数论的一些知识

一些常识

1. $a|c$、$b|c$，且 $(a,b)=1$ 则 $ab|c$
2. $a|bc$ 且 $(a,b)=1$，则 $a|c$
3. $p|ab$ 则 $p|a$ 或 $p|b$，前提是 p 是质数

---

1. 若 $d|a$ 且 $d|b$ 则 d 是 a、b 的「公约数」，最大公约数记为 $d=(a,b)$
2. 若 $a|d$ 且 $b|d$ 则 d 是 a、b 的「公倍数」，最小公倍数记为 $d=[a,b]$
3. $(a,b)*[a,b]=ab$

取模然后对约数取模，或者直接对约数取模，效果一样

> $d|m$，判断 $x \bmod m \bmod d = x \bmod d$
>
> 令 $t = x \bmod m \bmod d$，则 $x = k_1 m + k_2 d + t$
>
> 所以 $x \bmod d = t$

最大公约数：

```
ll gcd(ll x, ll y) {
    return y == 0 ? x : gcd(y, x % y);
}
```

最小公倍数：

```
ll lcm = a / gcd(a, b) * b;
```

如果需要求 $a_1, a_2, a_3, \cdots, a_n$ 的最大公约数：

```
d = a1
ls = [a1, a2, a3, ..., an]
for i in ls:
    d = gcd(d, i)
```

这段代码的时间复杂度是：$n + log(a_{max})$

---

1. 如果 a 和 b 都是奇数，那么 $(a,b)=(a-b,b)$

2. 如果 a 是偶数，b 是奇数，那么 $(a,b)=(a/2,b)$
3. 如果 a，b 都是偶数，那么 $(a,b)=2(a/2,b/2)$

关于整数解：

1. 若 $(a,b)|sum$，则必定存在 $ax+by=sum$，且 x、y 都是整数（*存在整数解*）
2. 对于方程 $ax+by=c$，如果 $(a,b)|c$ 则存在整数解，否则一定没有！
3. 因为 $(a,b)|(ax+by)$，所以 $(a,b)|c$
4. 对于方程 $ax+by+cz=d$，则 $(a,b,c)|(ax+by+cz)$，即 $(a,b,c)|d$

扩展欧几里得：

```
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, y, x);
    y -= a / b * x;
    return d;
}
```

求方程 ax + by = c 的通解：

先求出 $ax+by=(a,b)$ 的一对解 x,y

则特解：$\frac{c}{(a,b)}*x,\frac{c}{(a,b)}*y$

通解为：$\frac{c}{(a,b)}*x+\frac{b}{(a,b)}*N,\frac{c}{(a,b)}*y-\frac{a}{(a,b)}*N$

证明如下：

> $\frac{c}{(a,b)}(ax+by)=c$
>
> $\frac{c}{(a,b)}a(x+k_1),b(y+k_2)=c$
>
> $\frac{c}{(a,b)}(ax+by)+\frac{c}{(a,b)}(ak_1+bk_2)=c$，且 $ak_1+bk_2=0$
>
> 所以：$ak_1=-bk_2=-[a,b]*N=-\frac{ab}{(a,b)}*N$
>
> 所以：$k_1=-\frac{b}{(a,b)}*N$，$k_2=\frac{a}{(a,b)}*N$

求一组 ax + by = c 的正整数解：

如果求出 $ax+by=d$ 的一组解为 x_0,y_0，则原方程的特解为：$\frac{c}{d}*x_0,\frac{c}{d}*y_0$

先让 x_0 变成最小非负整数解：$\frac{c}{d}*x_0+\frac{b}{d}*N$ 转变为问题：$\frac{c}{d}*x_0$ 需要加上或者减去多少个 $\frac{b}{d}$ 才会变成非负数，先求出 $\frac{c}{d}*x_0$ 取模 $\frac{b}{d}$ 的余数，如果余数是负数，则需要再加上一个 $\frac{b}{d}$ 这样就可以编程最小非负整数

解！

如果 x_0 已经变成最小非负整数解了，那么 y_0 如果还是负数的话，就需要减去多一个 $-\frac{a}{d}$，那么平行项 x_0 就需要减掉一个 $\frac{b}{d}$ 会变成负数，所以肯定不可能成立！

