1.3 数论

- 若 $a \equiv b \pmod m$，则 $(a, m) = (b, m)$。
- $a \equiv b \pmod{m_i} (1 \le i \le n)$ 同时成立，当且仅当 $a \equiv b \pmod{[m_1, m_2, \ldots, m_n]}$。

- **素数定理** 设 $\pi(x)$ 为不超过 x 的素数个数，当 $x\to\infty$ 时，$\pi(x)\sim\frac{x}{\ln(x)}$。

线性求逆元

- 设模数为质数 P，待求逆元的为 k，则

$$P=\lfloor\frac{P}{k}\rfloor k+(P\mod k)\Leftrightarrow\lfloor\frac{P}{k}\rfloor k+(P\mod k)\equiv 0(\mod P)$$

- 同乘 $k^{-1}(P\mod k)^{-1}$ 后移项，得

$$k^{-1}\equiv-\lfloor\frac{P}{k}\rfloor(P\mod k)^{-1}(\mod P)$$

```
    inv[1] = 1;
    for (int i = 2; i <= n; ++i)
        inv[i] = 1ll * (mod - mod / i) * inv[mod % i] % mod;
```

快速乘

算法一

- 对乘数 b 进行二进制拆分，化乘法为加法，时间复杂度 $\mathcal{O}(\log b)$。

```
inline ll quickPower(ll a, ll b)
{
    ll res = 0;
    while (b)
    {
        if (b & 1)
            add(res, a);
        add(a, a);
        b >>= 1;
    }
    return res;
}
```

算法二

- $ab\mod p=ab-\lfloor\frac{ab}{p}\rfloor p$。
- 利用 `long double` 存储 $\lfloor\frac{ab}{p}\rfloor$，时间复杂度 $\mathcal{O}(1)$，有极小的概率出错。

```
typedef long long ll;
typedef long double ld;
const ld eps = 1e-8;

inline ll quickPower(ll a, ll b)
{
    ll res = a * b - (ll)((ld)a / mod * b + eps) * mod;
    return res < 0 ? res + mod : res;
}
```

扩展欧几里得

- 设 $d=\gcd(a,b)$，求 $ax+by=d$ 的一组 (x,y)

$$ax+by=d, bx'+(a \mod b)y'=d$$
$$bx'+(a-\lfloor\frac{a}{b}\rfloor b)y'=d$$
$$ay'+b(x'-\lfloor\frac{a}{b}\rfloor y')=d$$
$$x=y', y=x'-\lfloor\frac{a}{b}\rfloor y'$$

- 迭代到 $a'=d, b'=0$ 可得 $x'=1, y'=0$，由数学归纳法可得，若 $b\neq 0$，最后解的大小满足：
$$|x|\leq b, |y|\leq a$$

> **证明**
>
> 1. （基础）当 $a=kd, b=d, k\in\mathbb{N}_+$ 时，$x'=0, y'=1$，显然成立。
> 2. （归纳）当 $|x'|\leq a \bmod b, |y'|\leq b$ 时，有
>
> $$|x|=|y'|\leq b, |y|\leq|x'|+\lfloor\frac{a}{b}\rfloor|y'|\leq(a \bmod b)+b\lfloor\frac{a}{b}\rfloor\leq a$$

```cpp
inline void exGcd(ll a, ll b, ll &x, ll &y, ll &g)
{
    if (!b)
    {
        x = 1;
        y = 0;
        g = a;
        return ;
    }
    exGcd(b, a % b, y, x, g);
    y -= a / b * x;
}
```

线性同余方程

- **定理1** 若 $(a,b)|c$，则二元一次方程 $ax+by=c(a,b,c\in\mathbb{N})$ 有无穷多解，否则方程不存在整数解。
- **定理2** 若二元一次方程 $ax+by=c(a,b,c\in\mathbb{N})$ 有解且特解为 $x=x_0, y=y_0$，那么方程的解可表示为 $x=x_0+\frac{b}{d}t, y=y_0-\frac{a}{d}t, t\in\mathbb{Z}$
- **定理3** n 元一次不定方程 $\sum\limits_{i=1}^{n}a_ix_i=c(a_i,c\in\mathbb{N})$ 有解的充要条件为 $(a_1,a_2,\ldots,a_n)|c$

解 n 元一次不定方程

- 解 n 元一次不定方程 $\sum\limits_{i=1}^{n}a_ix_i=c(a_i,c\in\mathbb{N})$。
  顺次求出 $(a_1,a_2)=d_2,(d_2,a_3)=d_3,\ldots,(d_{n-1},a_n)=d_n$。
  若 $d_n|c$，则方程有解，作方程组
  $$a_1x_1+a_2x_2=d_2t_2$$
  $$d_2t_2+a_3x_3=d_3t_3$$
  $$\cdots$$
  $$d_{n-1}t_{n-1}+a_nx_n=c$$
  求出最后一个方程的解，依次往上迭代。

解 n 元一次不定方程组

- 解 m 个 n 元一次不定方程组成的方程组，其中 $m < n$，由 $m-1$ 个不定方程，消去 $m-1$ 个未知数，将方程组转化为 $n-m+1$ 元的一次不定方程。

解一元线性同余方程

- 二元一次不定方程 $ax+by=c \Leftrightarrow$ 求一元线性同余方程 $ax \equiv c \pmod b$ 整数解，设 $d=\gcd(a,b)$，若 d 不能整除 c 则无解，否则在 $\bmod c$ 的意义下解有 d 个，解的形式为

$$x = x_0 + \frac{b}{d}t, t \in \mathbb{Z}$$

解一元线性同余方程组

算法1 合并法

- 以合并下面两个方程为例，

$$\begin{aligned} x &\equiv b_1 \pmod{m_1} \\ x &\equiv b_2 \pmod{m_2} \end{aligned}$$

  设 $x = b_1 + m_1 y_1 = b_2 + m_2 y_2$，即 $m_2 y_2 - m_1 y_1 = b_1 - b_2$
  解出 y_2，设 $m = [m_1, m_2]$，得到方程 $x \equiv b_2 + m_2 y_2 \pmod m$
  若有多个方程，依据上述过程直至消成单个方程即可。

```cpp
typedef __int128 ll;

inline ll calc_single(ll a, ll b, ll c)
{   // 返回 ax = c (mod b) 的特解
    ll x, y, e;
    exGcd(a, b, x, y, e);
    if (c % e != 0)
        EXIT();
    b /= e, c /= e;
    return ((x * c) % b + b) % b;
}

inline ll calc_multi(ll *m, ll *c, int xp)
{   // 返回方程组 x = ci(mod mi) (1 <= i <= xp) 在 [0,lcm(mi)) 内的特解
    ll lcm = m[1], res = c[1];
    for (int i = 2; i <= xp; ++i)
    {
        ll y = calc_single(lcm, m[i], ((c[i] - res) % m[i] + m[i]) % m[i]);
        res = lcm * y + res;
        lcm = lcm / std::__gcd(lcm, m[i]) * m[i];
        res = (res % lcm + lcm) % lcm;
    }
    return res;
}
```

算法2 中国剩余定理

- 设 $m_1, m_2, \ldots, m_r$ 为两两互素的正整数，则同余方程组

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\cdots \\ x &\equiv a_r \pmod{m_r} \end{aligned}$$

有在模 $M=\prod\limits_{i=1}^{r} m_i$ 意义下的唯一解，即中国剩余定理。

- 设 $M_i=\frac{M}{m_i}$，t_i 为 $M_it_i=1(\mod m_i)$ 的解，则唯一解为 $x=\left(\sum\limits_{i=1}^{r} a_it_iM_i\right) \mod M$

