return 0;
}
```

0x24.2 中国剩余定理

- **同余方程组**

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \cdots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

其中 $a_1, a_2, \ldots, a_n$ 是整数，$m_1, m_2, \ldots, m_n$是正整数且两两互质。

- **中国剩余定理**

我们设：

$M = m_1 \times m_2 \times \cdots \times m_n$，$M_i = \frac{M}{m_i}$

$M_i \times M_i^{-1} \equiv 1 \pmod{}m_i$，其中$M_i^{-1}$是$M_i$的逆元。

我们可以构造出一个解 $x = \sum_{i=1}^{k} a_i M_i M_i^{-1}$

由此，任意解x_0 即为$x + k \times M$

最小正整数解 $x_{\min} = x_0 \,\%\, M$

如何证明构造出来的解 x 对于所有的同余方程都成立呢?
我们首先对第一个式子，$x \bmod m_1$ ，其中 $i \in 2 \sim k$ 中的 M_i 里面都是 m_i 的倍数，$M_i \bmod m_i$ 都等于 0，$M_1 \times M_1^{-1} \equiv 1 \pmod{m_i}$，也就是说只有第一项 $\bmod\ m_i$ 不为 0，等于 a_i ，那么也就是说满足第一个同余方程，同理也满足所有的同余方程。

Code

```
const ll N = 5e5 + 7;
int n, a[N], m[N];
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if(b == 0){
        x = 1; y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
    ll z = x;x = y;y = z - (a / b) * x;
    return d;
}
int main()
{
    ll M = 1;
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i) {
        scanf("%d%d", &m[i], &a[i]);
        M *= m[i];
    }
    ll res = 0;
    for(int i = 1; i ≤ n; ++ i) {
        ll Mi = M / m[i];
        ll ti, y;
        //exgcd求逆元：解同余方程：ax + my = 1;(ax ≡ 1 mod m)
        ll d = exgcd(Mi, m[i], ti, y);
        ti = (ti % m[i] + m[i]) % m[i];
        res += a[i] * ti * Mi;
    }
    printf("%lld\n", (res % M + M) % M);//可能为负数，所以需要处理一下
    return 0;
}
```

0x24.3 拓展中国剩余定理

仍然是上面的问题，只是 $m_1 c \ldots m_n$ 不保证两两互质了。

考虑如果只有两个方程如何求解：

$$\begin{cases} x \equiv a_1 \pmod{m_1} & (1) \\ x \equiv a_2 \pmod{m_2} & (2) \end{cases}$$

对于第一个方程而言，解的形式是 $x = a_1 + km_1$ 。带入第二个方程，得到：$a_1 + km_1 = a_2 \pmod{m_2}$

这个方程中只有 k 是未知的。用扩展欧几里得解出来一个 k_0 ，则任意 $km_1 = k_0m_1 + zm_2$ 对于等式 (2) 都是合法的，为了满足等式 (1) 所以要求 $zm_2 = u \times \operatorname{lcm}(m_1, m_2)$ 。现在，x 的表现形式为 $a_1 + k_0m_1 + u \times \operatorname{lcm}(m_1, m_2)$ ，我们合并出了一个新的方程：
$x \equiv a_1 + k_0m_1 \pmod{\operatorname{lcm}(m_1, m_2)}$
$z, u \in \mathbb{Z}$。

然后再合并 n 次即可。

Code

```cpp
typedef long long ll;const int N = 100007, M = 1000007, INF = 0x3f3f3f3f;
const double eps = 1e-8;
const int mod = 10007;
ll n, m;
ll bi[N], ai[N];
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if(b == 0){x = 1; y = 0; return a;}
    ll d = exgcd(b, a % b, x, y);
    ll z = x;x = y;y = z - a / b * y;
    return d;
}
ll mul(ll a, ll b, ll c)//注意数据范围可能会爆long long需要用到龟速乘
{
    if(b < 0) a = - a, b = - b;
    ll res = 0;
    while(b){
        if(b & 1)res  = (res + a) % c;
        a = (a + a) % c;
        b >>= 1;
    }
    return res;
}
ll excrt()//拓展中国剩余定理
{
    ll x, y, k;
    ll M = bi[1], ans = ai[1];//第一个方程的特解
    for(int i = 2; i ≤ n; ++ i) {
        ll a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;
        ll d = exgcd(a, b, x, y);
        ll bg = b / d;//lcm
        if(c % d =≠ 0)return -1; //判断是否无解，然而这题其实不用
        x = mul(x, c / d, bg);
        ans += x * M;//更新前k个方程组的答案
        M *= bg;//M为前k个m的lcm
        ans = (ans % M + M) % M;
    }
    ans = (ans % M + M) % M;
    //if(ans == 0) ans = M;//视情况而定，等于0的时候是因为给定的模数均为1，此时答案应该取任意值均可，而不是只有解 0 ，有时需要特判一下。
    return ans;
}

int main()
{
    scanf("%lld", &n);
    //bi -> m[i], ai -> a[i]
    for(int i = 1; i ≤ n; ++ i)
        scanf("%lld%lld", &bi[i], &ai[i]);
    printf("%lld\n", excrt());
    return 0;
}
```

A、（P3868 [TJOI2009]）猜数字

Weblink

https://www.luogu.com.cn/problem/P3868

Problem

现有两组数字，每组 k 个。

第一组中的数字分别用 $a_1, a_2, \cdots, a_k$ 表示，第二组中的数字分别用 $b_1, b_2, \cdots, b_k$ 表示。

其中第二组中的数字是两两互素的。求最小的 $n \in \mathbb{N}$，满足对于 $\forall i \in [1, k]$，有 $b_i | (n - a_i)$。

Solution

$b_i \mid (n - a_i)$
$(n - a_i) \% b_i = 0$
$n - a_i \equiv 0 \pmod{b_i}$
$n \equiv a_i \pmod{b_i}$

显然有 n 个同余方程，直接 CRT 解方程组即可。

注意数据保证 $\prod_{i=0}^{k} b_i \leq 10^{18}$，即 $M \leq 10^{18}$，那么CRT的过程中随便一乘就会爆 `long long` ，所以要用快速乘。用 $\log n$ 的快速乘竟然 T 了... 所以需要加一些经典优化或者用 $O(1)$ 的 `long double` 快速乘，可以处理小于 1.7×10^{308} 的数据。

Code