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

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, y, x);
    y -= a / b * x;
    return d;
}

ll a, b, c;
ll x, y;

void solve() {
    std::cin >> a >> b >> c;
    ll d = exgcd(a, b, x, y);
    if (c % d) {
        std::cout << -1 << '\n'; return;
    }
    {
        a /= d; b /= d; c /= d;
        __int128 x1 = x; x1 *= c;
        __int128 y1 = y; y1 *= c;
        __int128 x2 = (x1 % b + b) % b;
        __int128 y2 = y1 - (x2 - x1) / b * a;
        if (y2 < 0) {
            std::cout << -1 << '\n'; return;
        }
        std::cout << (ll)x2 << ' ' << (ll)y2 << '\n';
    }
}

int main() {
    std::ios::sync_with_stdio(0);
    std::cin.tie(0); std::cout.tie(0);
    ll t; std::cin >> t; while (t --)
    solve();
    return 0;
}
```

求 ax mod m 的最小值

> $ax(\bmod m)$
>
> $ax+my(\bmod m)$

> $t \times \gcd(a,m) (\bmod m)$

求 (ax + by) mod m 的最小值

答案是：0

证明过程如下：

> $(ax+by) \bmod m$
>
> 等价于：$(ax+by+tm) \bmod m$
>
> 而 $(ax+by+tm) = k * \gcd(a,b,m)$
>
> 要让 $k * \gcd(a,b,m)$ 最小，只需要让 $k=0$ 即可

求 (ax + by + c) mod m 的最小值

答案：$\min c \bmod \gcd(a,b,m), c \bmod \gcd(a,b,m) - \gcd(a,b,m)$

证明过程如下：

> $(ax+by+c) \bmod m$
>
> 等价于：$(ax+by+c+tm) \bmod m$
>
> $k * \gcd(a,b,m) + c$ 最小
>
> 所以令：$k = -c/\gcd(a,b,m)$
>
> k 是否是真的可以让取模后的值最小呢？
>
> 假设 k 并不是答案，那么再加上 K 个 $\gcd(a,b,m)$ 后的式子 $k * \gcd(a,b,m) + K * \gcd(a,b,m) + c$ 才是答案
>
> 由于 $\gcd(a,b,m)|m$ 所以无论再继续加多少个 $\gcd(a,b,m)$ 都只会是一个以 $k * \gcd(a,b,m) + c$ 为首项，$\gcd(a,b,m)$ 为公差的循环节
>
> 还可以讨论一下为什么不是 $m - \gcd(a,b,m) + k * \gcd(a,b,m) + c$ 最小
>
> 因为 $k * \gcd(a,b,m) + c < \gcd(a,b,m)$
>
> 而 $\gcd(a,b,m)|m$，所以 $m = k\gcd(a,b,m), k \geq 1$ 所以：$m - \gcd(a,b,m) + k * \gcd(a,b,m) + c \geq (k-1) * \gcd(a,b,m)$ 对于 $k > 1$ 的情况都成立，但是如果 $k = 1$ 呢？还是有可能的哦

关于方程 ax + by + cz = d 的特解：

1. 首先 $(a,b,c)|d$ 必定成立
2. 先求出 $ax+by=(a,b)$ 的一对特解记为 x_1, y_1
3. 再求出 $(a,b)t + cz = ((a,b),c)$ 的一对特解，记为 t_1, z_1
4. 则 $ax+by+cz=(a,b,c)$ 的特解为：(x_1t_1, y_1t_1, z_1)

同余的一些性质

若 $a \equiv b(\bmod\ m)$ 且 $a \equiv b(\bmod\ n)$ 成立，则 $a \equiv b(\bmod\ [m,n])$

> $m|a-b, n|a-b$ 所以 $[m,n]|a-b$

若 $(k,m)=d$，且 $ka \equiv ka\prime(\bmod\ m)$ 则 $a \equiv a\prime(\bmod\ \frac{m}{d})$

> $m|k(a-a\prime)$
>
> $\frac{m}{d}|\frac{k}{d}(a-a\prime)$
>
> 因为 d 是 m 和 k 的最大公约数，所以 $\frac{m}{d}$ 与 $\frac{k}{d}$ 互质
>
> 所以就只可能：$\frac{m}{d}|a-a\prime$

如何求线性同余方程：$ax \equiv b(\bmod\ m)$

> $ax+my=b$
>
> 用 $exgcd$ 求出一个特解

容斥原理

假设有 n 个集合：S_1，S_2，…，S_n，求：$S_1 \cup S_2 \cup \cdots \cup S_n$

答案：1个集合的组合 - 2个集合的组合 + 3个集合的组合 - 4个集合的组合 … …

题目：给定一个整数 n 和 m 个整数，求 $1 \sim n$ 中能被这 m 个整数的某一个整除的个数有多少个?

