

整除及剩余

整除定义

设 a, b 是两个整数, 且 $b \neq 0$. 如果存在整数 c , 使得 $a = bc$, 则称 a 被 b 整除, 或 b 整除 a , 记作 $b \mid a$. 此时, 又称 a 是 b 的倍数, b 是 a 的因子。

整除的基本性质

- (1) 若 $a \mid b, a \mid c$, 则 $a \mid (b+c)$
- (2) 若 $a \mid b$, 那么对所有整数 $c, a \mid bc$
- (3) 若 $a \mid b, b \mid c$, 则 $a \mid c$

同余基本定义和定理

定义 1: 带余除法

设 a, b 是两个整数, 且 $b \neq 0$, 则存在唯一的整数 q 和 r , 使

$$a = qb + r, 0 \leq r < |b|$$

这个式子叫做带余除法, 并记余数 $r = a \bmod b$ 。例如 $-13 \bmod 5 = 3, 10 \bmod 2 = 0$.

定义 2: 同余

给定正整数 m , 若用 m 去除两个整数 a 和 b 所得余数相同, 称 a 和 b 对模 m 同余, 记作 $a \equiv b \pmod{m}$, 并称该式为同余式; 否则称 a 和 b 对模 m 不同余。

定义 3: 剩余类

在模 m 的意义下, 余数相同的归为一个集合, 那么所有整数被分为 m 个不同的集合, 模 m 的余数分别为 $0, 1, 2, 3, \dots, m-1$, 这些集合被称为模 m 剩余类 (同余类)。每个同余类中的任意两个整数都是模 m 同余的。

定义 4: 完系

一个整数的集合, 对 m 取模后, 余数遍历了 $0, 1, 2, \dots, m-1$. 那么该整数集合是模 m 完全剩余系, 如 $\{-4, 3, 5, 10\}$, 即为模 4 的完全剩余系。

定理 1: $a \equiv b \pmod{m}$, 当且仅当存在整数 k , 使得 $a = b + km$, 即 $m \mid (a-b)$

定理 2: 同余关系是等价关系, 即类似“=”

- (1) 自反性: $a \equiv a(\text{mod } m)$
- (2) 对称性: 若 $a \equiv b(\text{mod } m)$, 则 $b \equiv a(\text{mod } m)$
- (3) 传递性: 若 $a \equiv b(\text{mod } m)$, $b \equiv c(\text{mod } m)$, 则 $a \equiv c(\text{mod } m)$

定理 3: 同余的三则运算

若 a, b, c 是整数, m 是正整数, 且 $a \equiv b(\text{mod } m)$, 则

- (1) $a + c \equiv b + c(\text{mod } m)$
- (2) $a - c \equiv b - c(\text{mod } m)$;
- (3) $ac \equiv bc(\text{mod } m)$;

定理 4: 同余式的三则运算

设 a, b, c, d 为整数, m 为正整数, 若 $a \equiv b(\text{mod } m)$, $c \equiv d(\text{mod } m)$ 则

- (1) $ax + cy \equiv bx + dy(\text{mod } m)$, 其中 x, y 为任意整数, 即同余式可以相加
- (2) $ac \equiv bd(\text{mod } m)$, 即同余式可以相乘
- (3) $a^n \equiv b^n(\text{mod } m)$, 其中 $n > 0$
- (4) $f(a) \equiv f(b) (\text{mod } m)$, 其中 $f(x)$ 为任一整系数多项式

定理 5: 设 a, b, c, d 为整数, m 为正整数, 则

- (1) 若 $a \equiv b(\text{mod } m)$, 且 $d \mid m$, 则 $a \equiv b(\text{mod } d)$
- (2) 若 $a \equiv b(\text{mod } m)$, 则 $(a, m) = (b, m)$;
- (3) $a \equiv b(\text{mod } m_i) (1 \leq i \leq n)$ 同时成立, 当且仅当 $a \equiv b(\text{mod } [m_1, m_2, \dots, m_n])$

