> **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》（六）正文 0x60 原根（ACM / OI / MO）（二十万字符数论书）

《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

0x60 原根

本章只简单涉及了一些原根的性质和应用，具体原根更详细的性质证明详见：https://oiwiki.org/math/number-theory/primitive-root/#_7

0x61 整数的阶、原根与指标

0x61.1 整数的阶

定义： 若a，p为正整数，且 $gcd(a,p)=1$ ，称满足 $a^r\equiv 1\pmod p$ 的**最小正整数** r 为 a **模** p **的阶** ，记作$ord_p\ a$。

我们可以将整数的阶理解为 $a\%p$下的阶 t 相当于 $a\%p$ 的循环节，每次乘上 a^t ，值都不变。

- **基本性质**

性质61.1.1： 由欧拉定理 $a^{\varphi(p)} \equiv 1 \pmod p$，故可以得到 阶 $t \mid \varphi(p)$。

性质61.1.2： 如果 a 是 p 互素的整且 $p>0$ ，则正整数 x 是同余式 $a^x \equiv \pmod p$的一个解当且仅当$ord_p\, a \mid x$。

性质61.1.3： 如果 a，n 是互素的整数且 $n>0$ ，那么 $a^i \equiv a^j \pmod p$，当且仅当 $i \equiv j \pmod{ord_p\, a}$，其中 i 和j 是非负整数。

> 因为 a 的阶 r，$a^r \equiv 1 \pmod p$，$a^i \equiv a^j \pmod p$两边同余两边一直除以a^r，相当于同时减去 r，所以 $i \equiv j \pmod{ord_p\, a}$ 的正确性显然。

性质61.1.4： 设 a 关于模 p 的阶为 r，则 $a^0, a^1, a^2, \cdots, a^{r-1}$ 互不相同。

0x61.2 整数的原根

定义： 若 $Ord_p(a) = \varphi(p)$ ，则称这样的 a 为**模数** p 的原根，记作$Rt(p) = a$。

原根就是特殊情况的阶。

理解： 对于模数 p ，找到一些数 a，满足 a 的次方**循环节**恰好为$\varphi(p)$，这些数就是模数 p 的原根

-• **基本性质定理：**

定理61.2.1： 如果 r 和 n 互质且$n>0$，则如果 r 是模 n 的一个原根，那么下列整数：

$a, a^2 \cdots a^{\varphi(n)}$

构成了模 n 的既约剩余系（简化剩余系）并且有$p-1$个，也就是恰好为$1 \cdots p-1$中与 p 互质的数的一个排列，对应着欧拉函数的定义。

即可以得到：若 p 为素数，对于模 p 的原根 g ，满足 $g^1, g^2, \cdots, g^{p-1}$，在模 p 的意义下，互不相同。（这点非常重要，引申出了 **0x64.1的乘法换加法算法**）

性质61.2.2： 若一个正整数 n 有原根，则其有 $\varphi(\varphi(n))$ 个原根 。

性质61.2.3： $x^d \equiv 1 \pmod p$恰好有 d 个解 $(d|(p-1))$

性质61.2.3： 如果 $ord_p\, a = t$ 并且 u 是一个正整数，那么有 $ord_p(a^u) = \frac{t}{gcd(t,u)}$。

> **性质61.2.2证明：** 如果 a 是模 m 的原根，那么对于任意和 $\varphi(m)$ 互质的正整数 s ，a^s 也是模 m 的原根（此时 $s, 2s, \ldots \varphi(m)\cdot s$ 在模 $\varphi(m)$ 下互不相同，于是$a^s, a^{2s}, \ldots a^{\varphi(m)\cdot s}$ 在模 m 下互不相同）。容易证明，它们就是 m 的全部原根（注意到模 m 下与 m 互质的数一共只有 $\varphi(m)$ 个，已经被 $a^1, a^2, \ldots a^{\varphi(m)}$ 占据完了）。所以原根的数量就是模 m 意义下 s 的数量，即 $\varphi(\varphi(m))$个 。

推论61.2.4： 恒等式，$a^{\frac{\varphi(xy)}{2}} \equiv 1 \pmod{xy}$ (x，y互素且a与xy互素)