```
#include <bits/stdc++.h>
#define ll long long

const ll N = 1e2;
ll n, m;
ll a[N];
ll res;

inline ll lcm(ll x, ll y) {
    return x / std::__gcd(x, y) * y;
}

std::vector<ll> get(ll x) {
    ll i = 1, j = 0;
    std::vector<ll> res;
    while (i <= x) {
        if (x & i) res.push_back(a[j + 1]);
        i <<= 1; j += 1;
    }
    return res;
}

void solve() {
```

```
    std::cin >> n >> m;
    for (ll i = 1; i <= m; i ++) std::cin >> a[i];
    for (ll i = 1; i < (1 << m); i ++) {
        auto vt = get(i);
        ll mo = (vt.size() & 1) ? 1 : -1;
        ll t = 1;
        for (auto x : vt) {
            t = lcm(t, x);
            if (t > n) break;
        }
        if (t > n) continue;
        res += mo * n / t;
    }
    std::cout << res << '\n';
}

signed main() {
    std::ios::sync_with_stdio(0);
    std::cin.tie(0); std::cout.tie(0);

    solve();
    return 0;
}
```

组合数

排列组合

用二进制表示组合排列：

```
for (ll i = 0; i < (1 << n); i ++)
```

用 `dfs` 求组合排列

求组合数

$C(a,b) = \frac{a!}{b!(a-b)!} = C(a-1,b) + C(a-1,b-1)$ 杨辉三角的推导方法推出下面一项

如果需要求的组合数很大，并且需要对一个质数 p 取模，可以用卢卡斯定理求：

> $C(a,b) = C(a \bmod p, b \bmod p) * C(a/p, b/p)$ 递归求下一项

```
#include <bits/stdc++.h>
#define ll long long

const ll N = 1e5 + 100;
ll fac[N]; // 阶乘
```

```cpp
ll a, b, p;

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, y, x);
    y -= a / b * x;
    return d;
}

ll C(ll a, ll b, ll mod) {
    if (a < b) return 0;
    ll p = fac[a], q = fac[b] * fac[a - b] % mod;
    ll d, x, y; d = exgcd(q, mod, x, y);
    return (p * x % mod + mod) % mod;
}

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

void solve() {
    fac[0] = 1;
    std::cin >> a >> b >> p;
    for (ll i = 1; i < N; i ++) fac[i] = fac[i - 1] * i % p;
    std::cout << lucas(a, b, p) << '\n';
}

signed main() {
    std::ios::sync_with_stdio(0);
    std::cin.tie(0); std::cout.tie(0);
    ll t; std::cin >> t; while (t --)
    solve();
    return 0;
}
```

欧拉函数

求 $1 \sim n$ 范围内与 n 互质的个数：

$n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$

则答案为：$n * \frac{p_1 - 1}{p_1} * \frac{p_2 - 1}{p_2} * \frac{p_3 - 1}{p_3} * \cdots * \frac{p_k - 1}{p_k}$

该式子也称为欧拉函数！

时间复杂的：$O(\sqrt{n})$

```cpp
// 欧拉函数
// 求与 n 互质的元素个数
ll phi(ll n) {
```

```
    if (n <= 1) return 0;
    ll ans = n;
    for (ll i = 2; i * i <= n; i ++) {
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    }
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}
```

欧拉定理

欧拉函数 $phi(m)$ 等于比 m 小的互质的个数;

令 $1 \sim m$ 范围内与 m 互质的个数为 $phi(m)$ (欧拉函数)，若 $\gcd(a, m) \equiv 1$ 则 $a^{phi(m)} \equiv 1(\bmod m)$

特殊情况下，若 m 是质数，则 $phi(m) = m - 1$，所以有：$a^{m-1} \equiv 1(\bmod m)$

若 a 和 m 互质，则:

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

如果 a 和 m 互质或者不互质，都有:

> 若 $b > phi(m)$:
>
> > $a^b \equiv a^{b \bmod phi(m) + phi(m)}$
>
> 若 $b \leq phi(m)$，直接用快速幂求解

遇到扩展欧拉定理的题很有可能需要用到高精度:

高精度加法

正整数相加