```cpp
#include <bits/stdc++.h>
#include <algorithm>
#include <cstring>
#include <cmath>
#include <map>
#include <queue>
using namespace std;
#define int long long
typedef long long ll;
typedef int itn;
typedef pair<int, int> PII;
typedef pair<double, int> PDI;
const int N = 50 + 7, mod = 1e18 + 7, INF = 0x3f3f3f3f;
const double PI = acos(-1.0), eps = 1e-8;

int n, t, kcase;
int M, m[N], a[N], k;
/*
int mul(int a, int b, int mod)
{
    int res = 0;
    while(b) {
        if(b & 1) res = (res + a) % mod;
        a = (a + a) % mod;
        b >>= 1;
    }
    return res;
}*/

int mul(int x, int y, int p)
{
    int z = (long double)x / p * y;
    ll res = (unsigned long long)x * y - (unsigned long long) z * p;
    return (res + p) % p;
}

ll exgcd(int a, int b, int &x, int &y)
{
    if(b == 0) {
        x = 1, y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
    ll z = x;
    x = y;
    y = z - y * (a / b);
    return d;
}

void solve()
```

```cpp
{
    ll M = 1;
    scanf("%lld", &k);
    for(int i = 1; i ≤ k; ++ i) {
        scanf("%lld", &a[i]);
    }
    for(int i = 1; i ≤ k; ++ i) {
        scanf("%lld", &m[i]);
        M *= m[i];
    }
    for(int i = 1; i ≤ k; ++ i) a[i] = (a[i] % m[i] + m[i]) % m[i];
    ll res = 0;
    for(int i = 1; i ≤ k; ++ i) {
        ll Mi = M / m[i];
        ll ti, y;
        ll d = exgcd(Mi, m[i], ti, y);
        ti = (ti % m[i] + m[i]) % m[i];
        res += mul(mul(a[i], ti, M), Mi, M);
    }
    printf("%lld\n", (res % M + M) % M);
}

signed main()
{
    solve();
    return 0;
}
```

B、古代猪文 （P2480 [SDOI2010]）

Problem

猪王国的文明源远流长，博大精深。

iPig 在大肥猪学校图书馆中查阅资料，得知远古时期猪文文字总个数为 n。当然，一种语言如果字数很多，字典也相应会很大。当时的猪王国国王考虑到如果修一本字典，规模有可能远远超过康熙字典，花费的猪力、物力将难以估量。故考虑再三没有进行这一项劳猪伤财之举。当然，猪王国的文字后来随着历史变迁逐渐进行了简化，去掉了一些不常用的字。

iPig 打算研究古时某个朝代的猪文文字。根据相关文献记载，那个朝代流传的猪文文字恰好为远古时期的 $1/k$，其中 k 是 n 的一个正约数（可以是 1 或 n）。不过具体是哪 $1/k$，以及 k 是多少，由于历史过于久远，已经无从考证了。

iPig 觉得只要符合文献，每一种 $k|n$ 都是有可能的。他打算考虑到所有可能的 k。显然当 k 等于某个定值时，该朝的猪文文字个数为 n/k。然而从 n 个文字中保留下 n/k 个的情况也是相当多的。iPig 预计，如果所有可能的 k 的所有情况数加起来为 p 的话，那么他研究古代文字的代价将会是 g^p。

现在他想知道猪王国研究古代文字的代价是多少。由于 iPig 觉得这个数字可能是天文数字，所以你只需要告诉他答案除以 999911659 的余数就可以了。

Solution

void 函数写成 ll 了，没有返回值 RE 了...

懒得写题解了...

题目大意：求$G^{\sum d|n\ C_n^d}\ mod\ 999911659$

思路与其他题解相像，考虑到999911659是质数，那么就用欧拉定理的推论得：

$$G^{\sum d|n\ C_n^d}\ mod\ 999911659 = G^{\sum d|n\ C_n^d\ mod\ 999911658}\ mod\ 999911659$$

那么关键计算$\sum d|n\ C_n^d\ mod\ 999911658$.直接$Lucas$绝对挂，那么尝试把模数缩小再合并

将999911658因数分解，可得$999911658 = 2 \times 3 \times 4679 \times 35617$.那么把模数缩小，枚举$n$的因数$d$，然后运用$Lucas$定理把$C_n^d$算出来，分别计算出$\sum d|n\ C_n^d$对$2, 3, 4679, 35617$四个质数取模的结果，记为$a_1, a_2, a_3, a_4$.

最后，用中国剩余定理求解一下方程组：

$$\begin{cases} x \equiv a_1 (\mathrm{mod}\ 2) \\ x \equiv a_2 (\mathrm{mod}\ 3) \\ x \equiv a_3 (\mathrm{mod}\ 4679) \\ x \equiv a_4 (\mathrm{mod}\ 35617) \end{cases}$$

然后就得到了最小的非负整数解x，之后用快速幂求一下G^x就得到答案

上述题解来源...

Code

简单实现一下就行了

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef int itn;
const ll N = 5e5 + 7, p = 999911659, phi = 999911658, INF = 0x3f3f3f3f;
const double PI = acos(-1.0), eps = 1e-8;

ll val;
ll a[10];
ll M = 1;
ll fact[N];
ll infact[N];
ll n, g, t, cnt;
ll mo[10] = {0, 2, 3, 4679, 35617};

ll qpow(ll a, ll b, ll mod)
{
    ll res = 1;
    while(b) {
        if(b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
     }
     return res;
}

ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if(b == 0) {
        x = 1, y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
    ll z = x;x = y, y = z - y * (a / b);
    return d;
}

void init(ll p)
{
    fact[0] = infact[0] = 1;
    for(ll i = 1; i < p; ++ i) {
        fact[i] = fact[i - 1] * i % p;
        infact[i] = infact[i - 1] * qpow(i, p - 2, p) % p;
    }
}
```

```cpp

ll C(ll a, ll b, ll p)
{
    if(a < b) return 0;
    return fact[a] * infact[b] % p * infact[a - b] % p;
}


ll lucas(ll a, ll b, ll p)
{
    if(b == 0) return 1;
    if(a < p && b < p) return C(a, b, p);
    return C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
}

ll CRT()
{
    ll res = 0;
    for(ll i = 1; i ≤ 4; ++ i) {
        ll Mi = M / mo[i];
        ll ti, y;
        ll d = exgcd(Mi, mo[i], ti, y);
        ti = (ti % mo[i] + mo[i]) % mo[i];
        res = (res + a[i] * ti % M * Mi % M) % M;
    }
    return (res % M + M) % M;
}

void solve()
{
    M = phi;
    scanf("%lld%lld", &n, &g);
    if(g % p == 0) {
        puts("0");
        return ;
    }

    for(ll k = 1; k ≤ 4; ++ k) {
        init(mo[k]);
        for(ll i = 1; i * i ≤ n; ++ i) {
            if(n % i == 0) {
                a[k] = (a[k] + lucas(n, i, mo[k])) % mo[k];
                if(i * i ≠ n)
                    a[k] = (a[k] + lucas(n, n / i, mo[k])) % mo[k];
            }
        }
    }
    printf("%lld\n", qpow(g, CRT(), p));
}

int main()
{
    solve();
    return 0;
}
```