在2次幂的时候就有$a^{2^k} \equiv 1 \pmod{2k+2}$ 恒成立，所以 2 的 3 次及以上次幂没有原根

5 模 2^{k+2} 的指数正好是 2^k 。

- **求解原根**

验证一个数a是否为 $Rt(P)$ ，我们可以使用试除法。显然 $a^{\varphi(P)}\equiv 1$ ，那么我们现在只需要证明**最小循环节**为 $\varphi(P)$ 即可，而如果存在更小的循环节，则一定为 $\varphi(P)$ 的因子。

我们知道一个数 n 的原根最多只有 $\varphi(\varphi(n))$ 个，我们知道$\varphi(n)\le\sqrt{n}$，这也就意味着我们在求原根的时候可以直接暴力枚举，时间复杂度也只有$O(n^{\frac{1}{4}})$。

那么预处理出 $\varphi(p)$ 的所有的质因数$(p_1,p_2\dots p_k)$，暴力判断一下，如果
$\exists i,a^{\frac{P-1}{p_i}}\equiv 1\pmod{P-1}$
说明存在更小的循环节，也就说明 a 不是模 P 的原根，因为根据原根的定义，需要保证是第一个满足 $\varphi(P)$ $(a^{P-1}\equiv 1\pmod{P-1})$的数。

Code

```cpp
ll n, m;
vector<int>factor;
vector<int> get_factor(ll n)
{
    vector<int>res;
    for(int i = 1; i * i <= n; ++ i) {
        if(n % i == 0) {
            res.push_back(i);
            if(i != n / i) {
                res.push_back(n / i);
            }
        }
    }
    return res;
}
int qpow(int a, int b, int c)
{
    int res = 1;
    while(b){
        if(b & 1) res = ((ll)res * a) % c;
        a = ((ll)a * a) % c;
        b >>= 1;
    }
    return res % c;
}
int main()
{
    scanf("%lld", &n);
    ll phi_m = n - 1;
    factor = get_factor(phi_m);
    for(int i = 2; i < n; ++ i) {
        bool flag = 1;
```

```
        for(int j = 0; j < factor.size(); ++ j) {
            if(factor[j] != phi_m && qpow(i, factor[j], n) == 1) {
                flag = 0;
                break;
            }
        }
        if(flag) {
            printf("%d\n", i);
            break;
        }
    }
    return 0;
}

```

其他的原根可用最小原根的g^u表示（g^u是 p 的原根当且仅当$\gcd(u, \varphi(p)) = 1$）

找到所有的原根

```
const int N = 1000010;
int phi[N], p[N], a[N], ans[N], n, r, t;
bool v[N];
int gcd(int a, int b) {
    return (b == 0) ? a : gcd(b, a % b);
}
ll qpow(ll x, int y, int mo) {
    ll s = 1;
    while (y) {
        if (y & 1)
            s = s * x % mo;
        x = x * x % mo;
        y >>= 1;
    }
    return s;
}
int check(int n) {
    if (n % 2 == 0)
        return -1;
    if (!v[n])
        return n;
    for (int i = 2; p[i]*p[i] <= n; i++)
        if (n % p[i] == 0) {
            while (n % p[i] == 0)
                n /= p[i];
            if (n > 1)
                return -1;
            else
                return p[i];
```

```cpp
        }
    return -1;
}
bool solve(int g, int mo) {
    if (gcd(g, mo) != 1)
        return 0;
    for (int i = 1; i <= r; i++)
        if (qpow(g, phi[mo] / a[i], mo) == 1)
            return 0;
    return 1;
}
int getg(int mo) {
    int n = phi[mo];
    r = 0;
    for (int i = 1; p[i]*p[i] <= n; i++)
        if (n % p[i] == 0) {
            while (n % p[i] == 0)
                n /= p[i];
            a[++r] = p[i];
        }
    if (n > 1)
        a[++r] = n;
    for (int i = 2; i < mo; i++)
        if (solve(i, mo))
            return i;
}
int main() {
    phi[1] = 1;
    for (int i = 2; i <= 1000000; i++) {
        if (!v[i]) {
            phi[i] = i - 1;
            p[++t] = i;
        }
        for (int j = 1; j <= t; j++) {
            if (i * p[j] > 1000000)
                break;
            v[i * p[j]] = 1;
            if (i % p[j] == 0) {
                phi[i * p[j]] = phi[i] * p[j];
                break;
            } else
                phi[i * p[j]] = phi[i] * (p[j] - 1);
        }
    }
    while (scanf("%d", &n) != EOF) {
        if (n == 2) {
            puts("1");
```

```
            continue;
        }
        if (n == 4) {
            puts("3");
            continue;
        }
        int ch = (n % 2 == 1) ? check(n) : check(n / 2);
        if (ch == -1) {
            puts("-1");
            continue;
        }
        int g = getg(n);
        int l = 0;
        for (int i = 1; i <= phi[n]; i++)
            if (gcd(i, phi[n]) == 1)
                ans[++l] = qpow(g, i, n);
        sort(ans + 1, ans + 1 + l);
        for (int i = 1; i <= l; i++) {
            printf("%d", ans[i]);
            if (i == l)
                puts("");
            else
                printf(" ");
        }
    }
}
```