BSGS

- 若 $a^x\equiv b(\mod p)$ 的最小自然数 x。
- 若 $a\perp p$，
    - 由抽屉原理，只需找到在 $[0,p)$ 内的解，设 $t=\lceil\sqrt{p}\rceil$，则 $[0,p)$ 内的整数均可以用 $\{x=it-j,i\in[1,t],j\in(0,t]\}$。
    - 因此可将原方程化为 $a^{it}\equiv ba^j(\mod p)$，将右侧插入哈希表，左侧暴力枚举即可。
- 若 $a\not\perp p$，不断取 $d_n=(a,\frac{p}{\prod\limits_{j=1}^{n-1} d_j})$，直至 $(a,\frac{p}{\prod\limits_{j=1}^{n} d_j})=1$，设 $D=\prod\limits_{j=1}^{n} d_j$，取 $a'=\frac{a^n}{D},b'=\frac{b}{D},p'=\frac{p}{D}$，原方程可化为：

$$a^{x-n}\equiv b'(a')^{-1}(\mod p')$$

    - 暴力枚举 $x<n$ 的情况，$x\geq n$ 的情况用互质的做法解决。
- 若 $a\perp p$，只调用函数 `solve_BSGS` 即可，否则沿用整个程序。

```
#include <bits/stdc++.h>

const int Maxn = 1e9;
int a, p, b;

struct hashtable
{
    #define MOD 4999963
    #define H 5000005

    int adj[H], cst[H], to[H], nxt[H], stk[H];
    int top, T;
    bool vis[H];

    inline void Clear()
    {
        while (top)
        {
            int x = stk[top--];
            adj[x] = 0;
            vis[x] = false;
        }
        T = 0;
    }

    inline int Find(int y)
    {
        int x = y % MOD;
        for (int e = adj[x]; e; e = nxt[e])
            if (to[e] == y)
                return cst[e];
        return Maxn;
    }
```

```

    inline void Insert(int y, int z)
    {
        int x = y % MOD;
        if (!vis[x])
            vis[stk[++top] = x] = true;
        nxt[++T] = adj[x]; adj[x] = T; to[T] = y; cst[T] = z;
    }
}ht;

inline int quick_pow(int x, int k, int mod)
{
    int res = 1;
    while (k)
    {
        if (k & 1)
            res = 1ll * res * x % mod;
        x = 1ll * x * x % mod;
        k >>= 1;
    }
    return res;
}

inline int ex_gcd(int a, int b, int &x, int &y)
{
    if (!b)
    {
        x = 1;
        y = 0;
        return a;
    }
    int res = ex_gcd(b, a % b, y, x);
    y -= a / b * x;
    return res;
}

inline int solve_equ(int a, int b, int c)
{
    int x, y;
    int d = ex_gcd(a, b, x, y);
    if (c % d != 0) return -1;

    int mod = b / d;
    return (1LL * c / d * x % mod + mod) % mod;
}

inline int solve_BSGS(int a, int b, int p)
{
    int t = ceil(sqrt(p));
    ht.Clear();

    int tmp = b;
    for (int i = 1; i <= t; ++i)
    {
        tmp = 1LL * tmp * a % p;
        ht.Insert(tmp, i);
    }

```

```cpp
    int pw = quick_pow(a, t, p);
    tmp = pw;
    for (int i = 1; i <= t; ++i)
    {
        int res = ht.Find(tmp);
        if (res != Maxn)
            return i * t - res;
        tmp = 1LL * tmp * pw % p;
    }
    return -1;
}

inline bool check()
{
    int k = 1 % p;
    for (int i = 0; i <= 40; ++i)
    {
        if (k == b)
            return printf("%d\n", i), true;
        k = 1LL * k * a % p;
    }
    if (!a)
        return puts("No Solution"), true;
    return false;
}

int main()
{   // 求 a^x = b(mod p) 的最小自然数 x，无解输出 No Solution
    while (scanf("%d%d%d", &a, &p, &b), a || p || b)
    {
        a %= p, b %= p;
        if (check())
            continue;

        int d;
        int ap = 1, n = 0;
        bool flg = false;

        while ((d = std::__gcd(a, p)) != 1)
        {
            ++n;
            ap = 1LL * ap * (a / d) % p;
            p /= d;

            if (b % d)
            {
                flg = true;
                break;
            }
            b /= d;
        }

        if (flg)
            puts("No Solution");
        else
        {
            int res = solve_BSGS(a, 1LL * b * solve_equ(ap, p, 1) % p, p);
            if (res == -1)
```

```
                puts("No Solution");
            else
                printf("%d\n", res + n);
        }
    }
    return 0;
}
```

原根

阶

- 设 $n, a \in \mathbb{N}_+, n > 1, a \perp n$，则 $\exists 1 \le r \le n, a^r \equiv 1 \pmod n$，将最小的 r 称为 a 模 n 的阶，记作 $\mathrm{Ord}_n(a)$。
- 若 $a \perp n, a^N \equiv 1 \pmod n$，则 $\mathrm{Ord}_n(a) | N$。
- 若 $a \perp n$，$\mathrm{Ord}_n(a) | \phi(n)$。

原根

- 设 $a, n \in \mathbb{N}_+, n > 1, a \perp n$，若 $\mathrm{Ord}_n(a) = \phi(n)$，则称 a 为模 n 的一个原根。
- **剩余类** 所有模 n 同余的整数构成的集合，设余数为 r，则该剩余类简记为 $\overline{r}$。
- **剩余系** 模 n 所得的余数域。
- **简化剩余系** 若模 n 的一个剩余类内所有数都与 n 互素，就称其为与模 n 互素的剩余类，在与模 n 互素的全体剩余类中，从每一个类中任取一个数作为代表组成的集合，称为模 n 的一个简化剩余系，容易证明，简化剩余系关于模 n 乘法封闭。
- 记 $\delta = \mathrm{Ord}_n(a)$，则 $a^0, a^1, \dots, a^{\delta-1}$ 两两不同余，当 a 是 n 的原根时，$a^0, a^1, \dots, a^{\delta-1}$ 构成模 n 的简化剩余系，当 n 为质数时，$\{a^0, a^1, \dots, a^{\delta-1}\}$ 与 $\{1, 2, \dots, n-1\}$ 构成双射。
- 只有 $2, 4, p^k, 2p^k(p\text{为奇素数})$ 有原根。
- 若 n 存在原根，设 n 的最小原根为 g，则 $g^s \mod n (1 \le s \le \phi(n), s \perp \phi(n))$ 一定也是它的原根，因此 n 共有 $\phi(\phi(n))$ 个原根。

原根的求法

- 暴力枚举最小原根 g，满足 $g \perp n$，要验证 g 是否是 n 的原根。
- 根据 $\mathrm{Ord}_n(a) | \phi(n)$，我们只需验证对于每个 $d | \phi(n) (d \ne \phi(n))$，均有 $a^d \not\equiv 1 \pmod n$。
- 进一步地，设 $\phi(n) = \prod_{i=1}^{m} p_i^{c_i}$，我们只需对所有 p_i，验证 $a^{\frac{\phi(n)}{p_i}} \not\equiv 1 \pmod n$。
- 之后暴力枚举 s，满足 $s \perp \phi(n)$，就能求出所有的原根。

```
inline void findRoot(int x)
{
    int u = x, phi = x;
    for (int i = 2, im = sqrt(x); i <= im && i <= u; ++i)
        if (u % i == 0)
        {
            phi = phi / i * (i - 1);
            u /= i;
            while (u % i == 0)
                u /= i;
        }
    if (u > 1)
        phi = phi / u * (u - 1);
    cm = am = 0;
    u = phi;
    for (int i = 2, im = sqrt(phi); i <= im && i <= u; ++i)
```

```cpp
            if (u % i == 0)
            {
                cur[++cm] = i;
                u /= i;
                while (u % i == 0)
                    u /= i;
            }
    if (u > 1)
        cur[++cm] = u;
    int g;
    for (g = 1; g < x; ++g)
    {
        if (std::__gcd(g, x) > 1)
            continue ;
        bool flag = false;
        for (int i = 1; i <= cm; ++i)
            if (quick_pow(g, phi / cur[i], x) == 1)
            {
                flag = true;
                break ;
            }
        if (!flag)
            break ;
    }
    if (g == x)
        return ;
    int res = 1;
    for (int s = 1; s <= phi; ++s)
    {
        res = 1ll * res * g % x;
        if (std::__gcd(phi, s) > 1)
            continue ;
        ans[++am] = res;
    }
    std::sort(ans + 1, ans + am + 1);
}
```