素数

素数的定义

素数（质数）是大于 1 的正整数，并且除了 1 和它本身不能被其他正整数整除。大于 1 的非素数的正整数称为合数。

素数的分布

素数有无穷多个，能估计出小于一个正实数 x 的素数有多少个，并用 $\pi(x)$ 来表示，这就是素数定理

素数定理： 随着 x 的增长， $\frac{\pi(x)}{x/\ln(x)}=1$ 。具体数据如表

n	10^3	10^4	10^5	10^6	10^7
$\pi(n)$	168	1229	9592	78498	664579
$n/\ln(n)$	145	1086	8686	72382	620421
$\pi(n)/(n/\ln n)$	1.159	1.132	1.104	1.085	1.071

推论： 令 p_n 是第 n 个素数，其中 n 是正整数，那么 $p_n \sim n\ln(n)$

算术基本定理

每个正整数都可以惟一地表示成质数的乘积，其中质数因子从小到大依次出现（这里的“乘积”可以有 0 个、1 个或多个质因子）。

换句话说，任意正整数 n 可以写成

$$n = p_1^{a_1} p_2^{a_2} \cdots p_m^{a_m} = \prod_{i=1}^m p_i^{a_i}$$

其中 p_i 是素数， a_i 是非负整数。

这个定理也叫做惟一分解定理。

素数判定

若要判定一个数 n 是否是素数，常用方法是枚举 2 到 \sqrt{n} 之内的所有数 i ，依次判断 n 是

否是 i 的倍数，因为若 $n = pq$ ，则 p, q 中必定有一个数小于等于 \sqrt{n} 。

素数筛法

有时我们需要知道 1 到 n 内的所有数是否是素数，这时我们就不能一个个枚举判定每个数字了（复杂度太高），常用的方法是使用筛法，即用每个素数去“筛”出是它的倍数的非素数。复杂度 $O(n \log \log n)$ 。

```
1. void sieve() {
2.     memset(prime, true, sizeof(prime));
3.     prime[0] = prime[1] = false;
4.     for (int i = 2; i <= n; ++i) {
5.         if (!prime[i]) continue;
6.         for (int j = i * i; j <= n; j += i)
7.             prime[j] = false;
8.     }
9. }
```

欧拉函数

欧拉函数 $\varphi(n)$ 指不超过 n 且与 n 互素的正整数的个数，其中， n 是一个正整数。

欧拉函数的性质：它在整数 n 上的值等于对 n 进行质因子分解后，所有的素数幂上的欧拉函数之积。即对于 $n = p_1^{a_1} p_2^{a_2} p_3^{a_3} \dots p_s^{a_s}$ 有 $\varphi(n) = \varphi(p_1^{a_1}) \varphi(p_2^{a_2}) \varphi(p_3^{a_3}) \dots \varphi(p_s^{a_s})$

定理 1：如果 p 是素数，那么 $\varphi(p) = p - 1$ ；反之，如果 p 是一个正整数且满足 $\varphi(p) = p - 1$ ，那么 p 是素数。

定理 2：设 p 是素数， a 是一个正整数，那么 $\varphi(p^a) = p^a - p^{a-1}$

定理 3：设 m 和 n 是互质的正整数，那么 $\varphi(mn) = \varphi(m)\varphi(n)$

定理 4：设 $n = p_1^{a_1} p_2^{a_2} p_3^{a_3} \dots p_s^{a_s}$ 为正整数 n 的素数幂分解，那么

$$\varphi(n) = n \left(1 - \frac{1}{p_1}\right) \left(1 - \frac{1}{p_2}\right) \left(1 - \frac{1}{p_3}\right) \dots \left(1 - \frac{1}{p_s}\right)$$

推论：当 n 为奇数时，有 $\varphi(2n) = \varphi(n)$

定理 5: 设 n 是一个大于 2 的正整数, 那么 $\varphi(n)$ 是偶数。

定理 6: 设 n 为一个正整数, 那么 $\sum_{d|n} \varphi(d) = n$

欧拉定理: 对于任何两个互质的正整数 $a, m (m \geq 2)$ 有 $a^{\varphi(m)} \equiv 1 \pmod{m}$

费马小定理: 当 m 是质数时候, $a^{m-1} \equiv 1 \pmod{m}$

欧拉函数的应用

最直接的应用就是利用欧拉函数求得不超过 n 且与 n 互素的正整数的个数, 其次可以利用欧拉定理与费马小定理来求得一个乘法逆元, 欧拉定理中的 m 适用任何正整数, 而费马小定理只能要求 m 是质数。