0x61.3 整数的指标（也称指数、离散对数）

定义： 设 r 为模 P 的原根，有 $r^x \equiv n \pmod p, (1 \le x \le \varphi(m))$成立的唯一整数 x，则称 x 为以 r 为底，n 的离散对数，记作：$x = ind_r(n)$ 我们简记为 $x = I_r(n)$

我们发现这个方程实际上就是我们前面讲过的第一种高次同余方程 $a^x \equiv b \pmod p$。

我们发现指标与数学中的对数拥有很多相似的性质，只需要将等式用模 $\varphi(m)$ 的同余式来代替即可。

- **基本性质定理**

性质61.3.1： $a \equiv b \pmod p \Longleftrightarrow I_r(a) \equiv I_r(b) \pmod{\varphi(p)}$

性质61.3.2： 指标类似对数，可以进行乘法与加法的转换：$I_r(ab) \equiv I_r(a) + I_r(b) \pmod{\varphi(p)}$

性质61.3.2： 类似对数，$I_r(a^k) \equiv k \times I_r(a) \pmod{\varphi(p)}$

- **求解**

求整数的指标实际上就是求出$r^x \equiv n \pmod p$。我们可以先用枚举法求出 p 的原根 r，再用 BSGS 算法求出以 r 为底的 n 的离散对数 x 即可。

0x62 素数的原根

定理62.1.1： 每个素数都有原根。

定理62.1.2： **若 g 是某一奇素数 p 的原根，那么 g 也是 p^k 的原根**

> 这是一条非常有用的定理。

0x62.1 多项式同余

假设 $f(x)$ 是一个整数系数多项式，称整数 c 是 $f(x)$ 模 m 的根是指 $f(c) \equiv 0 \pmod m$。

易知，如果 c 是 $f(x)$ 模 m 的根，那么每一个模 m 同余于与 c 的整数也是一个根。

定理62.1.1（拉格朗日定理）： 假设 $f(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$ 是一个次数为 n ，首项系数 a_n ，不能被 p 整除的整系数多项式，且 $n\geq 1$，那么 $f(x)$ 至多只有 n 个模 p 不同余的根。

定理62.1.2： 假设 p 为素数，且 d 是 $p-1$ 的因子，那么多项式 x^d-1 恰有 d 个模 p 不同余的根。

定理62.1.3： 假设 p 为素数，且 d 是 $p-1$ 的正因子，那么比 p 小且模 p 的阶为 d 的正整数个数不超过 $\varphi(d)$个。

定理62.1.3： 假设 p 为素数，且 d 是 $p-1$ 的正因子，那么模 p 的阶为 d 且不同余的整数个数为 $\varphi(d)$个。

推论62.1.4： 每个素数都有原根。

0x63 原根的存在性

定理63.1.1： 如果 p 是一个素数，且有原根 r ，那么 r 或 $r+p$ 是模 p^2 的一个原根。

性质63.1.2： 根据 **性质61.2.2** 可知，一个模数 m 有原根的充要条件是
$m=1,2,4,p^C,2p^C(\text{p is prime},C\in Z^*)$