N 次剩余

- 解方程 $x^N \equiv a \pmod p$，只考虑 p 为素数的情况，解出的 x 是模 p 意义下的剩余类。
- 令 g 为 p 的原根，则 $\{1,2,\ldots,p-1\}$ 可与 $\{g^1,g^2,\ldots,g^{p-1}\}$ 建立一一对应关系。
- 令 $g^y \equiv x \pmod p, g^t \equiv a \pmod p$，则原式可化为 $g^{Ny} \equiv g^t \pmod p$，在指数上等价于解线性同余方程 $Ny \equiv t \pmod{(p-1)}$，代回即可解出 x。

数论函数

- **积性函数** 函数 f 满足 $\forall a,b \in \mathbb{N}, a \perp b$，$f(ab)=f(a)f(b)$。
- **完全积性函数** 函数 f 满足 $\forall a,b \in \mathbb{N}$，$f(ab)=f(a)f(b)$。

常见积性函数

欧拉函数

- 记作 $\varphi(n)$，指不超过 n 且与 n 互素的正整数个数，由中国剩余定理可知其为积性函数。
- **定理1** 设 $n=\prod_{i=1}^{m} p_i^{c_i}$，则 $\varphi(n)=\prod_{i=1}^{m}(p_i-1)p_i^{c_i-1}=\prod_{i=1}^{m}(1-\frac{1}{p_i})$。
  - **推论1** 当 n 为奇数时，$\varphi(2n)=\varphi(n)$。
  - **推论2** 设 n 为一个大于 2 的正整数，那么 $\varphi(n)$ 是偶数。
- **定理2** 当 $n>1$ 时，$[1,n]$ 中与 n 互质的整数的和为 $\frac{n\varphi(n)}{2}$。
- **欧拉定理** 对于任意两个互质的正整数 $a,m(m\geq 2)$，有 $a^{\varphi(m)}\equiv 1(\mod m)$，故 $a^{\varphi(m)-1}$ 为 a 在模 m 意义下的逆元。
  - **推论（费马小定理）** 设 m 为质数，$a^{m-1}\equiv 1(\mod m)$，故 a^{m-2} 为 a 在模 m 意义下的逆元。

> **证明**
>
> - $\forall b,c$，$ab\equiv ac(\mod m)\Leftrightarrow a(b-c)\equiv 0(\mod m)$，因为 $a\perp m$，$b\equiv c(\mod m)$，故当 $b\not\equiv c(\mod m)$ 时，$\overline{ab},\overline{ac}$ 也表示不同的剩余类。
> - 设 m 的简化剩余系为 $\{\overline{a_1},\overline{a_2},\dots,\overline{a_{\varphi(m)}}\}$，由其模 m 乘法封闭的性质以及上述结论，可以推知 $\{\overline{aa_1},\overline{aa_2},\dots\overline{aa_{\varphi(m)}}\}$ 也能表示 m 的简化剩余系，故：
>
> $$a^{\varphi(m)}a_1a_2\dots a_{\varphi(m)}\equiv(aa_1)(aa_2)\dots(aa_{\varphi(m)})\equiv a_1a_2\dots a_{\varphi(m)}(\mod m)$$
>
> - 由简化剩余系的定义可知 $a^{\varphi(m)}\equiv 1(\mod m)$，证毕。

- **扩展欧拉定理** 对于任意正整数 a,b,m，
  - 若 a,m 不互质，且 $b<\varphi(m)$，不能应用该定理进行降幂。
  - 若 a,m 不互质，且 $b\geq\varphi(m)$，则 $a^b\equiv a^{b\mod\varphi(m)+\varphi(m)}(\mod m)$。
  - 若 a,m 互质，则 $a^b\equiv a^{b\mod\phi(m)}(\mod m)$。

莫比乌斯函数

- 记作 $\mu(n)$，若 n 有平方数因子，则 $\mu(n)=0$，否则 n 为 k 个不同质数的乘积，则 $\mu(n)=(-1)^k$。特别地 $\mu(1)=1$。

除数函数

- $\sigma_k(n)$ 表示 n 所有正因子的 k 次幂之和。
- $d(n)=\sigma_0(n)$ 表示 n 的正因子个数，$\sigma(n)=\sigma_1(n)$ 表示 n 的所有正因子之和。

幂函数

- $Id_k(n)=n^k,1(n)=Id_0(n)=1,Id(n)=Id_1(n)=1$。

单位函数

- $\varepsilon(n)=[n=1]$。

Dirichlet 卷积

- 定义数论函数 f,g 的 Dirichlet 卷积为 $(f*g)(n)=\sum_{d|n}f(d)g(\frac{n}{d})$，满足交换律、结合律、分配律，且有单位元 ε。
- 若 f,g 均为积性函数，则 $f*g$ 也为积性函数。
- 常见的 Dirichlet 卷积：

$$d=1*1,\sigma=Id*1,\varphi=\mu*Id,\varepsilon=\mu*1$$

- 由于 $Id = 1 * \mu * Id = 1 * \varphi$，即 $n = \sum_{d|n} \varphi(d)$。

莫比乌斯反演

- 对于两个函数 f, g:

$$f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(\frac{n}{d}) f(d)$$

> **证明**
>
> $$\begin{aligned} &\because f = 1 * g \Rightarrow 1 * \mu * g = \mu * f \Rightarrow g = \mu * f \\ &\quad g = \mu * f \Rightarrow 1 * g = 1 * \mu * f \Rightarrow f = 1 * g \\ &\therefore f = 1 * g \Leftrightarrow g = \mu * f \end{aligned}$$

- 另一种形式:

$$f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} f(d) \mu(\frac{d}{n})$$

> **证明**
>
> $$g(n) = \sum_{n|d} f(d) \mu(\frac{d}{n}) = \sum_{n|d} \mu(\frac{d}{n}) \sum_{d|t} g(t) = \sum_{n|t} g(t) \sum_{d|\frac{t}{n}} \mu(d) = \sum_{n|t} g(t)[n = t] = g(n)$$
>
> $$f(n) = \sum_{n|d} g(d) = \sum_{n|d} \sum_{d|t} f(t) \mu(\frac{t}{d}) = \sum_{n|t} f(t) \sum_{d|\frac{t}{n}} \mu(d) = \sum_{n|t} f(t)[n = t] = f(n)$$