练习题

POJ 2689 ; POJ 3421 ; POJ 3090

（拓展）欧几里得算法

最大公约数与最小公倍数

定义 1

设 a 和 b 是两个整数，如果 $d \mid a$ 且 $d \mid b$ ，则称 d 是 a 与 b 的公因子

定义 2

设 a 和 b 是两个不全为 0 的整数，称 a 与 b 的公因子中最大的为 a 与 b 的最大公因子，或最大公约数，记作 $\gcd(a, b)$ 。

定义 3

设 a 和 b 是两个非零整数，称 a 与 b 最小的正公倍数为 a 与 b 的最小公倍数，记作 $\text{lcm}(a, b)$

最大公约数与最小公倍数的性质：

(1) 若 $a \mid m, b \mid m$ ，则 $\text{lcm}(a, b) \mid m$

(2) 若 $d \mid a, d \mid b$ ，则 $d \mid \gcd(a, b)$

(3) $\text{lcm}(a, b) = \frac{ab}{\gcd(a, b)}$

(4) 设 m, a, b 是正整数，则 $\text{lcm}(ma, mb) = m \cdot \text{lcm}(a, b)$ ， $\gcd(ma, mb) = m \cdot \gcd(a, b)$

求最大公约数

算法 1 素因子分解法

首先将 a 和 b 进行素因子分解成

$$a = p_1^{r_1} p_2^{r_2} \cdots p_k^{r_k}, b = p_1^{s_1} p_2^{s_2} \cdots p_k^{s_k}$$

这里 p_1, p_2, \dots, p_k 是不同的素数，则有

$$\gcd(a, b) = p_1^{\min(r_1, s_1)} p_2^{\min(r_2, s_2)} \cdots p_k^{\min(r_k, s_k)}$$

$$\text{lcm}(a, b) = p_1^{\max(r_1, s_1)} p_2^{\max(r_2, s_2)} \cdots p_k^{\max(r_k, s_k)}$$

算法 2 欧几里得算法(辗转相除法)

定理 1 $\gcd(a, b) = \gcd(b, a \bmod b)$

也就是说两个数的最大公约数等于其中一个数与两个数模去的余数。

证明 设 $a = qb + r$, $d = \gcd(a, b)$ 。下证 $d = \gcd(b, r)$ 成立:

因为 $d = \gcd(a, b)$, 所以 $d \mid b, d \mid a$ 。

又因为 $r = qb - a$, 所以 $d \mid r$ 。因此 d 为 b 与 r 的最公约数。

假设 d 不是 b 与 r 的最大公约数

那么存在一个大于 d 的数 $e = \gcd(b, r)$, 则 $e \mid b, e \mid r$ 。

由 $a = qb + r$ 得 $e \mid a$ 。

这样我们得到 e 为 a, b 的公约数, 而 e 大于 d

这与 d 为 a, b 最大公约数矛盾, 因此 d 为 b, r 的最大公约数。

证毕。

定理 2 $\gcd(a, b) = \gcd(a, a - b)$

证明类似上面

算法实现

其实两个定理都很好写, 这给我们提供了一个迭代的式子, 只要出现 $b=0$ 的时候就可以往回回溯了。

定理 1 适用于大部分的求最大公约数, 因为比较高效

定理 2 适用于高精中的求最大公约数, 因为高精中的除法较低效。

拓展欧几里得算法

定理 设 a 和 b 不全为 0, 则存在整数 x 和 y , 使得 $\gcd(a, b) = ax + by$ 。

有时候, 我们不仅仅要求出 a, b 的最大公约数 r , 还要求出 $ax + by = r$ 的一组 $\{x, y\}$ 整数解, 那么我们就要用到拓展欧几里得算法。