0x64 原根的应用

0x64.1乘法换加法（取模意义下）

我们根据原根的**性质61.2.1**：**若 p 为素数，对于模 p 的原根 g ，满足 $g^1,g^2,\cdots,g^{p-1}$，在模 p 的意义下，互不相同。** 可得：任**何一个小于正整数m的正整数都可以表示为原根的某次幂**，那么求一个由小于m正整数组成的数列的累乘值可以转化为原根的指数的求和，这样可以不用计算高精度，且由周期性还可以缩小指数的范围。

Problem A （题目链接）

小C有一个集合 S ，里面的元素都是小于 M 的非负整数。有一个数列生成器，可以生成一个长度为 N 的数列，数列中的每个数都属于集合 S 。他用这个生成器生成了许多这样的数列。问给定整数 x，求所有可以生成出的，且满足数列中所有数的乘积 $\bmod M$ 的值等于 x 的不同的数列的有多少个。认为，两个数列$\{A_i\}$和$\{B_i\}$不同，当且仅当至少存在一个整数i，满足$A_i\neq B_i$。输出答案

$\bmod 1004535809$。（$1 \le N \le 10^9, 3 <= M <= 8000$，M为质数，$1 \le x \le M-1$，输入数据保证集合 S 中元素不重复）

Solution

对于本题我们需要求出数列中所有元素的乘积，我们可以应用乘法转加法得到数列乘积的原根指数，再将 x 处理成原根的某次幂，根据阶的**性质61.1.3** 即可判定是否同余。

更详细的题解：https://www.luogu.com.cn/blog/ZigZagKmp/solution-p3321

Code

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

using namespace std;
#define int long long
using ll = long long;
const int N = 2e6 + 7, mod = 1004535809, G = 3;

ll a[N], b[N], h[N];
int n, m, X, S, GG;
int limit, RR[N], L;
ll g[N], f[N], dtol[N], ltod[N];
vector<int>factor;

int qpow(int a, int b, int mod = 1004535809)
{
    int res = 1;
    while(b){
        if(b & 1) res = 1ll * res * a % mod;
        a = 1ll * a * a % mod;
        b >>= 1;
    }
    return res;
}

vector<int> get_factor(ll n)
{
    vector<int>res;
    for(int i = 1; i * i <= n; ++ i) {
        if(n % i == 0) {
            res.push_back(i);
            if(i != n / i) {
                res.push_back(n / i);
            }
        }
    }
```

```
    return res;
}

void get_RT(int n)
{
    ll phi_m = n - 1;
    factor = get_factor(phi_m);
    for(int i = 2; i < n; ++ i) {
        bool flag = 1;
        for(int j = 0; j < (int)factor.size(); ++ j) {
            if(factor[j] != phi_m && qpow(i, factor[j], n) == 1) {
                flag = 0;
                break;
            }
        }
        if(flag) {
            GG = i;
            break;
        }
    }
    int tmp = 1;
    for(int i = 0; i < phi_m; ++ i, tmp = 1ll * tmp * GG % n)
        dtol[tmp] = i, ltod[i] = tmp;
    return ;
}

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

void NTT(ll *A, int type = 1)
{
    for(int i = 0; i < limit; ++ i)
        if(i < RR[i])
            swap(A[i], A[RR[i]]);
    for(int mid = 1; mid < limit; mid <<= 1) {
        ll wn = qpow(G, (mod - 1) / (mid * 2));
        if(type == -1) wn = qpow(wn, mod - 2);
        for(int len = mid << 1, pos = 0; pos < limit; pos += len) {
            ll w = 1;
            for(int k = 0; k < mid; ++ k, w = (w * wn) % mod) {
                int x = A[pos + k], y = w * A[pos + mid + k] % mod;
                A[pos + k] = (x + y) % mod;
                A[pos + k + mid] = (x - y + mod) % mod;

            }
        }
    }

```

```cpp
    if(type == -1) {
        ll limit_inv = inv(limit);
        for(int i = 0; i < limit; ++ i)
            A[i] = (A[i] * limit_inv) % mod;
    }
}

void mul(ll *f, ll *g, ll *ans, int n, int m, int mod_len)
{
    limit = 1, L = 0;
    while(limit < n + m - 1) limit <<= 1, ++ L;
    for(int i = 0; i < limit; ++ i)
        RR[i] = (RR[i >> 1] >> 1) | ((i & 1) << (L - 1));

    for(int i = 0; i < n; ++ i)
        a[i] = f[i];
    for(int i = n; i < limit; ++ i)
        a[i] = 0;
    for(int i = 0; i < m; ++ i)
        b[i] = g[i];
    for(int i = m; i < limit; ++ i)
        b[i] = 0;