Problem C. 屠龙勇士（P4774 [NOI2018]）

小 D 最近在网上发现了一款小游戏。游戏的规则如下：

- 游戏的目标是按照编号 $1 \to n$ 顺序杀掉 n 条巨龙，每条巨龙拥有一个初始的生命值 a_i 。同时每条巨龙拥有恢复能力，当其使用恢复能力时，它的生命值就会每次增加 p_i ，直至生命值非负。只有在攻击结束后且当生命值 **恰好** 为 0 时它才会死去。
- 游戏开始时玩家拥有 m 把攻击力已知的剑，每次面对巨龙时，玩家只能选择一把剑，当杀死巨龙后这把剑就会消失，但作为奖励，玩家会获得全新的一把剑。

小 D 觉得这款游戏十分无聊，但最快通关的玩家可以获得 IONI2018 的参赛资格，于是小 D 决定写一个笨笨的机器人帮她通关这款游戏，她写的机器人遵循以下规则：

- 每次面对巨龙时，机器人会选择当前拥有的，攻击力不高于巨龙初始生命值中攻击力最大的一把剑作为武器。如果没有这样的剑，则选择 **攻击力最低** 的一把剑作为武器。
- 机器人面对每条巨龙，它都会使用上一步中选择的剑攻击巨龙固定的 x 次，使巨龙的生命值减少 $x \times ATK$ 。
- 之后，巨龙会不断使用恢复能力，每次恢复 p_i 生命值。若在使用恢复能力前或某一次恢复后其生命值为 0 ，则巨龙死亡，玩家通过本关。

那么显然机器人的攻击次数是决定能否最快通关这款游戏的关键。小 D 现在得知了每条巨龙的所有属性，她想考考你，你知道应该将机器人的攻击次数 x 设置为多少，才能用最少的攻击次数通关游戏吗?

当然如果无论设置成多少都无法通关游戏，输出 -1 即可。

Solution

可以直接用 `multiset` 代替平衡树，计算出每条龙所需要的剑，并且这把剑是一定能打败这条龙的，不然最后方程组无解，就会输出 -1，所以我们直接把打败龙的奖励放入 set 里，再选下一把剑。显然题目中的恢复机制我们可以得到一个同余方程：

$C_i x \equiv A_i (\mod P_i)$

题目数据不保证 `m[i]` 一定互质 $\Rightarrow$ excrt

一般的同余方程为：

$x \equiv A_i (\mod P_i)$

可以直接用拓展中国剩余定理。

但是本题的式子长这个样子：

$C_i x \equiv A_i (\mod P_i)$

考虑转成标准式子

$C_i x \equiv A_i (\mod P_i)$
显然有 $C_i x + P_i y = A_i$
exgcd解得 x' y'

$$x = x' + k\frac{b}{d}$$
$$= x' + k\frac{P_i}{\gcd(P_i, C_i)}$$
$$(\mod \frac{P_i}{\gcd(P_i, C_i)})$$
$$\Rightarrow x \equiv x' (\mod \frac{P_i}{\gcd(P_i, C_i)})$$

拓展中国剩余定理求解即可。

注意特判 $A_i > P_i$ 的情况即可，此时 $P_i = 1$，答案显然就是 $max\{\frac{A[i]}{use[i]}\}$

Code

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define int long long
typedef pair<int, int> PII;
const ll N = 5e6 + 7, INF = 0x3f3f3f3f;
const double PI = acos(-1.0), eps = 1e-8;

int n, m, k;
int A[N], P[N], award[N], atk[N], one;
multiset <int> st;
int C[N], use[N];
int ai[N], bi[N];

void init()
{
    one = true;
    st.clear();
```

```
    memset(use, 0, sizeof use);
}

int mul(int x, int y, int p)
{
    int z = (long double)x / p * y;
    int res = (unsigned long long)x * y - (unsigned long long) z * p;
    return (res + p) % p;
}

int exgcd(int a, int b, int &x, int &y)
{
    if(b == 0) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;
    x = y, y = z - (a / b) * y;
    return d;
}