线性筛

- 线性筛素数的流程如下:
  - 设 low_i 表示 i 的最小质因子。
  - 考虑从 2 到 n 枚举 i，若 low_i 还未被计算出来，则 i 为质数，$low_i = i$。
  - 枚举所有不超过 low_i 的质数 p，令 $low_{ip} = p$。
- 在线性筛中，我们能得到每个数 n 的最小质因子 p 以及它的次数 k，则 $f(n) = f(p^k) f(\frac{n}{p^k})$，若 $f(p^k)$ 可以快速求出，就能在线性筛的同时快速求出 $f(n)$，有时也可利用积性函数本身的性质简化计算。
- 以 $\mu(n), \varphi(n), d(n)$ 的筛法为例。

```
inline void sieve(int lim)
{
    d[1] = phi[1] = mu[1] = 1;
    for (int i = 2; i <= lim; ++i)
    {
        if (!low[i])
        {
            pri[++pn] = low[i] = i;
            phi[i] = i - 1;
            mu[i] = -1;
            mx_d[i] = 1; // i 最小质因子的次数
            mx_i[i] = i; // i 最小质因子的乘幂
            d[i] = 2;
        }
        for (int j = 1; j <= pn && 1ll * i * pri[j] <= lim; ++j)
        {
```

```
            int k = pri[j] * i;
            low[k] = pri[j];
            if (low[i] == pri[j])
            {
                phi[k] = phi[i] * pri[j];
                mu[k] = 0;
                mx_d[k] = mx_d[i] + 1;
                mx_i[k] = mx_i[i] * pri[j];
                d[k] = d[k / mx_i[k]] * (mx_d[k] + 1);
                break ;
            }
            mu[k] = -mu[i];
            phi[k] = phi[i] * (pri[j] - 1);
            mx_d[k] = 1;
            mx_i[k] = pri[j];
            d[k] = d[i] * 2;
        }
    }
}
```

整除分块

- **性质1** $\lfloor \frac{\lfloor \frac{a}{b} \rfloor}{c} \rfloor = \lfloor \frac{a}{bc} \rfloor$。
- 易证，$\lfloor \frac{n}{d} \rfloor$ 的取值只有至多 $2\sqrt{n}$ 种，且 $\lfloor \frac{n}{d} \rfloor = \lfloor \frac{n}{x} \rfloor$ 最大的 x 即满足 $\lfloor \frac{n}{d} \rfloor x \le n, x = \lfloor \frac{n}{\lfloor \frac{n}{d} \rfloor} \rfloor$。
- n, m 两维的整除分块枚举如下：

```
    for (int i = 1, x, im = Min(n, m); i <= im; i = x + 1)
    {
        x = Min(n / (n / i), m / (m / i));
        ...
    }
```

- **性质2** $\lceil \frac{x+y}{P} \rceil = \lfloor \frac{x}{P} \rfloor + \lfloor \frac{y}{P} \rfloor + [x \bmod P + y \bmod P \ge P]$
- **性质3** $\lceil \frac{x}{P} \rceil = \lfloor \frac{x+P-1}{P} \rfloor = \lfloor \frac{x-1}{P} \rfloor + 1$

常见模型

- $1 \le n, m \le 10^7$，询问组数 $T \le 10^4$：

$$\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=1}^{m}(i,j)\\
=&\sum_{t=1}^{\min\{n,m\}}t\sum_{i=1}^{n}\sum_{j=1}^{m}[(i,j)=t]\\
=&\sum_{t=1}^{\min\{n,m\}}t\sum_{i=1}^{\lfloor\frac{n}{t}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{t}\rfloor}[(i,j)=1]\\
=&\sum_{t=1}^{\min\{n,m\}}t\sum_{i=1}^{\lfloor\frac{n}{t}\rfloor}\sum_{j=1}^{\lfloor\frac{m}{t}\rfloor}\sum_{\substack{d|i\\d|j}}\mu(d)\\
=&\sum_{t=1}^{\min\{n,m\}}t\sum_{d}\mu(d)\lfloor\frac{n}{td}\rfloor\lfloor\frac{m}{td}\rfloor\\
=&\sum_{T=td=1}^{\min\{n,m\}}\sum_{t|T}t\mu(\frac{T}{t})\lfloor\frac{n}{T}\rfloor\lfloor\frac{m}{T}\rfloor\\
=&\sum_{T=1}^{\min\{n,m\}}\varphi(T)\lfloor\frac{n}{T}\rfloor\lfloor\frac{m}{T}\rfloor
\end{aligned}$$

- 预处理出 φ 的前缀和，整除分块即可 ，类似上述推导可以直接得到以下代换式：

$$(i,j)=\sum_{d|i,d|j}\varphi(d)$$

- **结论** $d(ij)=\sum_{x|i}\sum_{y|j}[(x,y)=1]$。
    - **推论** $d(ijk)=\sum_{x|i}\sum_{y|j}\sum_{z|k}[(x,y)=1][(y,z)=1][(x,z)=1]$

> **证明** 考虑对于 ij 中的每一个质因数 p，其在 ij 中的次数为 b，在 i 中的次数为 a。
>
> 对于 ij 的任意一个约数 d，设 p 在 d 中的次数为 c，初始时令 $x=y=1$。
>
> - 若 $c\le a$，令 x 乘上 p^c。
> - 若 $a<c\le b$，令 y 乘上 p^{c-a}。
>
> 则对于任意一个约数 d，我们都能构造出唯一的一组互质的 x,y 与之对应，原命题得证。

典例 SDOI2018 旧试题

题目大意

- 求 $\left(\sum_{i=1}^{A}\sum_{j=1}^{B}\sum_{k=1}^{C}d(ijk)\right)\mod(10^9+7)$，$A,B,C\le 10^5$。

解法

$$
\begin{aligned}
\sum_{i=1}^{A}\sum_{j=1}^{B}\sum_{k=1}^{C} d(ijk) &= \sum_{i=1}^{A}\sum_{j=1}^{B}\sum_{k=1}^{C}\sum_{x|i}\sum_{y|j}\sum_{z|k}[(x,y)=1][(y,z)=1][(x,z)=1] \\
&= \sum_{x=1}^{A}\sum_{y=1}^{B}\sum_{z=1}^{C}[(x,y)=1][(y,z)=1][(x,z)=1]\lfloor\frac{A}{x}\rfloor\lfloor\frac{B}{y}\rfloor\lfloor\frac{C}{z}\rfloor \\
&= \sum_{x=1}^{A}\sum_{y=1}^{B}\sum_{z=1}^{C}[(x,y)=1][(y,z)=1][(x,z)=1]\lfloor\frac{A}{x}\rfloor\lfloor\frac{B}{y}\rfloor\lfloor\frac{C}{z}\rfloor \\
&= \sum_{x=1}^{A}\sum_{y=1}^{B}\sum_{z=1}^{C}\sum_{a|x,a|y}\mu(a)\sum_{b|y,b|z}\mu(b)\sum_{c|x,c|z}\mu(c)\lfloor\frac{A}{x}\rfloor\lfloor\frac{B}{y}\rfloor\lfloor\frac{C}{z}\rfloor \\
&= \sum_{a=1}^{A}\sum_{b=1}^{B}\sum_{c=1}^{C}\mu(a)\mu(b)\mu(c)\sum_{x|\mathrm{lcm}(a,c)}\lfloor\frac{A}{x}\rfloor\sum_{y|\mathrm{lcm}(a,b)}\lfloor\frac{B}{y}\rfloor\sum_{z|\mathrm{lcm}(b,c)}\lfloor\frac{C}{z}\rfloor
\end{aligned}
$$

- 将 a,b,c 看作图中的点，点数最多为 $\max\{A,B,C\}$，对于图中任意两点 u,v，若 $\mu(u)\mu(v)\neq 0$ 且 $\mathrm{lcm}(u,v)\leq\max\{A,B,C\}$ 时连边 (u,v)，转化为三元环计数。

杜教筛

- 设 $S(n)=\sum\limits_{i=1}^{n} f(i)$，则有:

$$
\sum_{i=1}^{n}(f*g)(i)=\sum_{i=1}^{n}\sum_{d|i}g(d)f\left(\frac{i}{d}\right)=\sum_{d=1}^{n}g(d)S\left(\lfloor\frac{n}{d}\rfloor\right)
$$
$$
g(1)S(n)=\sum_{i=1}^{n}(f*g)(i)-\sum_{d=2}^{n}g(d)S\left(\lfloor\frac{n}{d}\rfloor\right)
$$