    NTT(a), NTT(b);

    for(int i = 0; i < limit; ++ i) {
        a[i] = 1ll * a[i] * b[i] % mod;
    }
    NTT(a, -1);

    for(int i = 0; i < mod_len; ++ i)
        ans[i] = a[i];
    for(int i = mod_len; i < limit; ++ i)
        ans[i % mod_len] = (ans[i % mod_len] + a[i]) % mod;
    for(int i = mod_len; i < limit; ++ i)
        ans[i] = 0;
}

signed main()
{
    scanf("%lld%lld%lld%lld", &n, &m, &X, &S);
    get_RT(m);
    for(int i = 1; i <= S; ++ i) {
        int x;
        scanf("%lld", &x);
        if(x != 0) {
            f[dtol[x]] ++ ;
```

```
        }
    }
    g[0] = 1;
    while(n) {
        if(n & 1) mul(f, g, g, m - 1, m - 1, m - 1);
        mul(f, f, f, m - 1, m - 1, m - 1);
        n >>= 1;
    }

    printf("%lld\n", g[dtol[X]]);
    return 0;
}
```

0x64.2 快速数论变换

快速数论变换实际上是

我们发现FFT里的大多操作，都跟单位根没有什么关系，（例如选点插值，奇偶分治），我们随便选择一个 n个点$v, v^2, v^3, \dots, v^N$，最后可以得到公式$A(k) = A1(x) + V^k \times A2(k)$，现在问题是这个公式只有在 $k < \frac{N}{2}$的时候才能成立，我们需要找到 $k \geq \frac{N}{2}$ 的时候的答案，这样才能得到整体的答案。所以FFT的关键问题就是如何得到 $k \geq \frac{N}{2}$ 时的统一的递推式，进而可以把两个子问题合并，使用分治的思想$O(nlogn)$的时间完成整个过程。

我们之所以使用单位根，即令$v = \omega_N$，也就是$e^{\frac{2i\pi}{N}}$，就是 因为单位根拥有四个重要性质：

- $\omega_N^N = 0$
- $\omega_N^{\frac{N}{2}} = -1$
- $\omega_{2N}^{2k} = \omega_N^k$（折半定理）
- $\omega_N^{k+\frac{N}{2}} = -\omega_N^k$（消去定理）

我们根据性质1和3，可以得到$k < \frac{N}{2}$的时候，$A1(\omega_N^k) = A1(\omega_{\frac{N}{2}}^k) + \omega_N^k \times A2(\omega_{\frac{N}{2}}^k)$

根据性质2和4，可以得到$k \geq \frac{N}{2}$的时候，$A1(\omega_N^k) = A1(\omega_{\frac{N}{2}}^k) - \omega_N^k \times A2(\omega_{\frac{N}{2}}^k)$

（上面的证明请参考我的快速傅里叶变换）

那么 ω_N 涉及到复数，可能会出现精度误差，那么有没有同样拥有上述性质的 n 个数 v，并且满足互不相等呢？

我们做数学题经常会用到取模运算，所以我们考虑模**质数** p 意义下的一个神奇的东西：原根。上面我们讲述了原根的性质，所以我们可以来证明一下它是否具有和ω相同的性质。

我们设 g 是 p 的原根，令 $g_N = g^{\frac{p-1}{N}}$ ，其中要求$p-1$ 能被 N 整除

我们参照 ω 考虑证明下面同样的四个性质。

- $g_N^N \equiv 1 \pmod p$
- $g_N^{\frac{N}{2}} \equiv -1 \pmod p$
- $g_{2N}^{2k} \equiv g_N^k \pmod p$
- $g_N^{\frac{N}{2}} = g^{p-1} \equiv -1 \pmod p$

简单证明：

> 性质1：根据原根的定义
> $g^{\varphi(p)}=g^{p-1}\equiv 1\pmod p$
> $g_N^N=(g_N)^N=(g^{\frac{p-1}{N}})^N=g^{p-1}\equiv 1\pmod p$ □

> 性质2：我们化简：$g_N^{\frac{N}{2}}=(g^{\frac{p-1}{N}})^{\frac{N}{2}}=g^{\frac{p-1}{2}}$
> 而 $(g^{\frac{p-1}{2}})^2=g^{p-1}\equiv 1\pmod p$