```
std::vector<ll> add(const std::vector<ll>& a, const std::vector<ll>& b) const {
    std::vector<ll> res;
    ll n = a.size(), m = b.size();
    for (ll i = 0, t = 0; i < n || i < m || t; i ++) {
        if (i < n) t += a[i];
        if (i < m) t += b[i];
        res.push_back(t % mod);
        t /= mod;
    }
    return res;
}
```

高精度减法

大 - 小

```
std::vector<ll> sub(const std::vector<ll>& a, const std::vector<ll>& b) const {
    std::vector<ll> res;
    ll n = a.size(), m = b.size();
    for (ll i = 0, t = 0; i < n; i ++) {
        t += a[i];
        ll t_ = 0;
        if (i < m) t -= b[i];
        if (t < 0) {
            t += mod;
            t_ = -1;
        }
        res.push_back(t);
        t = t_;
    }
    if (res.back() == 0) {
        ll t = n - 1;
        while (t >= 0 && res[t] == 0) t --;
        if (t < 0) res = {0};
        else res.erase(res.begin() + t + 1, res.end());
    }
    return res;
}
```

高精度乘法

```
std::vector<ll> mul(const std::vector<ll>& a, const std::vector<ll>& b) const {
    ll n = a.size(), m = b.size();
    std::vector<ll> res(n + m + 10, 0);
    for (ll i = 0; i < n; i ++) {
        for (ll j = 0; j < m; j ++) {
            res[i + j] += a[i] * b[j];
        }
        for (ll _ = i, t = 0; _ < i + m || t; _ ++) {
            t += res[_];
            res[_] = t % mod;
            t /= mod;
        }
    }
    if (res.back() == 0) {
        ll t = res.size() - 1;
        while (t >= 0 && res[t] == 0) t --;
        if (t < 0) res = {0};
        else res.erase(res.begin() + t + 1, res.end());
    }
    return res;
}
```

高精度除以单精度

```
// 高精度除法
ll div(const std::vector<ll>& a, ll b, std::vector<ll>& c) {
    ll t = 0;
    ll n = a.size();
    for (ll i = n - 1; ~i; i --) {
        t = t * 10 + a[i];
        c.push_back(t / b);
        t %= b;
    }
    c = std::vector<ll>(c.rbegin(), c.rend());
    while (c.size() > 1 && c.back() == 0) c.pop_back();
    return t;
}
```

逆元

快速幂求逆元：

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

bool ny(ll a, ll b, ll& res) {
    if (std::gcd(a, b) != 1) return false;
    res = qpow(a, b - 2, b);
    return true;
}
```

exgcd 求逆元：

> $ax \equiv 1 (\bmod\ b)$
>
> $ax + by \equiv 1$

```
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, y, x);
```

```
    y -= a / b * x;
    return d;
}

bool ny(ll a, ll b, ll& res) {
    ll d, x, y;
    d = exgcd(a, b, x, y);
    if (d != 1) return false;
    res = (x % b + b) % b;
    return true;
}
```

逆元递推式:

令 $f(i)$ 代表 i 在 $\bmod\ p$ 的逆元，则:

> $f(i) = (p - p/i) * f(p \bmod i) \bmod p$
>
> 注意 p/i 为取整除

可以快速求 $1 \sim n$ 的逆元，时间复杂度：$O(n)$

```
#include <bits/stdc++.h>
#define ll long long

const ll N = 1e7 + 10;
ll n, m;
ll inv[N];

void solve() {
    std::cin >> m >> n;
    inv[1] = 1;
    for (ll i = 2; i <= n; i ++) {
        inv[i] = (m - m / i) * inv[m % i] % m;
    }

    ll res = 0;
    for (ll i = 1; i <= n; i ++) res ^= inv[i];
    std::cout << res << '\n';
}

signed main() {
    std::ios::sync_with_stdio(0);
    std::cin.tie(0); std::cout.tie(0);

    solve();
    return 0;
}
```