int excrt()
{
    int x, y, k;
    int M = bi[1], ans = ai[1];
    for(int i = 2; i ≤ n; ++ i) {
        int a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;
        int d = exgcd(a, b, x, y);
        int bg = b / d;
        if(c % d ≠ 0) return -1;
        x = mul(x, c / d, bg);
        ans += x * M;
        M *= bg;
        ans = (ans % M + M) % M;
    }
    return (ans % M + M) % M;
}

void sp_work()
{
    int ans = - INF;
    for(int i = 1; i ≤ n; ++ i) {
        if(use[i]) {
            ans = max(ans, (int)ceil((double)A[i] / use[i]));
        }
    }
    printf("%lld\n", ans);
}

bool work()
{
    int x, y;
    for(int i = 1; i ≤ n; ++ i) {
        int a = use[i], b = P[i], c = A[i];
        int d = exgcd(a, b, x, y), bg = b / d;
        if(c % d ≠ 0)
            return false;
        x = mul(x, c / d, bg);
        x = (x % b + b) % b;
        ai[i] = x;
        bi[i] = bg;
     }
     return true;
}

void solve()
{
    init();
```

```
    scanf("%lld%lld", &n, &m);
    for(int i = 1; i ≤ n; ++ i) scanf("%lld", &A[i]);
    for(int i = 1; i ≤ n; ++ i) {
        scanf("%lld", &P[i]);
        if(A[i] > P[i])
            one = false;
    }
    for(int i = 1; i ≤ n; ++ i) scanf("%lld", &award[i]);
    for(int i = 1; i ≤ m; ++ i) {
        scanf("%lld", &atk[i]);
        st.insert(atk[i]);
    }

    for(int i = 1; i ≤ n; ++ i) {
        multiset <int> :: iterator it;
        if(A[i] < *st.begin()) it = st.begin();
        else it =  -- st.upper_bound(A[i]);
        use[i] = *it;
        st.insert(award[i]);
        st.erase(it);
    }
    if(one == false) {
        sp_work();
        return ;
    }
    bool win = work();
    if(win == 0) {
        puts("-1");
        return ;
    }

    printf("%lld\n", excrt());
    return ;
}


signed main()
{
    int t;
    scanf("%lld", &t);
    while(t -- ) {
        solve();
    }
    return 0;
}

/*
487472861809
3871865111
7560798679
584853762636
670310334583
*/
```

0x25 高次同余方程（一）

关于高次同余方程，一般有 $a^x \equiv b \pmod p$ 以及 $x^a \equiv b \pmod p$两种，这里只讨论第一种高次同余方程，第二种高次同余方程需要利用原根，阶，指标等概念，我们将在下面介绍这些概念，第二种高次同余方程的解法，详见本文0x64.5 高次同余方程（二）

0x25.1 BSGS算法

我们这里使用BSGS（Baby Step,Giant Step）算法求解第一类高次同余方程。

> 给定正整数 a,b,p，保证 a,p 互质，求最小的非负整数 t，满足$a^t \equiv b \pmod p$。

显然，由欧拉定理，$a^{\varphi(p)} \equiv 1 \pmod p$，故 $a^t \equiv a^{t \bmod \varphi(p)} \pmod p$，故 $t < \varphi(p)$ ，即 $t \in [0,\varphi(p))$ ，换句话说， a^t 在模 p 意义下有一个长度为 $\varphi(p)$ 的**循环节**，显然我们只需要暴力枚举 $t \in [0,\varphi(p))$ 判断是否满足该高次同余方程即可，因为之后是无限的循环之中。时间复杂度为 $O(\varphi(p))$，但是当 p 是质数的时候，$\varphi(p) = p-1$，所以是一个 $O(n)$ 的算法，考虑优化。

我们设 $t = kx - y$，其中 k 为我们自己选定的一个定值，则原同余方程变为 $a^{kx-y} \equiv b \pmod p$

$$
\begin{aligned}
a^{kx-y} &\equiv b \pmod{p} \\
a^{kx} \times a^{-y} &\equiv b \pmod{p} \\
a^{kx} &\equiv ba^{y} \pmod{p}
\end{aligned} \tag{8}
$$

此时 x, y 未知，其中显然有 $x < \frac{\varphi(p)}{k}$，$y < k$ 。

我们显然可以从 1 到 k 枚举 y ，预处理出所有的 $ba^y \bmod p$，相同的只保留最小的 y（因为我们要求的是最小的正整数解，如果求的是最大的正整数解的话就要保留最大的 y ），时间复杂度为 $O(k)$。

然后再从 1 到 $\frac{\varphi(p)}{k}$ 枚举 x，在哈希表中查询判断是否有 $a^{kx} \equiv ba^y \pmod{p}$，找到则时间复杂度为 $O(\frac{\varphi(p)}{k})$。

这样处理给定 a, p，询问给出 b 的问题，时间复杂度为 $O(\frac{\varphi(p)}{k} + k)$，显然要想 $\frac{\varphi(p)}{k} + qk$ 最小，取 $\frac{\varphi(p)}{k} = k$，$k = \sqrt{\varphi(p)}$ 时，时间复杂度最优。但是我们没必要专门再计算出 $\varphi(p)$ 的值，显然有 $\sqrt{\varphi(p)} < \lceil \sqrt{p} \rceil$，我们直接取 $k = \lceil \sqrt{p} \rceil$，则整体的时间复杂度为 $O(\sqrt{p})$。

Hint

- 这里取$\lceil \sqrt{p} \rceil$是因为我们的 $t \in [0, \varphi(p) - 1]$，但是如果对于 p 求 $\varphi(p)$，若 p 过大，开销较大，没有必要，而我们知道 $\sqrt{\varphi(p)} < \lceil \sqrt{p} \rceil$，故我们可以就近取 $k = \lceil \sqrt{p} \rceil$ 作为代替，只要完全包含整个区间，枚举的时候可以覆盖整个区间，稍微多一点无所谓。

- 我们设 $t = kx - y$，选择 $-y$ 是因为我们下面要将 y 放到右边，减去，所以这里设为 $-y$ 会变成 $+y$ ，不用求逆元了，因为 y 是常数，所以取正负无影响。
- $0 \le x \le k, k = \lceil \sqrt{p} \rceil$，但是我们循环的时候要从 0开始循环，$x = 0$ 的情况单独出来特判，因为我们要求的是最小的正整数解 t，而如果 $x = 0$ ，$t = kx - y$，t 会为负数，不符合题意，故将 $x = 0$ 的情况单独拿出来特判。

Code