> 因为 g 是原根，根据原根的定义，$g^{\frac{p-1}{2}}$ 不能等于1，因为等于了就不是原根了，所以 $g^{\frac{p-1}{2}}$ 就只能等于-1了hhh □
> （这里也一并证明出了奇素数的原根必是它的二次非剩余）
> 剩余性质的证明参见 我的博文，里面包含了NTT具体的实现以及代码，这里不再列出。

0x65 高次同余方程（二）（N 次剩余）

在前面的章节我们讲解了第一种高次同余方程的解法，利用BSGS可以在非常优秀的时间复杂度下解决。这里我们来探讨第二种高次同余方程：$x^a\equiv b\pmod p$ 的解法。

Template 1 X^A Mod P （51 nod 1038）

解高次同余方程：$x^a\equiv b\pmod p$，其中 P 为质数。给出 P 和AB，求小于 P 的所有 X。所有数据中，解的数量不超过 $\sqrt{p}$。

$1\le T\le 100, 1\le A,B<P\le 10^9$，$P$为质数。

Solution

p 是奇质数且 p 不能整除 d ，判定 $x^n\equiv d\pmod p$ 是否有解

若 $n|(p-1)$ ，则 d 是模 p 的 n 次剩余的充要条件为：$d^{\frac{(p-1)}{n}}\equiv 1\pmod p$ **且有解时，解数为** n 。
若 n 不能整除 $p-1$ ，则 d 是模 p 的 n 次剩余的充要条件为：$d^{\frac{(p-1)}{k}}\equiv 1\pmod p$，其中 $k=gcd(n,p-1)$，**且有解时解数为** k 。

- **解高次同余方程的步骤**

1. 先求出 p 的一个原根 g 。
2. 求出以 g 为底的 b 关于模 p 的指数 r ，即 $g^r\equiv b\ (\mod p)$，即离散对数，前面讲解了如何求离散对数，直接使用BSGS求解即可。
3. 令 $x\equiv g^y\ (\mod p)$，带回原式就是求 $g^{ay}\equiv g^r\ (\mod p)$。(因为 1 ~ p 的数都可以表示为原根 g 的次幂模 p 的形式)
4. 那么根据阶的 **性质61.1.3** 我们知道：$ay\equiv r\ (\mod ord_p\ g)$，即为一个一次同余方程，我们可以直接使用 `exgcd` 来求解。
5. 在0 ~ $\varphi(p)-1$中求得 y 的解，带回式子 $x\equiv g^y\ (\mod p)$，又是一个一次同余方程，继续exgcd求解即可。

- **一些细节**

根据原根的**性质61.2.1**可得，我们求出来的答案一定是没有重复的。由于我们使用的是原根，p 是质数，原根一共有 $\varphi(\varphi(p))$ 个，仅求出通解的高次同余方程的时间复杂度为$O(\sqrt{p})$。

Code

```
#define int long long
const int N = 500007;
unordered_map<int, int> hsh;
int prime[N], tot, t, phi, ans[N], num, a, b, p, g, x, y;
void solve(int x) {
    int tmp = x;
    tot = 0;
    for (int i = 2; i * i <= x; i++) {
        if (tmp % i == 0) {
            prime[++tot] = i;
            while (tmp % i == 0)
                tmp /= i;
        }
    }
    if (tmp > 1)
        prime[++tot] = tmp;
}
int qpow(int x, int b) {
    int tmp = 1;
    while (b) {
        if (b & 1) tmp = tmp * x % p;
        b >>= 1;
        x = x * x % p;
    }
    return tmp;
}
int BSGS(int a, int b) {
    hsh.clear();
    int block = sqrt(p) + 1;
    int tmp = b;
    for (int i = 0; i < block; i++, tmp = tmp * a % p)
        hsh[tmp] = i;
    a = qpow(a, block);
    tmp = 1;
    for (int i = 0; i <= block; i++, tmp = tmp * a % p)
        if (hsh.count(tmp) && i * block - hsh[tmp] >= 0)
            return i * block - hsh[tmp];
}
int exgcd(int &x, int &y, int a, int b) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int d = exgcd(x, y, b, a % b);
```