这里可以使用迭代的方法, 在辗转相除的过程中计算出 $\{x, y\}$:

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

$$ax + by = d, \quad bx' + (a \bmod b)y' = d$$

$$bx' + (a - \left\lfloor \frac{a}{b} \right\rfloor \cdot b)y' = d$$

$$bx' + ay' - \left\lfloor \frac{a}{b} \right\rfloor \cdot by' = d$$

$$ay' + b(x' - \left\lfloor \frac{a}{b} \right\rfloor \cdot y') = d$$

$$x = y', y = x' - \left\lfloor \frac{a}{b} \right\rfloor \cdot y'$$

递归迭代到 $a'=d, b'=0$ 时可得此时 $x'=1, y'=0$, 再利用上面推出的 x, y 与 x', y' 的关系一步步代入计算, 得到 $ax+by=\gcd(a,b)$ 的一组解。

实际上这个递归算法就是定理的证明。

根据这个递归式, 我们还能用归纳法证明, 最后算法得出的解的大小满足:

$$|x| \leq b, |y| \leq a$$

拓展欧几里得算法的应用

拓展欧几里得算法的应用主要有以下三方面:

- (1) 求解不定方程
- (2) 求解模的逆元
- (3) 求解同余方程

线性同余方程

二元一次不定方程

定义 1 a, b, c 是整数, $a \neq 0, b \neq 0$, 那形如 $ax + by = c$ 的方程称为二元一次不定方程。

定理 1 设 a, b 是整数且 $d = (a, b)$, 如果 $d \mid c$, 那么方程存在无穷多个整数解, 否则方程不存在整数解。

定理 2 如果不定方程有解且特解为 $x = x_0, y = y_0$, 那么方程的解可以表示为

$$x = x_0 + \frac{b}{d}t, y = y_0 - \frac{a}{d}t, t \in Z$$

N 元一次不定方程

定理 3 n 元一次不定方程 $a_1x_1 + a_2x_2 + \dots + a_nx_n = c (a_1, a_2, \dots, a_n, c \in N)$ 有解的充要条件是 $(a_1, a_2, \dots, a_n) \mid c$ 。

解 N 元一次不定方程

解 n 元一次不定方程 $a_1x_1 + a_2x_2 + \dots + a_nx_n = c (a_1, a_2, \dots, a_n, c \in N)$ 时候, 可先顺次求出 $(a_1, a_2) = d_2, (d_2, a_3) = d_3, \dots, (d_{n-1}, a_n) = d_n$, 若 $d_n \mid c$, 则方程组有解, 作方程组

$$\begin{cases} a_1x_1 + a_2x_2 = d_2t_2 \\ d_2t_2 + a_3x_3 = d_3t_3 \\ \dots \\ d_{n-1}t_{n-1} + a_nx_n = c \end{cases}$$

求出最后一个方程的所有解, 然后把 t_{n-1} 的每一个值代入倒数第二个方程, 求出它的所有解, 以此类推, 即可的方程的所有解。

解 N 元一次不定方程组

M 个 n 元一次不定方程组成的方程组, 其中 $m < n$, 可以消去 $m-1$ 个未知数, 从而消去 $m-1$ 个不定方程, 将方程组转化为一个 $n-m+1$ 元的一次不定方程。

同余方程与不定方程

其实二元一次不定方程和同余方程可以互相转化，如在 $a>0$ 且 $b>0$ 的条件下，求二元一次方程 $ax+by=c$ 整数解 \Leftrightarrow 求一元线性同余方程 $ax \equiv c(\text{mod } b)$ 整数解（求出一个 x ，显然 y 也是确定的）。那么我们下面专注于求一元线性同余方程的解。