- 若 g 的单点值和 $\sum\limits_{i=1}^{n}(f*g)(i)$ 都可以快速计算，则可通过数论分块计算 $S(n)$。
- 设 $T(n)$ 为计算 $S(n)$ 的时间复杂度，则:

$$
T(n)=\mathcal{O}(\sqrt{n})+\mathcal{O}\left(\sum_{i=2}^{\sqrt{n}}\left(T(i)+T\left(\lfloor\frac{n}{i}\rfloor\right)\right)\right)
$$

- 更深层的展开是高阶无穷小，可直接略去，因而有:

$$
T(n)=\mathcal{O}(\sqrt{n})+\sum_{i=2}^{\sqrt{n}}\left(\mathcal{O}(\sqrt{i})+\mathcal{O}\left(\sqrt{\lfloor\frac{n}{i}\rfloor}\right)\right)=\mathcal{O}(\sqrt{n})+\mathcal{O}\left(\int_{2}^{\sqrt{n}}\sqrt{x}\mathrm{d}x\right)+\mathcal{O}\left(\int_{2}^{\sqrt{n}}\sqrt{\frac{n}{x}}\mathrm{d}x\right)=\mathcal{O}(n^{\frac{3}{4}})
$$

- 进一步地，若能 $\mathcal{O}(k)$ 预处理 $S(1)$ 到 $S(k)$ 的值，则 $T(n)$ 的计算式变为（假设 $k>\sqrt{n}$）：

$$
T(n)=\mathcal{O}(\sqrt{n})+\sum_{i=2}^{\lfloor\frac{n}{k}\rfloor}\mathcal{O}\left(\sqrt{\lfloor\frac{n}{i}\rfloor}\right)=\mathcal{O}(\sqrt{n})+\mathcal{O}\left(\int_{2}^{\frac{n}{k}}\sqrt{\frac{n}{x}}\mathrm{d}x\right)=\mathcal{O}(\frac{n}{\sqrt{k}})
$$

- 取 $k=n^{\frac{2}{3}}$，总时间复杂度 $\mathcal{O}(n^{\frac{2}{3}})$。
- 常见前缀和模型:

$$
S_\mu(n)=\sum_{i=1}^{n}\mu(i)\ ,\varepsilon=\mu*1\Rightarrow\ S_\mu(n)=1-\sum_{i=2}^{n}S_\mu\left(\lfloor\frac{n}{i}\rfloor\right)
$$
$$
S_\varphi(n)=\sum_{i=1}^{n}\varphi(i),Id=\varphi*1\Rightarrow S_\varphi(n)=\frac{n(n+1)}{2}-\sum_{i=2}^{n}S_\varphi\left(\lfloor\frac{n}{i}\rfloor\right)
$$
$$
S_0(n)=\sum_{i=1}^{n}i^2\varphi(i),Id^3=(Id^2\varphi)*Id^2\Rightarrow S_0(n)=\left(\frac{n(n+1)}{2}\right)^2-\sum_{i=2}^{n}i^2S_0\left(\lfloor\frac{n}{i}\rfloor\right)
$$

- 若需同时求 $S_\mu(n)$ 和 $S_\varphi(n)$，根据下式可将其转化为求 $S_\mu(n)$，时间复杂度分析类似，也为 $\mathcal{O}(n^{\frac{2}{3}})$，但单求 $S_\varphi(n)$ 常数更大。

$$\sum_{i=1}^{n}\sum_{j=1}^{n}[(i,j)=1]=\sum_{d=1}^{n}\mu(d)\lfloor\frac{n}{d}\rfloor^2=2S_\varphi(n)-1$$

- 以下为单求 $S_\varphi(n)$ 时的模板。

```
struct hashtable
{
    #define MOD 999979
    #define H 1000005
    int adj[H], to[H], nxt[H], stk[H];
    int top, T; ll cst[H];
    bool vis[H];

    inline void Clear()
    {
        while (top)
        {
            int x = stk[top--];
            adj[x] = 0;
            vis[x] = false;
        }
        T = 0;
    }

    inline ll Find(int y)
    {
        int x = y % MOD;
        for (int e = adj[x]; e; e = nxt[e])
            if (to[e] == y)
                return cst[e];
        return Maxn;
    }

    inline void Insert(int y, ll z)
    {
        int x = y % MOD;
        if (!vis[x])
            vis[stk[++top] = x] = true;
        nxt[++T] = adj[x]; adj[x] = T; to[T] = y; cst[T] = z;
    }
}phiH;

inline ll calcPhi(int n)
{
    if (n <= lim) // lim 取 max{n}^{2/3}
        return phi[n];
    ll res = phiH.Find(n);
    if (res != Maxn)
        return res;
    res = 1ll * n * (n + 1) / 2;
    for (ll i = 2, x; i <= n; i = x + 1)
    {
        x = n / (n / i);
        res -= (x - i + 1) * calcPhi(n / i);
```

```
    }
    phiH.Insert(n, res);
    return res;
}
```

Min_25 筛

- 当 $n < 10^{13}$ 时，可在 $\mathcal{O}\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$ 解决一类**积性函数** $f(n)$ 前缀和的问题。
- 要求（以下 p 均表示任意质数）：
  - $f(p)$ 可以被拆成常数个完全积性函数（例如关于 p 的低次多项式）。
  - $f(p^c)$ 可以快速计算。

记号

- $\text{isprime}(n) = [n\text{是质数}]$。
- p_k：全体质数中第 k 小的质数，特别地，令 $p_0 = 1$。
- low_n：n 的最小质因子。
- $F_{\text{prime}}(n) = \sum\limits_{2\le p\le n} f(p)$。
- $F_k(n) = \sum\limits_{i=2}^{n} [\text{low}_i \ge p_k] f(i)$。

算法流程

- 不难发现，所求即为 $F_1(n) + f(1) = F_1(n) + 1$。
- 关于计算 $F_k(n)$，有以下递推式（若设 $c_0 = \max\limits_{p_i^c \le n}\{c\}$，则 $\left\lfloor \frac{n}{p_i^{c_0}} \right\rfloor < p_i$，$F_{i+1}\left(\left\lfloor \frac{n}{p_i^{c_0}} \right\rfloor\right) = 0$ ）：

$$F_k(n) = F_{\text{prime}}(n) - F_{\text{prime}}(p_{k-1}) + \sum_{\substack{i\ge k \\ p_i^2 \le n}} \sum_{\substack{c\ge 1 \\ p_i^{c+1}\le n}} \left(f(p_i^c) F_{i+1}\left(\left\lfloor \frac{n}{p_i^c} \right\rfloor\right) + f(p_i^{c+1}) \right)$$