```
    int z = x; x = y; y = z - (a / b) * y;
    return d;
}
int cnt;
signed main() {
    scanf("%lld", &t);
    while (t -- ) {
        scanf("%lld%lld%lld", &p, &a, &b);
        b %= p;
        phi = p - 1;
        solve(phi);
        for (int i = 1; i <= p; i++) {
            bool flag = false;
            for (int j = 1; j <= tot; j++)
                if (qpow(i, phi / prime[j]) == 1) {
                    flag = true;
                    break;
                }
            if (flag == false) {
                g = i;
                break;
            }
        }
        int r = BSGS(g, b);
        int gcd = exgcd(x, y, a, phi);

        if (r % gcd != 0) {
            puts("No Solution");
            continue;
        }

        x = x * r / gcd;
        int k = phi / gcd;
        x = (x % k + k) % k;
        num = 0;
        while (x < phi) ans[++num] = qpow(g, x), x += k;
        sort(ans + 1, ans + 1 + num);
        for (int i = 1; i <= num; i ++ ) printf("%lld%s", ans[i], i == num ? "\n" : " ");
    }
    return 0;
}
```

Template 2【模板】N次剩余（Luogu P5668）

你需要解方程 $x^n \equiv k \pmod m$，其中 $x \in [0, m-1]$。

输出第一行 为不同解的个数 c。

若 $c \neq 0$，接下来第二行共 c 个整数，**升序输出所有可能解**，空格隔开。

数据保证 $\sum c_i \leq 10^6$ 。

$1 \leq T \leq 100, 1 \leq n \leq 10^9, 1 \leq k < m \leq 10^9$

设 m 的唯一分解形式为 $m = \prod_{i=1}^{k} p_i^{q_i}$。

保证方程 $x^n \equiv k \pmod{p_i^{q_i}}$在 $[0, p_i^{q_i})$中的解数 $\leq 10^6$

Solution

Code

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

typedef long long LL;

int A, B, mod;
int pow(int x, int y, int mod = 0, int ans = 1) {
    if (mod) {
        for (; y; y >>= 1, x = (LL) x * x % mod)
            if (y & 1) ans = (LL) ans * x % mod;
    } else {
        for (; y; y >>= 1, x = x * x)
            if (y & 1) ans = ans * x;
    }
    return ans;
}

struct factor {
    int prime[20], expo[20], pk[20], tot;
    void factor_integer(int n) {
        tot = 0;
        for (int i = 2; i * i <= n; ++i) if (n % i == 0) {
            prime[tot] = i, expo[tot] = 0, pk[tot] = 1;
            do ++expo[tot], pk[tot] *= i; while ((n /= i) % i == 0);
            ++tot;
        }
        if (n > 1) prime[tot] = n, expo[tot] = 1, pk[tot++] = n;
    }
    int phi(int id) const {
        return pk[id] / prime[id] * (prime[id] - 1);
    }
} mods, _p;

int p_inverse(int x, int id) {
    assert(x % mods.prime[id] != 0);
```

```cpp
    return pow(x, mods.phi(id) - 1, mods.pk[id]);
}

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 inverse(int x, int mod) {
    assert(std::__gcd(x, mod) == 1);
    int ret, tmp;
    exgcd(x, mod, ret, tmp), ret %= mod;
    return ret + (ret >> 31 & mod);
}

std::vector<int> sol[20];

void solve_2(int id, int k) {
    int mod = 1 << k;
    if (k == 0) { sol[id].emplace_back(0); return; }
    else {
        solve_2(id, k - 1); std::vector<int> t;
        for (int s : sol[id]) {
            if (!((pow(s, A) ^ B) & mod - 1))
                t.emplace_back(s);
            if (!((pow(s | 1 << k - 1, A) ^ B) & mod - 1))
                t.emplace_back(s | 1 << k - 1);
        }
        std::swap(sol[id], t);
    }
}

int BSGS(int B, int g, int mod) { // g^x = B (mod M) => g^iL = B*g^j (mod M) : iL - j
    std::unordered_map<int, int> map;
    int L = std::ceil(std::sqrt(mod)), t = 1;
    for (int i = 1; i <= L; ++i) {
        t = (LL) t * g % mod;
        map[(LL) B * t % mod] = i;
    }
    int now = 1;
    for (int i = 1; i <= L; ++i) {
        now = (LL) now * t % mod;
        if (map.count(now)) return i * L - map[now];
    }
    assert(0);
}