借助前缀乘法求逆元

给定一个集合 $a_1, a_2, a_3, \cdots a_n$ 求每一个元素的逆元：

> 令 $s_i = a_1 * a_2 * \cdots a_i$ 前缀乘法
>
> 求出 $s_1, s_2, s_3, \cdots, s_n$
>
> 先求出 s_n 的逆元为 t_n，则 $t_n = \frac{1}{a_1} * \frac{1}{a_2} * \frac{1}{a_3} * \cdots * \frac{1}{a_n}$
>
> 所以 $\frac{1}{a_n} = t_n * s_{n-1}$，$t_{n-1} = t_n * a_n$

```
#include <bits/stdc++.h>
#define ll long long

unsigned A, B, C;
inline unsigned rng61() {
    A ^= A << 16;
    A ^= A >> 5;
    A ^= A << 1;
    unsigned t = A;
    A = B;
    B = C;
    C ^= t ^ A;
    return C;
}

const ll N = 1e7 + 10;
ll mod, n;
ll a[N], s[N], inv[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, y, x);
    y -= a / b * x;
    return d;
}

bool ny(ll a, ll mod, ll& res) {
    ll d, x, y;
    d = exgcd(a, mod, x, y);
    if (d != 1) return false;
    res = (x % mod + mod) % mod; return true;
}

void solve() {
    s[0] = 1;
    for (ll i = 1; i <= n; i ++) {
        s[i] = s[i - 1] * a[i] % mod;
    }
    ll t; ny(s[n], mod, t);
    for (ll i = n; i; i --) {
        inv[i] = t * s[i - 1] % mod;
        t = t * a[i] % mod;
    }
```

```
    ll res = 0;
    for (ll i = 1; i <= n; i ++) res ^= inv[i];
    std::cout << res << '\n';
}

signed main() {
    std::ios::sync_with_stdio(0);
    std::cin.tie(0); std::cout.tie(0);
    std::cin >> mod >> n >> A >> B >> C;
    ll cnt = 0;
    for (ll i = 1; i <= n; i ++) {
        ll t = rng61() % mod;
        if (t == 0) continue;
        a[++ cnt] = t;
    }
    n = cnt;
    if (n == 0) {
        std::cout << 0 << '\n'; exit(0);
    }
    solve();
    return 0;
}
```

生成随机数

```
unsigned ll r64 = time(0);
unsigned ll rint() {
    r64 ^= r64 >> 12;
    r64 ^= r64 << 25;
    r64 ^= r64 >> 27;
    return r64;
}
```

```
unsigned r32 = time(0);
unsigned rint() {
    r32 ^= r32 << 13;
    r32 ^= r32 >> 17;
    r32 ^= r32 << 5;
    return r32;
}
```

```
ll rnum(ll n) {
    return rll() % n + 1;
}
```

中国剩余定理

若 $m_1, m_2, \cdots, m_n$ 两两互质，则同余方程：

> $x = a_1 (\bmod \ m_1)$ $x = a_2 (\bmod \ m_2)$ $\cdots$ $\cdots$ $x = a_n (\bmod \ m_n)$

有唯一解，且 $\bmod \ M$ 的解唯一，$M = m_1 * m_2 * \cdots m_n$

令 $M_i = \frac{M}{m_i}$，M_i^{-1} 为模 m_i 的逆元

则解为：

> $x \equiv (a_1 M_1 M_1^{-1} + a_2 M_2 M_2^{-1} + \cdots + a_n M_n M_n^{-1}) \bmod M$

```
#include <bits/stdc++.h>
#define ll long long

const ll N = 1e3;
ll n;
ll a[N], m[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, y, x);
    y -= a / b * x;
    return d;
}

void solve() {
    std::cin >> n;
    for (ll i = 1; i <= n; i ++) std::cin >> m[i] >> a[i];

    ll a1 = a[1], m1 = m[1];
    ll ans = a1;

    for (ll i = 2; i <= n; i ++) {
        ll a2 = a[i], m2 = m[i];
        ll a = m1, b = m2, c = ((a2 - a1) % m2 + m2) % m2;
        ll d, x, y;
        d = exgcd(a, b, x, y);
        if (c % d) {
            std::cout << -1 << '\n';
            return;
        }
        // 求出 x 的最小特解
        x = ((c / d * x) % (b / d) + (b / d)) % (b / d);
        ans = a1 + x * m1;
        m1 = m2 / d * m1;
        ans = (ans % m1 + m1) % m1;
        a1 = ans;
    }
```

```
    std::cout << ans << '\n';
}

signed main() {
    std::ios::sync_with_stdio(0);
    std::cin.tie(0); std::cout.tie(0);

    solve();
    return 0;
}
```

筛质数

欧拉筛法

时间复杂的：$O(n)$

```
ll vi[N], ps[N], cnt;
void get_prime(ll n) {
    // 欧拉筛法
    for (ll i = 2; i <= n; i ++) {
        if (vi[i] == 0) ps[++ cnt] = i;
        for (ll j = 1; ps[j] * i <= n; j ++) {
            vi[ps[j] * i] = 1;
            if (i % ps[j] == 0) break;
        }
    }
}
```