一元线性同余方程

定义 1

a, b 是整数， m 是正整数，形如 $ax \equiv b(\text{mod } m)$ ，且 x 是未知数的同余式成为一元线性同余方程。

定理 1

对于一元线性同余方程 $ax \equiv b(\text{mod } m)$ ，记 $(a, m) = d$ 。

若 $d \mid b$ 那么方程恰有 d 个模 m 不同余的解且这 d 个解的形式为 $x_0 + \frac{m}{d}t, t \in Z$ ，其中 x_0 是已知的一个解。若 $d \nmid b$ 不成立则方程无解。

求一元线性同余方程

算法

要求 $ax \equiv b(\text{mod } m)$ ，即为求 $ax + my = b$ 的解。记 $(a, m) = d$ ，先使用拓展欧几里得求 $ax + my = d$ ，如果 d 不能整除 b 则无解，否则 $\text{mod } m$ 意义下的解有 d 个，可以通过对其中某个解不断地加 $\frac{m}{d}$ 得到。（容易证明 $x_0 + n \times \frac{m}{d}$ 都为方程的解）

求一元线性同余方程组

由一元线性同余方程就有一元线性同余方程组。由上述可以知道，任何一个同余方程都可变成若干个形如 $x \equiv b(\text{mod } m)$ 的方程。

算法 1 合并法

对于模线性方程组，可以进行方程组合并，求出合并后的方程的解，这样就可以很快地推出方程的最终解。不管这样的方程有多少个，都可以两两解决，求得方程组的最终解，所以本小节以

$$x \equiv b_1 \pmod{m_1} \dots\dots\dots ①$$

$$x \equiv b_2 \pmod{m_2} \dots\dots\dots ②$$

两个方程构成的方程组为例进行讲解。

$$\text{令 } m = [m_1, m_2]$$

首先，此方程组有解的充分必要条件是 $(m_1, m_2) | (b_1 - b_2)$ ，此时方程仅有一个小于 m 的非负整数解。利用拓展欧几里得算法很容易解出来。

$$\text{式①等价于 } x = b_1 + m_1 y_1 ;$$

$$\text{式②等价于 } x = b_2 + m_2 y_2$$

$$\text{联立可得 } b_1 + m_1 y_1 = b_2 + m_2 y_2, \text{ 即 } b_1 - b_2 = m_2 y_2 - m_1 y_1$$

根据之前所学的解一元线性同余方程的方法，很容易得到此方程的解 y_2 ，因此小于 m 的非负整数解即为 $(b_2 + m_2 y_2) \% m$

算法 2 中国剩余定理

定理 4 若 $m_1, m_2, m_3, \dots, m_r$ 是两两互素的正整数，则同余方程组

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2}$$

$$x \equiv a_3 \pmod{m_3}$$

...

$$x \equiv a_r \pmod{m_r}$$

有模 $M = m_1 m_2 m_3 \dots m_r$ 的唯一解，即为中国剩余定理。

换句话说，若 $n = pq$ 且 $\gcd(p, q) = 1$ ，那么 $x \bmod p, x \bmod q$ 的值确定后， $x \bmod n$ 的值也会随之确定。

算法说明

令 $M_i = \prod_{j \neq i} m_j$ ，因为 $(M_i, m_i) = 1$ ，故存在 p_i, q_i 使 $M_i p_i + m_i q_i = 1$ 。

令 $e_i = M_i p_i$ ，有：

$$e_i = \begin{cases} 0 \pmod{m_i}, & j \neq i \\ 1 \pmod{m_j}, & j = i \end{cases}$$

故 $e_0 a_0 + e_1 a_1 + \dots + e_{n-1} a_{n-1}$ 是方程的一个解。

在 $\left[0, \prod_{i=0}^{n-1} m_i\right)$ 中只有唯一解，将求出的解对 $\prod_{i=0}^{n-1} m_i$ 取模即可。

例题

青蛙的约会 (POJ 1061)

题目大意：

有一个长为 L 的首尾相连的数轴，有两只青蛙 A,B，它们初始点分别为 s, t ，A 青蛙每步往右跳 p 步，B 青蛙每步往右跳 q 步，现在它们同时开始跳，求同时跳多少步后能跳到同一点上，若无解则输出 "Impossible"。

$$s, t, p, q, L < 2 \cdot 10^9$$

分析：

设最后的答案为 x ，根据题意可以列出下面的式子：

$$\begin{aligned} s + px &\equiv t + qx \pmod{L} \\ (p - q)x &\equiv t - s \pmod{L} \\ (p - q)x + Ly &= t - s \end{aligned}$$