```cpp
typedef long long ll;
int BSGS(int a, int b, int p)
{
    if (1 % p == b % p) return 0;
    int k = sqrt(p) + 1;
    unordered_map<int, int> hash;
    for (int i = 0, j = b % p; i < k; i ++ )
    {
        hash[j] = i;
        j = (ll)j * a % p;
    }
    int ak = 1;
    for (int i = 0; i < k; i ++ ) ak = (ll)ak * a % p;

    for (int i = 1, j = ak; i ≤ k; i ++ )
    {
        if (hash.count(j)) return (ll)i * k - hash[j];
        j = (ll)j * ak % p;
    }
    return -1;
}

int main()
{
    int a, p, b;
    //a^x ≡ b (mod p)
    cin >> p >> a >> b ;

    int res = BSGS(a, b, p);
    if (res == -1) puts("no solution");
    else cout << res << endl;

    return 0;
}
```

0x25.2 拓展BSGS算法

> 给定 a, p, b，求满足 $a^x \equiv b \pmod{p}$ 的最小自然数 x ，$a, p, b \le 10^9$ 。**不保证 a, p 互质**。

如果 a, p 不互质，就需要进行一些扩展

如果 $b=0$，则 $p=1$ 时有解，否则无解

如果 $b=1$，则 $x=0$

否则，设 $d=\gcd(a,p)$，如果 $d\nmid b$ 则无解，否则两边同除以 d，得到：
$(\frac{a}{d})a^{x-1}\equiv\frac{b}{d}\pmod{\frac{p}{d}}$
因为$\frac{a}{d},\frac{p}{d}$互质，所以
$a^{x-1}\equiv(\frac{a}{d})^{-1}(\frac{b}{d})\pmod{\frac{p}{d}}$

多次执行上面的过程，直到 a,p 互质，然后使用BSGS求解即可。

Code

```cpp
typedef long long ll;
map<ll, int> mp;
inline int read() {
    int ans = 0;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        ch = getchar();
    }
    while (ch ≥ '0' && ch ≤ '9') {
        ans = ans * 10 + (ch ^ 48);
        ch = getchar();
    }
    return ans;
}

inline ll qpow(ll x, ll p, ll mod) {
    ll ans = 1;
    while (p) {
        if (p & 1)
            ans = ans * x % mod;
        x = x * x % mod;
        p >>= 1;
    }
    return ans;
}

inline int bsgs(int a, ll b, int p) {
    if (p == 1)
        return 0;
    a %= p;
    b %= p;
    if (b == 1)
        return 0;
    int m = ceil(sqrt(p - 1)), i = 0;
    ll t = qpow(a, m, p);
    mp.clear();
    for (register ll j = b; i < m; i++, j = j * a % p) {
        mp[j] = i;
    }
    i = 1;
    for (register ll j = t; i ≤ m; i++, j = j * t % p) {
        if (mp.count(j))
            return i * m - mp[j];
    }
    return -1;
}

int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}

void exgcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return;
    }
    ll t;
```

```cpp
    exgcd(b, a % b, x, y);
    t = x;
    x = y;
    y = t - a / b * y;
}

inline ll inv(ll a, ll b) {
    ll x, y;
    exgcd(a, b, x, y);
    return (x % b + b) % b;
}

inline int exbsgs(int a, int b, int p) {
    if (p == 1)
        return 0;
    a %= p;
    b %= p;
    if (b == 1)
        return 0;
    int x = 0, t, ans;
    ll y = 1;
    while ((t = gcd(a, p)) ≠ 1) {
        if (b % t ≠ 0)
            return -1;
        b /= t;
        p /= t;
        x++;
        y = y * (a / t) % p;
        if (b == y)
            return x;
    }
    ans = bsgs(a, b * inv(y, p) % p, p);
    if (ans == -1)
        return -1;
    return ans + x;
}

void write(int n) {
    if (n ≥ 10)
        write(n / 10);
    putchar(n % 10 + '0');
}

