数论

拓展欧几里得定理

```
void exGcd(int a, int b, int& x, int& y) {
    if (!b) { x = 1, y = 0; }
    else exGcd(b, a % b, y, x), y -= a / b * x;
}
```

```
int exgcd(int a, int b, int &x, int &y) {
  int x1 = 1, x2 = 0, x3 = 0, x4 = 1;
  while (b != 0) {
    int c = a / b;
    std::tie(x1, x2, x3, x4, a, b) =
        std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
  }
  x = x1, y = x2;
  return a;
}
```

裴蜀定理

逆定理

设 a, b 是不全为零的整数，若 $d > 0$ 是 a, b 的公因数，且存在整数 x, y, 使得 $ax + by = d$，则 $d = \gcd(a, b)$。

特殊地，设 a, b 是不全为零的整数，若存在整数 x, y, 使得 $ax + by = 1$，则 a, b 互质。

多个整数

裴蜀定理可以推广到 n 个整数的情形：设 $a_1, a_2, \ldots, a_n$ 是不全为零的整数，则存在整数 $x_1, x_2, \ldots, x_n$, 使得 $a_1x_1 + a_2x_2 + \cdots + a_nx_n = \gcd(a_1, a_2, \ldots, a_n)$。其逆定理也成立：设 $a_1, a_2, \ldots, a_n$ 是不全为零的整数，$d > 0$ 是 $a_1, a_2, \ldots, a_n$ 的公因数，若存在整数 $x_1, x_2, \ldots, x_n$, 使得 $a_1x_1 + a_2x_2 + \cdots + a_nx_n = d$，则 $d = \gcd(a_1, a_2, \ldots, a_n)$。

埃筛 最小质因数

```
std::vector<int> minp, primes;

void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}
```

威尔逊定理

1. 当且仅当p为素数时，$(p-1)! \equiv -1 (\mod p)$
2. 当且仅当p为素数时，$(p-1)! \equiv p-1 (\mod p)$
3. 若p为质数，则p能被$(p-1)!+1$整除
4. 当且仅当p为素数时，$p \mid (p-1)!+1$

欧拉函数

$$\varphi(n) = (1-1/p_1)(1-1/p_2)(1-1/p_3)(1-1/p_4)\cdots(1-1/p_n);$$

```
// 递推求欧拉函数
// primes[] 素数,phi[](fan),
for (int i = 1; i < n; ++i)
{
    if (!vis[i])
    {
        primes[++j] = i;
        phi[i] = i - 1;
    }
    for (int k = 1; k <= j; ++k)
    {
        if (primes[k] * i > n)
            break;
        vis[primes[k] * i] = 1;
        if (i % primes[k] == 0)
        {
            phi[primes[k] * i] = primes[k] * phi[i];
            break;
        }
        else
            phi[primes[k] * i] = (primes[k] - 1) * phi[i];
    }
}
//
```

扩展欧拉定理

若正整数 a 与 m 互质，则

$$a^{\varphi(m)} \equiv 1 (\bmod m)$$

推论：

$$a^b \equiv a^{b \bmod \varphi(m)} (\bmod m)$$

当 a, m 不互质时，扩展 Euler 定理表述如下：

$$a^b \equiv a^{b \bmod \varphi(m) + \varphi(m)} (\bmod m)$$

式子仅在 $\varphi(m) \leq b$ 时成立

```
#include <bits/stdc++.h>
using namespace std;
bool large_enough = false; // 判断是否有b >= phi(m)
inline int read(int MOD = 1e9 + 7) // 快速读入稍加修改即可以边读入边取模，不取模时直接模一个大于数据
{
    int ans = 0;
    char c = getchar();
    while (!isdigit(c))
        c = getchar();
    while (isdigit(c))
    {
        ans = ans * 10 + c - '0';
        if (ans >= MOD)
        {
            ans %= MOD;
            large_enough = true;
        }
        c = getchar();
    }
    return ans;
}
int phi(int n) // 求欧拉函数
{
    int res = n;
    for (int i = 2; i * i <= n; i++)
    {
        if (n % i == 0)
            res = res / i * (i - 1);
        while (n % i == 0)
            n /= i;
    }
    if (n > 1)
        res = res / n * (n - 1);
    return res;
}
int qpow(int a, int n, int MOD) // 快速幂
{
    int ans = 1;
    while (n)
    {
        if (n & 1)
            ans = 1LL * ans * a % MOD; // 注意防止溢出
        n >>= 1;
```

```
        a = 1LL * a * a % MOD;
    }
    return ans;
}
int main()
{
    int a = read(), m = read(), phiM = phi(m), b = read(phiM);
    cout << qpow(a, b + (large_enough ? phiM : 0), m);
    return 0;
}
```

费马小定理

定义

若 p 为素数，$\gcd(a, p) = 1$，则 $a^{p-1} \equiv 1 \pmod{p}$。

另一个形式：对于任意整数 a，有 $a^p \equiv a \pmod{p}$。

扩展中国剩余定理

```python
import math
def MII():
    return map(int,input().split())

def exgcd(a,b):
    if b==0:
        return a,1,0
    d,y,x = exgcd(b,a%b)
    y -= a // b * x
    return d,x,y

def solve():
    N,M = MII()
    m1,a1 = 1,0
    for _ in range(N):
        m2,a2 = MII()

        if m1 == 1:
            m1,a1 = m2,a2
            continue

        a = a2 - a1
        g = math.gcd(m1,m2)

        if a%g != 0:
            print("he was definitely lying")
            # 无解
            return

        d,x,y = exgcd(m1,m2)
        l = m1 // g * m2
        x0 = x * a // g
        b = ((m1 * x0 + a1) % l + l) % l
        m1,a1 = l,b

    if a1 <= M:
        print(a1)
        # a1即为答案(在模M意义下)
    else:
        print("he was probably lying")
```

```
solve()
```