因此现在需要解 $ax + by = c$ 这个方程的最小非负整数解。

若 $\gcd(a, b) = 1$ 则 $ax + by = 1$ 的 x 在 $[0, b)$ 内有唯一解，而解 $ax + by = \gcd(a, b)$ 等价于

解 $\frac{a}{\gcd(a, b)}x + \frac{b}{\gcd(a, b)}y = 1$ ，所以 x 在 $[0, \frac{b}{\gcd(a, b)})$ 内有唯一解。

解 $ax + by = c$ 时我们先求 $G = \gcd(a, b)$ ，若 $C \bmod G \neq 0$ 则无解，否则解原式等价于

解 $\frac{a}{G}x + \frac{b}{G}y = \frac{c}{G}$ 且此时 $\gcd(\frac{a}{G}, \frac{b}{G}) = 1$, 令 $A = \frac{a}{G}, B = \frac{b}{G}, C = \frac{c}{G}$, 我们可以先求出

$Ax + By = 1$ 的 x 在 $[0, B)$ 内的唯一解, 再将解 x 乘 C 就能得到最后需要的解。

代码:

```
1. #include <cstdio>
2. #include <iostream>
3.
4. long long s, t, p, q, L;
5.
6. int Gcd(int a, int b) {
7.     return b ? Gcd(b, a % b) : a;
8. }
9.
10. long long exGcd(long long a, long long b, long long &x, long long &y) {
11.     if (b == 0) return x = 1, y = 0, a;
12.     long long r = exGcd(b, a % b, y, x);
13.     y -= (a / b) * x;
14.     return r;
15. }
16.
17. int main() {
18.     std::cin >> s >> t >> p >> q >> L;
19.     // s + px = t + qx (mod L)
20.     // (p - q)x = t - s (mod L)
21.     // (p - q)x + Ly = t - s
22.     // ax + by = c
23.     long long a = (p - q + L) % L, b = L, c = (t - s + L) % L;
24.     long long x, y, G = Gcd(a, b);
25.     if (c % G != 0) {
26.         puts("Impossible");
27.         return 0;
28.     }
29.     a /= G, b /= G, c /= G;
30.     exGcd(a, b, x, y);
31.     x = (x % b + b) % b;
32.     x = x * c % b;
33.     std::cout << x << std::endl;
34.     return 0;
35. }
```

The Balance (POJ 2142)

题目大意：

现有一天平，以及质量为 a 和 b 的砝码，已知砝码数量不限且天平左右均可放砝码，现要求在天平上称出质量为 c 的物品。两种砝码可以分开放两边也可以放在同一边。求一种可行方案。要求：放置的砝码数量尽可能少；当砝码数量相同时，总质量尽可能小。

$$a, b, c \leq 10^4$$

分析：

题目即求 $ax + by = c$ 的一组解，使得 $|x| + |y|$ 尽量小，在前者尽量小时 $|ax| + |by|$ 尽量小。根据前面所提到的解的形式，最后的解要么 x 尽量小，要么 y 尽量小，两种情况都求出后判断即可。

代码：

```
1. #include <cstdio>
2. #include <iostream>
3.
4. int a, b, c, x, y, u1, u2, v1, v2, g;
5.
6. int exGcd(int a, int b, int &x, int &y) {
7.     if (b == 0) return x = 1, y = 0, a;
8.     int r = exGcd(b, a % b, y, x);
9.     y -= (a / b) * x;
10.    return r;
11. }
12.
13. int main() {
14.     while (scanf("%d%d%d", &a, &b, &c), a + b + c > 0) {
15.         g = exGcd(a, b, x, y);
16.         a /= g, b /= g, c /= g;
17.
18.         u1 = (x % b * c % b + b) % b;
19.         v1 = (c - u1 * a) / b;
20.         if (v1 < 0) v1 = -v1;
21.         v2 = (y % a * c % a + a) % a;
22.         u2 = (c - v2 * b) / a;
23.         if (u2 < 0) u2 = -u2;
24.     }
```

```

25.     if (u1 + v1 > u2 + v2 || (u1 + v1 == u2 + v2 && a * u1 + b * v1 > a * u2
        + b * v2))
26.         u1 = u2, v1 = v2;
27.     printf("%d %d\n", u1, v1);
28. }
29. return 0;
30. }