int main() {
    while (true) {
        int a = read(), p = read(), b = read(), ans;
        if (a == 0 && p == 0 && b == 0)
            break;
        ans = exbsgs(a, b, p);
        if (ans == -1) {
            cout << "No Solution" << endl;
        } else {
            write(ans);
            putchar('\n');
        }
    }
    return 0;
}
```

- **竞赛例题选讲**

Problem A 随机数生成器　[luogu P3306 [SDOI2013]](https://www.luogu.com.cn/problem/P3306)

最近小 W 准备读一本新书，这本书一共有 p 页，页码范围为 $0 \sim p-1$。

小 W 很忙，所以每天只能读一页书。为了使事情有趣一些，他打算使用 NOI2012 上学习的线性同余法生成一个序列，来决定每天具体读哪一页。

我们用 x_i 来表示通过这种方法生成出来的第 i 个数，也即小 W 第 i 天会读哪一页。这个方法需要设置 3 个参数 a, b, x_1 ，满足 $0 \leq a, b, x_1 < p$，且 a, b, x_1 都是整数。按照下面的公式生成出来一系列的整数：

$$x_{i+1} \equiv a \times x_i + b \pmod{p}$$

其中 mod 表示取余操作。

但是这种方法可能导致某两天读的页码一样。

小 W 要读这本书的第 t 页，所以他想知道最早在哪一天能读到第 t 页，或者指出他永远不会读到第 t 页。

Solution

根据题意我们可以得到一个递推公式：

$$x_n = ax_{n+1} + b \tag{9}$$

（经典高中数列套路）设：

$$\begin{aligned} &x_n + c = a(x_{n-1} + c) \\ &x_n + c = ax_{n-1} + ac \\ &x_n = ax_{n-1} + ac - c \\ &x_n = ax_{n-1} + (a-1)c \\ &\rightarrow (a-1)c = b \rightarrow c = \frac{b}{a-1} \end{aligned} \tag{10}$$

$$x_n + \frac{b}{a-1} = a(x_{n-1} + \frac{b}{a-1})$$

即可得到：

$$\begin{aligned} x_n + \frac{b}{a-1} &= a(x_{n-1} + \frac{b}{a-1}) \\ &= a(a(x_{n-2} + \frac{b}{a-1})) \\ &= \cdots \\ &= a^{n-1}(x_1 + \frac{b}{a-1}) \end{aligned} \tag{11}$$

显然根据题意， $x_n = t$
我们就可以将答案的表达式转化为标准的高次同余方程的形式：

$$a^{n-1} \equiv \frac{t + \frac{b}{a-1}}{x_1 + \frac{b}{a-1}} \pmod{p} \tag{12}$$

使用 BSGS 算法求解即可。

注意观察等式，需要特判几个特殊情况，因为有分数，分母不能为 0：

1. $a = 1 \rightarrow a - 1 = 0$
2. $a = 0$
3. $x_1 + \frac{b}{a-1} = 0$
4. $a = 1, b = 0$，当 $n = 1$ ，$x_n = x_1$，当 $n \geq 2$，$x_n = b$
5. $a = 1, b \neq 0$，
   $x_n \equiv x_{n-1} + b \pmod{p} \rightarrow x_n = x_{n-1} + b \rightarrow x_n = (n-1)b + x_1 \rightarrow x_n \equiv (n-1)b + x_1 \pmod{p} \rightarrow b(n-1) + py = t - x_1 \rightarrow exgcd, x = n - 1$

Code

```cpp
#include <bits/stdc++.h>
using namespace std;
#define int long long
typedef long long ll;
typedef pair<int, int> PII;
const ll N = 5e6 + 7, INF = 0x3f3f3f3f;
const double PI = acos(-1.0), eps = 1e-8;

int n, m, p, a, b, x1, t, T;

int qpow(int a, int b, int mod)
{
    int res = 1;
    while(b) {
        if(b & 1) res = res * a % mod;
        a = a * a % mod;
```

```
        b >>= 1;
    }
    return res;
}

int inv(int a, int p)
{
    return qpow(a, p - 2, p);
}

int exgcd(int a, int b, int &x, int &y)
{
    if(b == 0) {
        x = 1; y = 0;
        return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;
    x = y, y = z - (a / b) * y;
    return d;
}

int BSGS(int a, int b, int p)
{
    if (1 % p == b % p) return 0;
    int k = sqrt(p) + 1;
    unordered_map<int, int> hash;
    for (int i = 0, j = b % p; i < k; i ++ )
    {
        hash[j] = i;
        j = j * a % p;
    }
    int ak = 1;
    for (int i = 0; i < k; i ++ ) ak = ak * a % p;

    for (int i = 1, j = ak; i ≤ k; i ++ )
    {
        if (hash.count(j)) return  i * k % p - hash[j] % p;
        j = j * ak % p;
    }
    return -1;
}

void solve()
{
    scanf("%lld%lld%lld%lld%lld", &p, &a, &b, &x1, &t);
    if(a == 0) {
        if(x1 == t) puts("1");
        else if(b == t) puts("2");
        else puts("-1");
        return ;
    }
    else if(a == 1) {
        if(b == 0) {
            if(x1 == t) puts("1");
            else puts("-1");
            return ;
        }
        int x, y;
        int d = exgcd(b, p, x, y);
        x = (x * (t - x1) % p / d + p) % p;
        printf("%lld\n", x + 1);
        return ;
    }
    else {
        int C = b * inv(a - 1, p) % p;
        int A = (x1 + C) % p;
        if(A == 0) {
```

```
            int res = (-C + p) % p;
            if(res == t) puts("1");
            else puts("-1");
            return ;
        }
        else {
            int B = (t + C) % p * inv(A, p) % p;
            int n = BSGS(a, B, p);
            if(n == -1) puts("-1");
            else printf("%lld\n", n + 1);
            return ;
        }
    }
}
signed main()
{
    scanf("%lld", &T);
    while(T -- ) {
        solve();
    }
    return 0;
}
```

0x26 【题型探究】公约数之和

0x26.1 母题：UVA11417 GCD

Problem

题目描述

给定 n，求

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j)$$

其中 $\gcd(i,j)$ 指的是 i 和 j 的最大公约数。

输入格式

本题有多组数据。

对于每组数据，输出一个整数 n，如果 $n=0$ 就终止程序。

输出格式

对于每组数据，输出计算结果。

说明 / 范围

对于 100% 的数据，$1\le n\le 501$。

Solution

数据很小，我们只需要直接暴力枚举即可。

Code

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <vector>
#include <iostream>

using namespace std;
typedef long long ll;
const int N = 5000007;

int gcd(int a, int b)
{
    if(b == 0) return a;
    return gcd(b, a % b);
```

```
}

ll solve(int n)
{
    ll res = 0;
    for(int i = 1; i ≤ n; ++ i){
        for(int j = i + 1; j ≤ n; ++ j){
            res += gcd(i, j);
        }
    }
    return res;
}
int n;
int main()
{
    while(scanf("%d", &n) ≠ EOF && n){
        printf("%d\n", solve(n));
    }
    return 0;
}
```

0x26.2 拓展一、UVA11426 GCD - Extreme (II)

Problem

题目描述

得定 n ，求

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j)$$

其中 $\gcd(i,j)$ 指的是 i 和 j 的最大公约数。

Solution

本题的数据开到了 $4*10^6$ ，所以暴力一定TLE。

啊，这，范围太大了，我可以莫比乌斯反演 + 整除分块，$O(\sqrt{n})$ 解决!