- 边界情况为当 $p_k > n$ 时，$F_k(n) = 0$。
- 若能预处理 $F_{\text{prime}}(n)$，按照上述求和式暴力计算即可，一般情况的时间复杂度为 $\mathcal{O}(n^{1-\epsilon})$，当 $n < 10^{13}$ 时，可以证明时间复杂度只有 $\mathcal{O}\left(\frac{n^{\frac{3}{4}}}{\log n}\right)$ 级别，具体参见朱震霆的候选队论文。
- 考虑怎样预处理 $F_{\text{prime}}(x)$，注意需要计算的 x 都可以被表示为 $\left\lfloor \frac{n}{d} \right\rfloor$ 的形式（包括 p_{k-1}，因为 $p_{k-1} \le \sqrt{n}$，总存在 $\left\lfloor \frac{n}{\left\lfloor \frac{n}{p_{k-1}} \right\rfloor} \right\rfloor = p_{k-1}$），由整除分块，$x$ 只有 $\mathcal{O}(\sqrt{n})$ 种，可以根据 x 是否大于 $\sqrt{n}$ 分别编号，空间复杂度 $\mathcal{O}(\sqrt{n})$。
- 由于 $f(p)$ 是关于 p 的低次多项式，不妨设 $f(p) = \sum\limits_{i=1}^{m} a_i p^{c_i}$，则 $F_{\text{prime}}(x) = \sum\limits_{i=1}^{m} a_i \sum\limits_{2\le p\le n} p^{c_i}$。
- 对于每个 i，我们设 **完全积性函数** $g(p) = p^{c_i}$，需对其快速求前缀和。
- 再设 $G_k(n) = \sum\limits_{i=1}^{n} [\text{low}_i > p_k \lor \text{isprime}(i)] g(i)$，即埃氏筛 p_k 筛完后未被标记的 $g(i)$ 之和，只需求出 $G_{k_0}(n)$ 满足 $k_0 = \max\limits_{p_k \le \sqrt{n}}\{k\}$，边界情况为 $G_0(n) = \sum\limits_{i=2}^{n} g(i)$，可直接通过通项计算，否则有以下递推式：

$$G_k(n) = G_{k-1}(n) - g(p_k)\left(G_{k-1}\left(\left\lfloor \frac{n}{p_k} \right\rfloor\right) - G_{k-1}(p_{k-1})\right)$$

- $G_{k-1}(p_{k-1})$ 显然可以预处理，同样忽略高阶无穷小，对于每个 $m = \left\lfloor \frac{n}{d} \right\rfloor$，只有 $p_k^2 \le m$ 的 p_k 才会产生贡献，该部分的时间复杂度可估计为：

$$
\begin{aligned}
T(n) &= \sum_{i^2 \le n} \left(\mathcal{O}(\pi(\sqrt{i})) + O\left(\pi\left(\left\lfloor \frac{n}{i} \right\rfloor \right) \right) \right) \\
&= \mathcal{O}\left(\int_2^{\sqrt{n}} \frac{\sqrt{x}}{\ln \sqrt{x}} \mathrm{d}x \right) + \mathcal{O}\left(\int_2^{\sqrt{n}} \frac{\sqrt{\left\lfloor \frac{n}{x} \right\rfloor}}{\ln \sqrt{\left\lfloor \frac{n}{x} \right\rfloor}} \mathrm{d}x \right) = \mathcal{O}\left(\frac{n^{\frac{3}{4}}}{\log n} \right)
\end{aligned}
$$

模板

- $f(p^k) = p^k(p^k - 1)$，拆成 $g_1(p) = p^2, g_2(p) = p$ 作差即可。
- 注意数组大小至少要比 $2\sqrt{n}$ 稍大一点。
- $G_k(n)$ 的递推式也可以用来求仅与素数有关的完全积性函数（例如区间素数个数）。

```cpp
#include <bits/stdc++.h>

using std::ios;
using std::cin;
using std::cout;

const int N = 3e5 + 5;
const int mod = 1e9 + 7;
const int inv2 = 500000004;
const int inv6 = 166666668;
typedef long long ll;
ll n, sta[N];
int Sn, sm, pr;
int low[N], idx1[N], idx2[N], g[N], h[N], pri[N], sumg[N], sumh[N];

inline void add(int &x, int y) {x += y; x >= mod ? x -= mod : 0;}
inline void dec(int &x, int y) {x -= y; x < 0 ? x += mod : 0;}
inline int askid(ll x) {return x <= Sn ? idx1[x] : idx2[n / x];}
inline int askf(ll x) {x %= mod; return 1ll * x * (x + mod - 1) % mod;}
inline int asksumg(ll x) {x %= mod; return 1ll * x * (x + 1) % mod * (x << 1 | 1) % mod * inv6 % mod;}
inline int asksumh(ll x) {x %= mod; return 1ll * x * (x + 1) % mod * inv2 % mod;}
inline int askg(int p) {return 1ll * p * p % mod;}
inline int askh(int p) {return p;}

inline void sieve(int lim)
{
    for (int i = 2; i <= Sn; ++i)
    {
        if (!low[i])
        {
            pri[++pr] = i;
            low[i] = i;
            add(sumg[pr] = sumg[pr - 1], askg(i));
            add(sumh[pr] = sumh[pr - 1], askh(i));
        }
        for (int j = 1; j <= pr && 1ll * pri[j] * i <= Sn; ++j)
        {
            int tmp = pri[j] * i;
```

```cpp
            low[tmp] = pri[j];
            if (low[i] == pri[j])
                break;
        }
    }
}

inline void calcG()
{
    for (ll i = 1, j; i <= n; i = j + 1)
    {
        ll v = n / i;
        j = n / v;
        sta[++sm] = v;
        g[sm] = asksumg(v); dec(g[sm], 1);
        h[sm] = asksumh(v); dec(h[sm], 1);
        (sta[sm] <= Sn ? idx1[sta[sm]] : idx2[j]) = sm;
    }

    for (int j = 1; j <= pr; ++j)
    {
        ll pri2 = 1ll * pri[j] * pri[j];
        for (int i = 1; i <= sm && pri2 <= sta[i]; ++i)
        {
            ll tmp = sta[i] / pri[j];
            int x = askid(tmp);
            g[i] = (1ll * g[i] + mod - 1ll * askg(pri[j]) * (g[x] + mod - sumg[j - 1]) % mod) % mod;
            h[i] = (1ll * h[i] + mod - 1ll * askh(pri[j]) * (h[x] + mod - sumh[j - 1]) % mod) % mod;
        }
    }
}

inline int calcF(ll s, int k)
{
    if (s <= 1 || pri[k] > s)
        return 0;
    int x = s <= Sn ? idx1[s] : idx2[n / s];
    int res = g[x]; dec(res, sumg[k - 1]);
    dec(res, h[x]); add(res, sumh[k - 1]);
    for (int i = k; i <= pr && 1ll * pri[i] * pri[i] <= s; ++i)
        for (ll t1 = pri[i], t2 = 1ll * pri[i] * pri[i]; t2 <= s; t1 = t2, t2 *= pri[i])
            res = (1ll * askf(t1) * calcF(s / t1, i + 1) + askf(t2) + res) % mod;
    return res;
}

int main()
{
    std::cin >> n;
    Sn = sqrt(n);
    sieve(Sn);
    calcG();
    int ans = calcF(n, 1);
    add(ans, 1);
    cout << ans << '\n';
```

```
}
```

质因数分解

Miller Rabin 素性测试

- 根据费马小定理，我们能够得出一种检验素数的思路（即**费马素性测试**）：
    - 验证 $\forall 1 < a < n, a^{n-1} \equiv 1(\mod n)$。
- 很遗憾，上述做法并不能准确地判断素数，对于满足上述条件的合数，我们称之为**卡迈克尔数。**
- Miller Rabin 素性测试是基于费马素性测试的优化。
- **二次探测定理** 若 P 为奇素数，则 $x^2 \equiv 1 \Leftrightarrow x \equiv 1(\mod P)$或 $x \equiv P-1(\mod P)$。
- 令 $n-1 = u \times 2^t$，随机取一个 a，先求出 $a^u(\mod n)$，对该数平方 t 次，若在这一过程中出现在模 n 意义下为 1 或为 $n-1$，可认为通过了此轮测试。
- 经过理论证明，单轮错误率大约 $\frac{1}{4}$，T 轮测试的错误率为 4^{-T}，一般取 $T = 8 \sim 12$ 即可。
- 若取 $a = 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37$（即前 12 个质数）可保证 $n < 2^{64}$ 确定性判素。