```

逆元

解一元线性同余方程

考虑解一元线性同余方程 $ax \equiv b(\text{mod } m)$ ，对于实数运算下的方程 $ax = b$ ，由于 a 存在倒数，因此很容易求解。如果在 $\text{mod } m$ 运算下，也有像满足 $ay \equiv 1(\text{mod } m)$ 这样的 a 的倒数一样的数 y 存在，一元线性同余方程就容易解了。我们把这样的 y 叫做 a 的逆元，记作 a^{-1} 。如果能求解逆元，那么对于 $ax \equiv b(\text{mod } m)$ ，方程两边同乘 a^{-1} 就可以解出 x 来了。

由于方程 $ay \equiv 1(\text{mod } m)$ 等价于存在整数 k 使得 $ay = 1 + km$ ，因此可以将它变形为求解不定方程 $ay - km = 1$ ，这个问题可以使用上面的拓展欧几里得算法来求解。同时这里也可以看出，若 $\text{gcd}(a, m) \neq 1$ ，那么逆元是不存在的。

```

1. int mod_inverse(int a, int m) {
2.     int x, y;
3.     exGcd(a, m, x, y);
4.     return (x % m + m) % m;
5. }

```

费马小定理

若 p 是素数，则对任意整数 x 都有 $x^p \equiv x(\text{mod } p)$ 。这个定理被称为费马小定理。

若 x 无法被 p 整除，则我们还有 $x^{p-1} \equiv 1(\text{mod } p)$ 。利用这个性质，在 p 是素数的情况下，我们很容易就可以求出一个数的逆元。上述式子变形后可以得到 $x^{-1} \equiv x^{p-2}(\text{mod } p)$ ，利用快速幂计算即可得到 x 的逆元。

在 p 不是素数的情况下，我们也有类似的欧拉定理可以使用：

若 $\gcd(x, m) = 1$ ，则有 $x^{\varphi(m)} \equiv 1 \pmod{m}$ ；若 $\gcd(x, m) \neq 1$ ，则有，存在一个 $h(x)$ ，对于任意 $y \geq h(x)$ ，有 $x^{y+\varphi(m)} \equiv x^y \pmod{m}$ 。（在竞赛中，通常 x 远小于 m ，因此经常直接将 $h(x)$ 当做 $\varphi(m)$ ）。 m 是素数时有 $\varphi(m) = m - 1$ ，因此欧拉定理可以看做是费马小定理的推广。

练习题

CF 527A ; POJ 2115 ; CF 449C ; POJ 1006 ; POJ 2720 ; POJ 2891 ; BZOJ 1876

高次不定方程

概述

由于高次同余方程比较复杂，在本节中仅讨论形如 $A^x \equiv B(\text{mod } C)$ 的解问题。即著名的离散对数问题。

指数模的周期性

定理 1

A^x 对 C 的模随着 x 的变化具有周期性，最大周期不超过 C 。

证明很简单，如下

由于 $A^x \text{ mod } C \in [0, 1, 2, \dots, C-1]$ ，一共只有 C 个元素，由抽屉原理得，在 n 个数之内，比如存在 A 的两个次方是同余的，即存在 $x_0, n \in \mathbb{Z}, 0 \leq n \leq C$ 使得 $A^{x_0} \equiv A^{x_0+n}(\text{mod } C)$ ，那么 $A^{x_0} A^t \equiv A^{x_0+n} A^t(\text{mod } C), t \in \mathbb{Z}$ ，既有 $A^x \equiv A^{x+n}$ 对于所有 x 成立，周期为 n 。