《算法竞赛入门经典训练指南》上提供了一个构造函数的方法：

我们设 $f(n)=\gcd(1,n)+\gcd(2,n)+\gcd(3,n)+...+\gcd(n-1,n)$

答案很明显就是 $S(n)=f(2)+f(3)+...+f(n)$

我们首先考虑如何求 $f(n)$

我们可以先把 $\gcd(x,n)$ 的值分类，我们发现 $\gcd(x,n)$ 肯定是 n 的约数，再设 $g(n,x)$ 表示 $\gcd(x,n)=i$ 的小于 n 的正整数的个数。

我们发现$[\gcd(x,n)=i]->[\gcd(\frac{x}{i},\frac{n}{i})==1]$, 和 $\frac{n}{i}$ 互质的数的个数即为 $\varphi(\frac{n}{i})$个。

显然 $f(n)=\sum_{i|n}i*g(n,i)$

我们直接求一下$S(n)$，既是答案。

Code

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <vector>
#include <iostream>

using namespace std;
typedef long long ll;
const int N = 5000007;

ll s[N], primes[N];
ll f[N];
ll n, m, cnt;
ll phi[N];
bool vis[N];

void get_phi(ll n)
```

```
{
    vis[1] = true;
    for(int i = 2; i < n; ++ i){
        if(vis[i] == 0){
            primes[ ++ cnt] = i;
            phi[i] = i - 1;
        }
        for(int j = 1; j ≤ cnt && primes[j] * i < n; ++j){
            vis[i * primes[j]] = true;
            if(i % primes[j] == 0){
                phi[i * primes[j]] = phi[i] * primes[j];
                break;
            }
            else phi[i * primes[j]] = phi[i] * (primes[j] - 1);
        }
    }
}

int main()
{
    get_phi(N - 5);
    for(int i = 1; i ≤ N - 1; ++ i){
        for(int j = i * 2; j < N; j += i)
            f[j] += i * phi[j / i];
    }
    for(int i = 1; i ≤ N - 1; ++ i)
        s[i] = s[i - 1] + f[i];
    while(scanf("%lld\n", &n) == 1 && n){
        printf("%lld\n", s[n]);
    }
    return 0;
}
```

0x26.3 扩展二、luogu P2398 GCD SUM

Problem

题目描述

求

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\gcd(i,j)$$

输入格式

第一行一个整数 n。

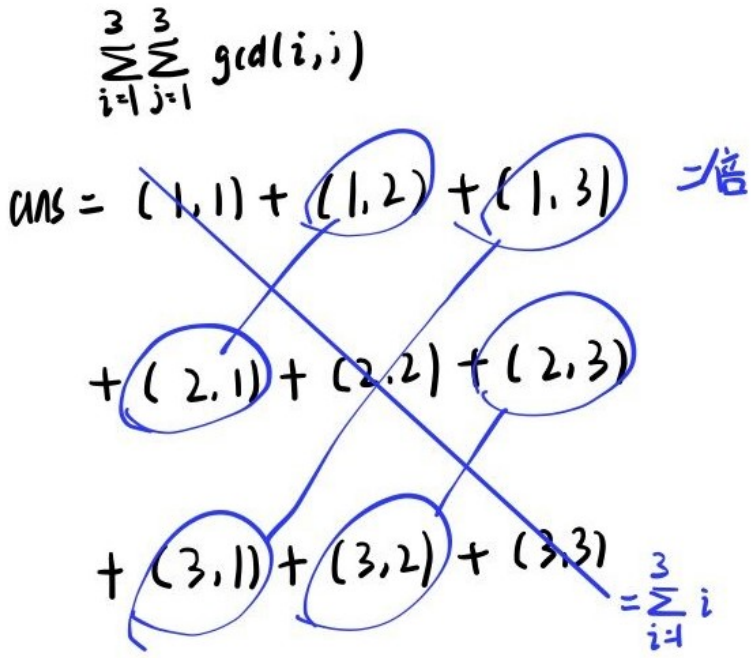
Solution

这道题要求的式子是：$\sum_{i=1}^{n}\sum_{j=1}^{n}\gcd(i,j)$

其实就是：$\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\gcd(i,j)[i<j]\right)\times 2+\sum_{i=1}^{n}i$ 答案就应该是：

$S(n)\times 2+\sum_{i=1}^{n}i$

$S(n)$是上一题的函数。



Code

```cpp
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <vector>
#include <iostream>

using namespace std;
typedef long long ll;
const int N = 100007;

ll s[N], primes[N];
ll f[N];
ll n, m, cnt;
ll phi[N];
bool vis[N];

void get_phi(ll n)
{
    vis[1] = true;
    for(int i = 2; i < n; ++ i){
        if(vis[i] == 0){
            primes[ ++ cnt] = i;
            phi[i] = i - 1;
        }
        for(int j = 1; j ≤ cnt && primes[j] * i < n; ++j){
            vis[i * primes[j]] = true;
            if(i % primes[j] == 0){
                phi[i * primes[j]] = phi[i] * primes[j];
                break;
            }
            else phi[i * primes[j]] = phi[i] * (primes[j] - 1);
        }
    }
}

int main()
{
    get_phi(N - 5);
    for(int i = 1; i ≤ N - 1; ++ i){
        for(int j = i * 2; j < N; j += i)
            f[j] += i * phi[j / i];
    }
    for(int i = 1; i ≤ N - 1; ++ i)
        s[i] = s[i - 1] + f[i];
    cin >> n;
    ll ans = 0;
    for(int i =1; i ≤ n; ++ i)
```