生日悖论

- 求 k 个 $[1, n]$ 的随机整数两两不相同的概率，设该事件为 A，则

$$P(A) = \prod_{i=1}^{k} \frac{n-i+1}{n} = \prod_{i=1}^{k} (1 - \frac{i-1}{n})$$

- n 较大时，根据近似 $1 - \frac{i-1}{n} \approx (1 - \frac{1}{n})^{i-1}$，且 $e \approx (1 + \frac{1}{n})^n$，原式可化为

$$P(A) \approx \prod_{i=1}^{k} (1 - \frac{1}{n})^{i-1} = (1 - \frac{1}{n})^{\frac{k(k-1)}{2}} \approx e^{-\frac{k(k-1)}{2n}}$$

- 观察该式可知，当 k 取到 $\mathcal{O}(\sqrt{n})$ 级别时，$P(A)$ 会发生骤降。
- 因此 k 个 $[1, n]$ 的随机整数首次出现相同数字时 k 的期望为 $\mathcal{O}(\sqrt{n})$。

Pollard Rho 算法

- Pollard Rho 算法是一种基于随机的质因数分解算法，可以以期望时间复杂度 $\mathcal{O}(n^{\frac{1}{4}})$ 找到合数 n 的某个非平凡因子。
- 考虑选取适当的 k 使得 $1 < d = \gcd(k, n) < n$，则显然 d 是 n 的一个约数，这样的 k 相对较多。
- 若 n 本身是质数，可直接用 Miller Rabin 素性测试判定。否则构造伪随机数列 $x_{i+1} = (x_i^2 + c) \mod n$，其中 c 为初始时指定的某一随机数，设 m 为 n 的最小非平凡因子，令 $y_i = x_i \mod m$，则由生日悖论，满足 $\exists i < j, y_i = y_j$ 所需的序列 x 的期望长度为 $\mathcal{O}(\sqrt{m}) \le \mathcal{O}(n^{\frac{1}{4}})$，期望枚举 $\mathcal{O}(\sqrt{m})$ 个 i 我们便能得到 n 的一个约数 $\gcd(|x_i - x_j|, n)$。
- 可实际情况并不允许我们去暴力枚举所有 i, j，考虑到 x_i 的周期性，我们采用一种基于倍增的实现方式，即正序枚举 $t \ge 0$，每次检查 $\gcd(|x_i - x_{i+k}|, n)(1 \le k \le 2^t)$ 是否满足条件。
- 注意到 $\gcd(ab \mod n, n) = \gcd(ab, n) \ge \gcd(a, n), 1 \le a, b < n$，实际实现时我们并不需要每次都暴力求 gcd，而可以将若干次检查合为一次，以减小时间常数。
- 上述步骤许多部分基于估计，实际上并不严谨，但 Pollard Rho 算法在实际环境中运行得相当不错。

```
using std::vector;
typedef long long ll;
typedef __int128 s128;
```

```cpp

template <class T>
inline T Abs(T x) {return x < 0 ? -x : x;}
template <class T>
inline void CkMax(T &x, T y) {x < y ? x = y : 0;}

inline ll quick_pow(ll x, ll k, ll mod)
{
    ll res = 1;
    while (k)
    {
        if (k & 1)
            res = (s128)res * x % mod;
        x = (s128)x * x % mod;
        k >>= 1;
    }
    return res;
}

const int pri[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

inline bool millerRabin(ll n)
{
    if (n < 3 || !(n & 1))
        return n == 2;
    ll u = n - 1;
    int t = 0;
    while (!(u & 1))
        u >>= 1, ++t;
    for (int i = 0; i < 12; ++i)
    {
        if (pri[i] >= n)
            break ;
        ll x = pri[i];
        ll res = quick_pow(pri[i], u, n);
        if (res == 1)
            continue ;
        int j = 0;
        for (j = 0; j < t; ++j)
        {
            if (res == n - 1)
                break ;
            res = (s128)res * res % n;
        }
        if (j >= t)
            return false;
    }
    return true;
}

inline ll f(ll x, ll c, ll n)
{
    return ((s128)x * x + c) % n;
}

inline ll pollardRho(ll n)
{
    ll rx = 0, x = 0, d, val = 1;
```

```
    ll c = rand() % (n - 1) + 1;
    for (int t = 1; ; t <<= 1, rx = x, val = 1)
    {
        for (int k = 1; k <= t; ++k)
        {
            x = f(x, c, n);
            val = (s128)val * Abs(x - rx) % n;
            if ((k % 127) == 0)
            {
                d = std::__gcd(val, n);
                if (d > 1)
                    return d;
            }
        }
        ll d = std::__gcd(val, n);
        if (d > 1)
            return d;
    }
}

inline void Factor(ll n, vector<ll> &fac, int cnt)
{   // fac 存储所有质因子，顺序混乱
    if (n == 1)
       return ;
    if (millerRabin(n))
    {
        while (cnt--)
            fac.push_back(n);
        return ;
    }
    ll p = pollardRho(n); // 求出 n 的某个非平凡正因子
    int _cnt = 0;
    while (n % p == 0)
        n /= p, ++_cnt;
    Factor(n, fac, cnt);
    Factor(p, fac, cnt * _cnt);
}
```

类欧几里得

- 视 a, b, c, n 同阶，设

$$f(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor$$
$$g(a,b,c,n) = \sum_{i=0}^{n} i \lfloor \frac{ai+b}{c} \rfloor$$
$$h(a,b,c,n) = \sum_{i=9}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$$

- 其中 f 可以独立计算，而 g, h 的计算则会导致 f, g, h 的交错递归，故将三项合并计算，时间复杂度 $\mathcal{O}(\log n)$。

```
inline void add(ll &x, ll y)
{
    x += y;
    x >= P ? x -= P : 0;
```

```
}

struct data
{
    data() {f = g = h = 0;}
    ll f, g, h;
};

data calc(ll n, ll a, ll b, ll c)
{
    ll ac = a / c, bc = b / c, m = (a * n + b) / c, n1 = n + 1, n21 = n << 1 | 1;
    data d;
    if (a == 0)
    {
        d.f = bc * n1 % P;
        d.g = bc * n % P * n1 % P * i2 % P;
        d.h = bc * bc % P * n1 % P;
        return d;
    }
    if (a >= c || b >= c)
    {
        d.f = (n * n1 % P * i2 % P * ac + bc * n1) % P;
        d.g = (ac * n % P * n1 % P * n21 % P * i6 + bc * n % P * n1 % P * i2) % P;
        d.h = (ac * ac % P * n % P * n1 % P * n21 % P * i6 +
               bc * bc % P * n1 + ac * bc % P * n % P * n1) % P;
        data e = calc(n, a % c, b % c, c);
        add(d.f, e.f), add(d.g, e.g);
        d.h = (d.h + e.h + 2ll * bc % P * e.f + 2ll * ac % P * e.g) % P;
        return d;
    }
    data e = calc(m - 1, c, c - b - 1, a);
    d.f = (n * m + P - e.f) % P;
    d.g = (m * n % P * n1 + P - e.h + P - e.f) % P * i2 % P;
    d.h = (n * m % P * (m + 1) + (P - e.g << 1) + (P - e.f << 1) + P - d.f) % P;
    return d;
}
```

Lucas 定理

- 用于快速计算 $\binom{n}{m} \bmod p$
- 若 p 为质数，预处理阶乘 $\mathcal{O}(p)$，单次询问 $\mathcal{O}(\log n)$。

```
inline int C(int n, int m)
{   //fac 和 ifac 预处理到 [0, mod)
    if (n < 0 || m < 0 || n < m)
        return 0;
    return 1ll * fac[n] * ifac[n - m] % mod * ifac[m] % mod;
}

inline int Lucas(int x, int y)
{   //调用该函数返回结果
    if (!y) return 1;
    return 1ll * C(x % mod, y % mod) * Lucas(x / mod, y / mod) % mod;
}
```