```

```cpp
int find_primitive_root(int id) {
    int phi = mods.phi(id); _p.factor_integer(phi);
    auto check = [&] (int g) {
        for (int i = 0; i < _p.tot; ++i)
            if (pow(g, phi / _p.prime[i], mods.pk[id]) == 1)
                return 0;
        return 1;
    };
    for (int g = 2; g < mods.pk[id]; ++g) if (check(g)) return g;
    assert(0);
}

void division(int id, int a, int b, int mod) { //  ax = b (mod M)
    int M = mod, g = std::__gcd(std::__gcd(a, b), mod);
    a /= g, b /= g, mod /= g;
    if (std::__gcd(a, mod) > 1) return;
    int t = (LL) b * inverse(a, mod) % mod;
    for (; t < M; t += mod) sol[id].emplace_back(t);
}

void solve_p(int id, int B = ::B) {
    int p = mods.prime[id], e = mods.expo[id], mod = mods.pk[id];
    if (B % mod == 0) {
        int q = pow(p, (e + A - 1) / A);
        for (int t = 0; t < mods.pk[id]; t += q)
            sol[id].emplace_back(t);
    } else if (B % p != 0) {
        int phi = mods.phi(id);
        int g = find_primitive_root(id), z = BSGS(B, g, mod);
        division(id, A, z, phi);
        for (int &x : sol[id]) x = pow(g, x, mod);
    } else {
        int q = 0; while (B % p == 0) B /= p, ++q;
        int pq = pow(p, q);
        if (q % A != 0) return;
        mods.expo[id] -= q, mods.pk[id] /= pq;
        solve_p(id, B);
        mods.expo[id] += q, mods.pk[id] *= pq;
        if (!sol[id].size()) return;

        int s = pow(p, q - q / A);
        int t = pow(p, q / A);
        int u = pow(p, e - q);

        std::vector<int> res;
        for (int y : sol[id]) {
            for (int i = 0; i < s; ++i)
```

```
                res.emplace_back((i * u + y) * t);
            }
            std::swap(sol[id], res);
        }
}

std::vector<int> allans;
void dfs(int dep, int ans, int mod) {
    if (dep == mods.tot) {allans.emplace_back(ans); return;}
    int p = mods.pk[dep], k = p_inverse(mod % p, dep);
    for (int a : sol[dep]) {
        int nxt = (LL) (a - ans % p + p) * k % p * mod + ans;
        dfs(dep + 1, nxt, mod * p);
    }
}

void solve() {
    std::cin >> A >> mod >> B, mods.factor_integer(mod);
    allans.clear();
    for (int i = mods.tot - 1; ~i; --i) {
        sol[i].clear();
        mods.prime[i] == 2 ? solve_2(i, mods.expo[i]) : solve_p(i);
        if (!sol[i].size()) {return std::cout << 0 << '\n', void(0);}
    }
    dfs(0, 0, 1), std::sort(allans.begin(), allans.end());
    std::cout << allans.size() << '\n';
    for (int i : allans) std::cout << i << ' '; std::cout << '\n';
}

int main() {
    std::ios::sync_with_stdio(0), std::cin.tie(0);
    int tc; std::cin >> tc; while (tc--) solve();
    return 0;
}

```

我们知道此方法解高次同余方程的时间复杂度为 $\sqrt{p}$，但是 p 很大，效率还是有些低下，所以当 $a=2$ 的特殊情况，即二次同余方程 $x^2 \equiv n \pmod p$，我们将在下一章介绍一个更为优秀的 Cipolla 算法来解决这类问题。

- **竞赛例题选讲**