埃式筛法

时间复杂的：$O(n\log(\log(n)))$ 很接近常数

```
ll vi[N], ps[N], cnt;
void get_prime(ll n) {
    // 埃式筛法
    for (ll i = 2; i * i <= n; i ++) {
        if (vi[i] == 0) {
            for (ll j = i * i; j <= n; j += i)
                vi[j] = 1;
        }
    }
    for (ll i = 2; i <= n; i ++) {
        if (vi[i]) continue;
        ps[++ cnt] = i;
    }
}
```

miller-rabin 判断素数

```
// ll qmul(ll a, ll b, ll mod) {
//     a = (a % mod + mod) % mod;
//     ll ans = 0;
//     while (b) {
//         if (b & 1) ans = (ans + a) % mod;
//         a = (a + a) % mod;
//         b >>= 1;
//     }
//     return ans;
// }

// ll qmul(ll x, ll y, ll m) {
//     x %= m; y %= m;
//     ll d = ((ll double)x * y / m);
//     d = x * y - d * m;
//     if (d >= m) d -= m;
//     if (d < 0) d += m;
//     return d;
// } // 32 位内可以试着用
ll qmul(__int128 a, __int128 b, __int128 mod) {
    return a % mod * (b % mod) % mod;
}
ll qpow(ll a,ll n,ll mod) { //快速幂
    ll res=1;
    while(n) {
        if(n&1) res=qmul(res,a,mod);
        a=qmul(a,a,mod);
        n>>=1;
    }
    return res;
}
bool MRtest(ll n) { //Miller Rabin Test
    if(n<3||n%2==0) return n==2;//特判
    ll u=n-1,t=0;
    while(u%2==0) u/=2,++t;
    ll ud[]={2,325,9375,28178,450775,9780504,1795265022};
    for(ll a:ud) {
        ll v=qpow(a,u,n);
        if(v==1||v==n-1||v==0) continue;
        for(ll j=1;j<=t;j++) {
            v=qmul(v,v,n);
            if(v==n-1&&j!=t){v=1;break;}//出现一个n-1，后面都是1，直接跳出
            if(v==1) return 0;//这里代表前面没有出现n-1这个解，二次检验失败
        }
        if(v!=1) return 0;//Fermat检验
    }
    return 1;
}
```

整数分块

给定一个正整数 n，求集合 $\frac{n}{1}, \frac{n}{2}, \frac{n}{3}, \cdots, \frac{n}{n}$ 的元素个数和集合

> 差不多只有 $2\sqrt{n}$ 种不同的取值

每一块的跳跃代码：

时间复杂度：$O(\sqrt{n})$

```
std::cin >> n; // 降序
for (ll l = 1; l <= n; l ++) {
    ll d = n / l, r = n / d;
    l = r;
    std::cout << d << ' ';
}
```

求 $\sum f(i)\frac{n}{i}$ 可以用整数分块，切分成很多小块 $[l,r]$ 并且对于 $i \in [l,r]$，$\frac{n}{i}$ 的值都相同，所以可以用 $\frac{n}{i}\sum f(i)$ 来求，一般会求 $f(i)$ 的前缀和！

快速幂

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

模乘

```
ll qmul(ll x, ll y, ll m) {
    x %= m; y %= m;
    ll d = ((ll double)x * y / m);
    d = x * y - d * m;
    if (d >= m) d -= m;
    if (d < 0) d += m;
    return d;
} // 32 位内可以试着用
```

```
ll qmul(ll a, ll b, ll mod) {
    a = (a % mod + mod) % mod;
    ll ans = 0;
    while (b) {
        if (b & 1) ans = (ans + a) % mod;
        a = (a + a) % mod;
        b >>= 1;
    }
    return ans;
}
```