- 若 p 不为质数，将其质因数分解，求出 $\binom{n}{m} \bmod {p_i}^{c_i}$ 后用 **CRT** 合并，时间复杂度 $\mathcal{O}(\sum(p_i^{c_i} + \log n + \log p))$。
- 对质数 P，考虑到 $\binom{n}{m} \bmod P^k = \dfrac{\frac{n!}{P^x}}{\frac{m!}{P^y}\frac{(n-m)!}{P^z}} P^{x-y-z} \bmod P^k$，其中 x, y, z 为对应阶乘内 P 的次数，因而只需求 $f(n) = \dfrac{n!}{P^x} \bmod P^k$ 以及对应的 x。
- 可将 $n!$ 拆解为以下形式：

$$n! = P^{\left\lfloor\frac{n}{P}\right\rfloor} \left\lfloor\frac{n}{P}\right\rfloor! \left(\prod_{i=1, i\neq 0(\bmod P)}^{P^k} i\right)^{\left\lfloor\frac{n}{P^k}\right\rfloor} \left(\prod_{i=P^k\left\lfloor\frac{n}{P^k}\right\rfloor, i\neq 0(\bmod P)}^{n} i\right)$$

- 两个乘积项显然可以预处理关于 P^k 的循环节，循环节中前 i 项的积（模 P 为 0 视为乘 1）记为 $sum(i)$，则 $f(n)$ 有以下递推式，且 x 可在递推中顺便计算：

$$f(n) = f\left(\left\lfloor\frac{n}{P}\right\rfloor\right) sum^{\left\lfloor\frac{n}{P^k}\right\rfloor}\left(P^k - 1\right) sum\left(n \bmod P^k\right)$$
$$x = x + \left\lfloor\frac{n}{P}\right\rfloor$$

```
inline int quick_pow(int x, ll k, int mod)
{
    int res = 1;
    while (k)
    {
        if (k & 1) res = 1ll * res * x % mod;
        x = 1ll * x * x % mod; k >>= 1;
    }
    return res;
}

inline ll ex_gcd(ll a, ll b, ll &x, ll &y)
{
    if (!b)
    {
        x = 1; y = 0;
        return a;
    }
    ll e = ex_gcd(b, a % b, y, x);
    y -= a / b * x;
    return e;
```

```
}

inline int query_inv(int a, int b)
{
    ll x, y;
    ex_gcd(a, b, x, y);
    return (x % b + b) % b;
}

inline int solve_fac(ll n, int p, int mul_p, int opt)
{
    if (n < p)
        return fac[n];
    tot += n / p * opt;
    return 1ll * quick_pow(sum[mul_p - 1], n / mul_p, mul_p) * sum[n % mul_p] % mul_p
               * solve_fac(n / p, p, mul_p, opt) % mul_p;
}

inline int C(ll n, ll m, int p, int mul_p)
{
    tot = 0;
    sum[0] = 1;
    for (int i = 1; i < mul_p; ++i)
    {
        if (i % p != 0)
            sum[i] = 1ll * sum[i - 1] * i % mul_p;
        else
            sum[i] = sum[i - 1];
    }
    fac[0] = 1;
    for (int i = 1; i < p; ++i)
        fac[i] = 1ll * fac[i - 1] * i % mul_p;

    return 1ll * solve_fac(n, p, mul_p, 1)
               * query_inv(solve_fac(m, p, mul_p, -1), mul_p) % mul_p
               * query_inv(solve_fac(n - m, p, mul_p, -1), mul_p) % mul_p
               * quick_pow(p, tot, mul_p) % mul_p;
}

inline int exLucas(ll n, ll m, int p)
{
    int x = p, ans = 0;
    for (int i = 2, im = sqrt(x); i <= im && i <= x; ++i)
        if (x % i == 0)
        {
            int tmp = x;
            x /= i;
            while (x % i == 0)
                x /= i;
            tmp /= x;
            ans = (ans + 1ll * C(n, m, i, tmp) * (p / tmp) % p * query_inv(p / tmp, tmp)) % p;
        }
    if (x > 1)
        ans = (ans + 1ll * C(n, m, x, x) * (p / x) % p * query_inv(p / x, x)) % p;
    return ans;
```

```
77 }
```

本原勾股数组

- **定义** 三元组 (a, b, c) 满足 $a^2+b^2=c^2$，$(a, b, c)=1$。
- **性质1** a, b 奇偶不同，且 c 为奇数

> **证明** 分情况讨论:
>
> - a, b 均为偶数，c 也为偶数，$(a, b, c)=2$，与定义矛盾。
> - a, b 均为奇数，c 为偶数，设 $a=2x+1, b=2y+1, c=2z$，则
>
> $a^2+b^2=c^2 \Leftrightarrow(2x+1)^2+(2y+1)^2=(2z)^2 \Leftrightarrow 2x^2+2y^2+2x+2y+1=2z^2$
>
>   等式左边为奇数，右边为偶数，矛盾。
> - 故只能满足 **性质1**。

- **性质2** a, b, c 两两互素。

> **证明** 反证法，若 $(a, b)=d(d>1)$，设 $a=dx, b=dy$，则
>
> $$c=\sqrt{(dx)^2+(dy)^2}=d\sqrt{x^2+y^2}$$
>
> 则 c 也含有约数 d，与定义矛盾，其余情况同理。

- **性质3** 假定 a 为奇数，b 为偶数，$c-b$，$c+b$ 均为平方数。

> **证明** 设 $d|(c-b, c+b)$，有
>
> - $d|(c-b+c+b) \Leftrightarrow d|2c$
> - $d|[c+b-(c-b)] \Leftrightarrow d|2b$
>
> 由 **性质 2** b, c 互素 且 $c-b$ 为奇数，一定有 $d=1$，则 $(c-b, c+b)=1$。
>
> 又因为 $(c-b)(c+b)=a^2$，$c-b$，$c+b$ 均为平方数。

- 由 **性质3** 设 $c-b=t^2$，$c+b=s^2$，$s>t\geq 1$，解得:

$$a=st, b=\frac{s^2-t^2}{2}, c=\frac{s^2+t^2}{2}$$

- 对于单位圆 $x^2+y^2=1, x=\frac{b}{c}, y=\frac{a}{c}$，代入上式并取 $m=\frac{t}{s}=\tan\frac{\theta}{2}, m\in\mathbb{R}$，可得:

$$(x, y)=(\frac{1-m^2}{1+m^2}, \frac{2m}{1+m^2})=(\frac{1-\tan^2\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}}, \frac{2\tan\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}})=(\cos\theta, \sin\theta)$$

威尔逊定理

- $p\text{为素数} \Leftrightarrow(p-1)!+1\equiv 0(\mod p)$

> **证明**
>
> - 先证 $(p-1)!+1\equiv 0(\mod p)\Rightarrow p\text{为素数}$
> - 反证法，假设 $p=ab, a, b\in(1, p)$。
>   - 若 $a\neq b$，显然有 $(ab-1)!\equiv 0(\mod ab)$，矛盾。
>   - 若 $a=b=\sqrt{p}$，当 $p\leq 4$ 时可直接验证，$p>4$ 必有 $\sqrt{p}, 2\sqrt{p}<p$，因此 $(p-1)!\equiv 0(\mod p)$，同样矛盾。
> - 再证 $p\text{为素数}\Rightarrow(p-1)!+1\equiv 0(\mod p)$

> - 对于集合 $S=\{x|1\le x<p, x\in\mathbb{N}\}$，不断取出 $x,y\in S, x\ne y, xy\equiv 1 \pmod p$ 配成一对，由逆元性质可知，最后必定剩下 $1, p-1$，因此有 $(p-1)!\equiv -1 \pmod p$。