```
48          ans += i;
49      ans += 2 * s[n];
50      printf("%lld\n", ans);
51      return 0;
52  }
```

0x26.4 扩展三、luogu P2568 GCD

Problem

题目描述

给定正整数 n，求 $1 \le x, y \le n$ 且 $\gcd(x, y)$ 为素数的数对 (x, y) 有多少对。

输入格式

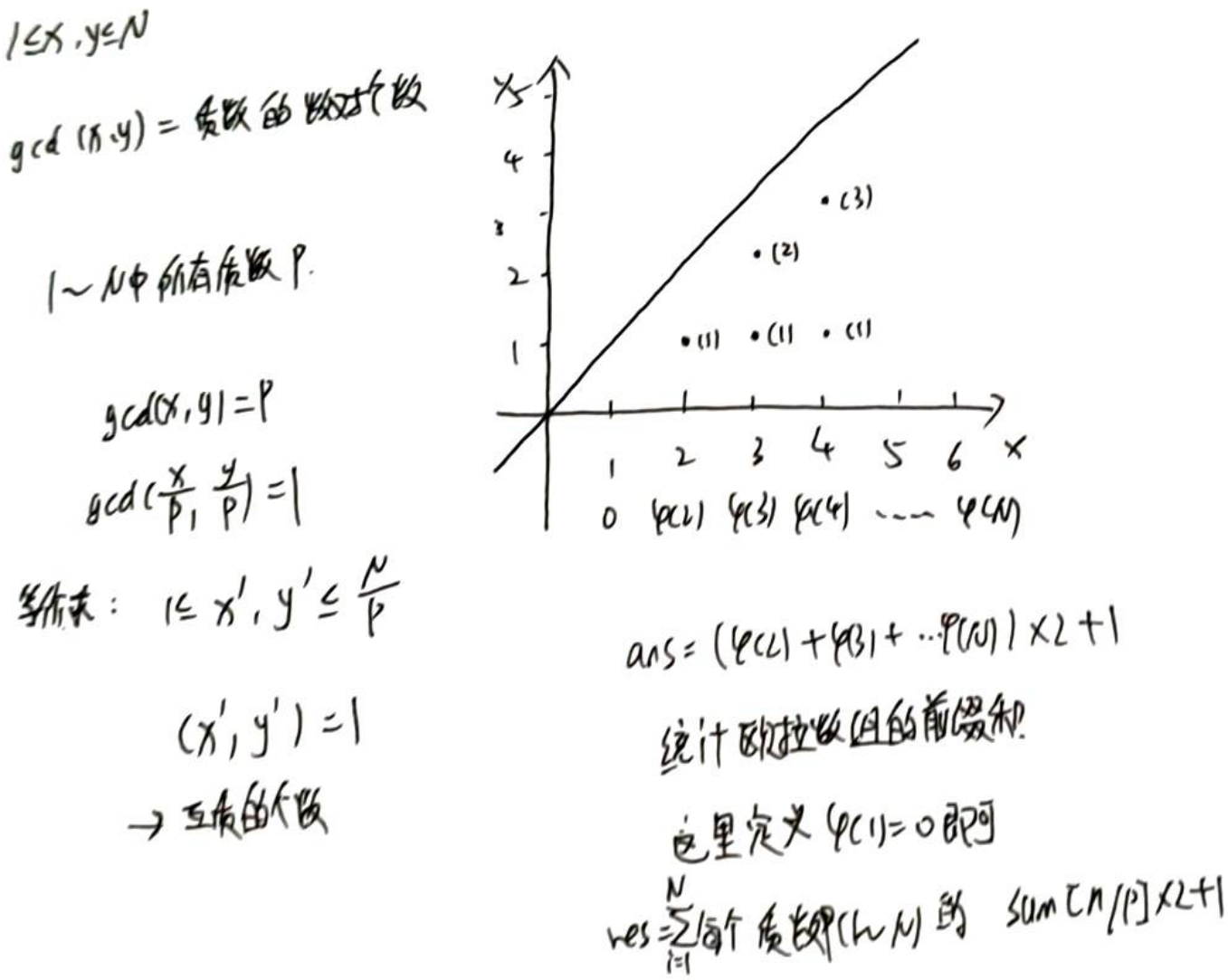
只有一行一个整数，代表n。

Solution

这道题要求的式子是：$\sum_{i=1}^{n}\sum_{j=1}^{n}[\gcd(i,j)\texttt{is prime}]$

其实就是：$\left(\sum_{i=1}^{n}\text{sumphi}(\frac{n}{i})[i\ \texttt{is prime}]\right)\times 2-\sum_{i=1}^{n}[i\ \texttt{is prime}]$

我们只需要用线性筛筛一下素数即可。



Code

```
1  #include<cstdio>
2  #include<algorithm>
3  #include<cstring>
4  #include<iostream>
5
6  using namespace std;
7  typedef long long ll;
8  const int N = 10000007;
9
```

```cpp
int n, m;
int primes[N];
int phi[N];
int cnt;
bool vis[N];
ll sum[N];

void init(int n)
{
    phi[1] = 0;//这里应该是0
    for(int i = 2; i ≤ n; ++ i){
        if(vis[i] == 0){
            primes[ ++ cnt] = i;
            phi[i] = i - 1;
        }
        for(int j = 1; j ≤ cnt && primes[j] * i ≤ n; ++ j){
            vis[i * primes[j]] = true;
            if(i % primes[j] == 0){
                phi[i * primes[j]] = phi[i] * primes[j];
                break;
            }
            phi[i * primes[j]] = phi[i] * (primes[j] - 1);
        }
    }

    for(int i = 1; i ≤ n; ++ i)//求phi[n]的前缀和
        sum[i] = sum[i - 1] + phi[i];
}

int main()
{
    scanf("%d", &n);
    init(n);

    ll ans = 0;
    for(int i = 1; i ≤ cnt; ++ i){
        int p = primes[i];
        ans += sum[n / p] * 2 + 1;
    }
    printf("%lld\n", ans);
    return 0;
}
```