高精度整数运算

不推荐使用，建议改用 $java$ 语言，如果 $java$ 超时，就只能用下面这个了，特殊地：

乘法运算时间复杂的：$O(n^2)$

除法运算时间复杂的：$O(n^2\log(10^n))$ 很慢很慢！

```
struct big {
    // 压位高精度
    const static ll mod = 10000000;
    const static ll len = 7;
    ll type;
    std::vector<ll> v;
    bool zero() { return v.size() == 1 && v[0] == 0; }
    big(ll x = 0) {
        type = 1;
        if (x == 0) {
            type = 0; v.push_back(0); return;
        }
        if (x < 0) { type = -1; x *= -1; }
        while (x) {
            v.push_back(x % mod); x /= mod;
        }
    }
    big(std::string s) {
        type = 1;
        ll n = s.size();
        if (s[0] == '-') {
            type = -1;
            s = std::string(s.begin() + 1, s.end());
            n = s.size();
        }
        if (s.size() == 1 && s[0] == '0') type = 0;
        for (ll i = n - 1; i >= 0; i -= len) {
            ll l = i - len + 1, r = i;
            if (l < 0) l = 0;
            ll res = 0;
            while (l <= r) {
                res = res * 10 + s[l] - '0';
```

```
                l ++;
            }
            v.push_back(res);
        }
    }
    big(const big& num) {
        v = num.v;
        type = num.type;
    }
    big& operator=(const big& num) {
        v = num.v;
        type = num.type;
        return *this;
    }
    void swap(big& num) {
        v.swap(num.v);
        std::swap(type, num.type);
    }
    void clear() { v.clear(); }
    ll size() const { return v.size(); }
    bool operator<(const big& num) const {
        if (type < num.type) return true;
        if (type > num.type) return false;
        if (v.size() != num.v.size()) return (type == 1) ? (v.size() <
num.v.size()) : (v.size() > num.v.size());
        ll n = v.size();
        for (ll i = n - 1; ~i; i --) {
            if (v[i] != num.v[i]) return (type == 1) ? (v[i] < num.v[i]) : (v[i] >
num.v[i]);
        }
        return false;
    }
    bool operator>(const big& num) const {
        return num < *this;
    }
    bool operator==(const big& num) const {
        return (*this < num) == false && (num < *this) == false;
    }
    bool operator<=(const big& num) const {
        return !(*this > num);
    }
    bool operator>=(const big& num) const {
        return !(*this < num);
    }
    std::string to_str() const {
        std::string res;
        if (type == -1) res = '-';
        auto f = [&](ll x) -> std::string {
            if (x == 0) return "0";
            std::string s;
            while (x) {
                s += x % 10 + '0'; x /= 10;
            }
            return std::string(s.rbegin(), s.rend());
```

```
        };
        res += f(v.back());
        for (auto i = v.rbegin() + 1; i != v.rend(); i ++) {
            auto t = f(*i);
            ll _ = len - t.size();
            while (_ --) res += "0";
            res += t;
        }
        return res;
    }
    friend
    std::ostream& operator<<(std::ostream& out, const big& num) {
        out << num.to_str();
        return out;
    }
    friend
    std::istream& operator>>(std::istream& on, big& num) {
        std::string s; on >> s; num = s;
        return on;
    }
    std::vector<ll> add(const std::vector<ll>& a, const std::vector<ll>& b) const
{
        std::vector<ll> res;
        ll n = a.size(), m = b.size();
        for (ll i = 0, t = 0; i < n || i < m || t; i ++) {
            if (i < n) t += a[i];
            if (i < m) t += b[i];
            res.push_back(t % mod);
            t /= mod;
        }
        return res;
    }
    std::vector<ll> sub(const std::vector<ll>& a, const std::vector<ll>& b) const
{
        std::vector<ll> res;
        ll n = a.size(), m = b.size();
        for (ll i = 0, t = 0; i < n; i ++) {
            t += a[i];
            ll t_ = 0;
            if (i < m) t -= b[i];
            if (t < 0) {
                t += mod;
                t_ = -1;
            }
            res.push_back(t);
            t = t_;
        }
        if (res.back() == 0) {
            ll t = n - 1;
            while (t >= 0 && res[t] == 0) t --;
            if (t < 0) res = {0};
            else res.erase(res.begin() + t + 1, res.end());
        }
        return res;
```

```
    }
    std::vector<ll> mul(const std::vector<ll>& a, const std::vector<ll>& b) const
{
        ll n = a.size(), m = b.size();
        std::vector<ll> res(n + m + 10, 0);
        for (ll i = 0; i < n; i ++) {
            for (ll j = 0; j < m; j ++) {
                res[i + j] += a[i] * b[j];
            }
            for (ll _ = i, t = 0; _ < i + m || t; _ ++) {
                t += res[_];
                res[_] = t % mod;
                t /= mod;
            }
        }
        if (res.back() == 0) {
            ll t = res.size() - 1;
            while (t >= 0 && res[t] == 0) t --;
            if (t < 0) res = {0};
            else res.erase(res.begin() + t + 1, res.end());
        }
        return res;
    }
    std::vector<ll> div(const std::vector<ll>& a, ll b) const {
        std::vector<ll> res(a.size());
        ll n = a.size();
        ll t = 0;
        for (ll i = n - 1; i >= 0; i --) {
            t *= mod; t += a[i];
            res[i] = t / b; t %= b;
        }
        ll _ = 0;
        for (ll i = 0; i < n; i ++) {
            _ += res[i];
            res[i] = _ % mod;
            _ /= mod;
        }
        while (res.size() && res.back() == 0) res.pop_back();
        if (res.size() == 0) res = {0};
        return res;
    }
    ll bmod(const std::vector<ll>& a, ll b) const {
        std::vector<ll> res(a.size());
        ll n = a.size();
        ll t = 0;
        for (ll i = n - 1; i >= 0; i --) {
            t *= 10; t += a[i];
            res[i] = t / b; t %= b;
        }
        return t;
    }
    bool cmp(const std::vector<ll>& a, const std::vector<ll>& b) const {
        ll n = a.size(), m = b.size();
        if (n != m) return n < m;
```

```
        for (ll i = n - 1; ~i; i --) {
            if (a[i] != b[i]) return a[i] < b[i];
        }
        return false;
    }
    big operator+(const big& num) const {
        big res;
        if (type == -1 && num.type == -1) {
            res.v = add(v, num.v);
            res.type = -1;
        }
        else if (type == -1) {
            if (cmp(v, num.v)) {
                res.type = 1;
                res.v = sub(num.v, v);
                if (res.zero()) res.type = 0;
            }
            else {
                res.type = -1;
                res.v = sub(v, num.v);
                if (res.zero()) res.type = 0;
            }
        }
        else if (num.type == -1) {
            if (cmp(v, num.v)) {
                res.type = -1;
                res.v = sub(num.v, v);
                if (res.zero()) res.type = 0;
            }
            else {
                res.type = 1;
                res.v = sub(v, num.v);
                if (res.zero()) res.type = 0;
            }
        }
        else {
            res.type = 1;
            res.v = add(v, num.v);
            if (res.zero()) res.type = 0;
        }
        return res;
    }
    big operator-(const big& num) const {
        big t = num;
        t.type *= -1;
        return *this + t;
    }
    big operator*(const big& num) const {
        big res;
        if (type == -1 && num.type == -1) {
            res.type = 1;
            res.v = mul(v, num.v);
            if (res.zero()) res.type = 0;
        }
```

```
        else if (type == -1 || num.type == -1) {
            res.type = -1;
            res.v = mul(v, num.v);
            if (res.zero()) res.type = 0;
        }
        else {
            res.type = 1;
            res.v = mul(v, num.v);
            if (res.zero()) res.type = 0;
        }
        return res;
    }
    big operator/(ll num) const {
        big res;
        if (type < 0 && num < 0) {
            res.type = 1;
            res.v = div(v, std::abs(num));
            if (res.zero()) res.type = 0;
        }
        else if (type < 0 || num < 0) {
            res.type = -1;
            res.v = div(v, std::abs(num));
            if (res.zero()) res.type = 0;
        }
        else {
            res.type = 1;
            res.v = div(v, num);
            if (res.zero()) res.type = 0;
        }
        return res;
    }
    big operator%(ll num) const {
        return *this - (*this) / num * num;
    }
    big operator/(const big& num) const {
        big res;
        big a = *this, b = num;
        if (a < 0) a *= -1;
        if (b < 0) b *= -1;
        res.type = 1;
        if (type < 0 && num.type >= 0 || type >= 0 && num.type < 0) {
            res.type = -1;
        }
        if (a < b) return 0;
        big l = big(0), r = a, mid;
        while (l < r) {
            mid = (l + r + 1) / 2;
            if (mid * b <= a) l = mid;
            else r = mid - 1;
        }
        res.v = r.v;
        if (res.zero()) res.type = 0;
        return res;
    }
```

```
    big operator%(const big& num) const {
        return *this - (*this) / num * num;
    }
    big& operator+=(const big& num) {
        return *this = *this + num;
    }
    big& operator-=(const big& num) {
        return *this = *this - num;
    }
    big& operator*=(const big& num) {
        return *this = *this * num;
    }
    big& operator/=(const big& num) {
        return *this = *this / num;
    }
    big& operator%=(const big& num) {
        return *this = *this % num;
    }
    big& operator/=(ll num) {
        return *this = *this / num;
    }
    big& operator%=(ll num) {
        return *this = *this % num;
    }
};
```