方程 $A^x \equiv B(\text{mod } C)$ 的解性

上述指数模的周期性可知，有解的充分必要条件为，在 $0 \leq x < C$ 内存在解。即我们只要解出在 $0 \leq x < C$ 的解即可。再结合周期性，得到通解。

注意，解出的 x 并不是个模 C 的剩余类，这与 $Ax \equiv B(\text{mod } C)$ 是不一样的，比如要求 $2^x \equiv 2(\text{mod } 2)$ 而 $x=2$ 是解， $x=0$ 不是解，说明解不可能是一个剩余类。

Baby-step giant-step 思想

Baby_step_giant_step，意为先小步，后大步，这是一个很神奇的想法。能够降低枚举的规模从 n 到 \sqrt{n} ，怎么实现的呢，我们看下面。

考虑朴素算法，我们从 0 到 $C-1$ 枚举每个数，判断每个数 x 是否满足 $A^x \equiv B(\text{mod } C)$ 。

其实我们可以将枚举的数 $0 \sim C-1$ 分为 n 组每组 $m = \frac{C}{n}$ 个数，对于每一组的数进行询问，

在这组 m 个数内是否有答案。如果有，那么我们就得到答案了。其实这样若不加优化实际上还是枚举 $0 \sim C-1$ 的复杂度，不过这时是有办法优化的。

n 我们可以随便取，为了简化复杂度，当然取 $n = \lfloor \sqrt{C} \rfloor$ 组，那么每组就有 $m = \frac{C}{n}$ 个数，然后我们对每一组进行询问，这时组内的答案可以看做是： $A^{im-y} \equiv B(\text{mod } C)$ ，这里 $1 \leq i \leq n, 0 \leq y < m$ （实际上枚举的数变为了 $1 \sim C$ ，这没有什么影响）。移项后我们可以得到 $A^{im} \equiv A^y B(\text{mod } C)$ ，因此我们枚举 $0 \sim m-1$ 将所有 $A^y B(\text{mod } C)$ 的值存进哈希表中，再枚举 $1 \sim n$ 计算 A^{im} 的值，在哈希表中查询是否有这个值，进而得出解。复杂度为 $O(\sqrt{C})$ 。

上面的做法适用于 $\text{gcd}(A, C) = 1$ 的情况，即 $(\text{mod } C)$ 意义下存在 A 的逆元，若不存在的话则做法是错误的，因为此时 a^{-1} 不存在，就不能将 A^x 看做是 $A^{im} \times A^{-y}$ 。

解决方法是预处理 A, C 成互质，保证逆元存在。

将 $A^x \equiv B(\text{mod } C)$ 看做是 $A^x + Cy = B$ 方便叙述与处理。

我们将方程一直除去 A, C 的最大公约数进行变形，最终使得 A 和 C 互质。

将方程同除 $d_1 = \text{gcd}(A, C)$ ，得到 $B_1 = \frac{A}{d_1} A^{x-1} + C_1 y$ 。有可能 A 和 C_1 不互素，因此继

续将方程同除 $d_2 = \text{gcd}(A, C_1)$ 得到 $B_2 = \frac{A^2}{d_1 d_2} A^{i-2} + C_2 y$ 。一直这样下去直到 A 和 C_i 互素。

这里也能看出，若 B_i 不被 $\text{gcd}(A, C_i)$ 整除则无解。

最终得到 $B_n = \frac{A^n}{d_1 d_2 \dots d_n} A^{x-n} + C_n y$ ，并记 $D = \frac{A^n}{d_1 d_2 \dots d_n}$ ，易证明 $\text{gcd}(D, C_n) = 1$ ，因

此存在 D 的逆元，可以将最后的式子变为 $A^{x-n} \equiv B_n \cdot D^{-1}(\text{mod } C_n)$ ，此时就能求解了。

例题

Clever Y (POJ 3243)

题目大意：

给定 X, Z, K ，求 $X^Y \equiv K(\text{mod } Z)$ 的最小非负整数解。

分析:

BSGS 模板题, 注意 $\gcd(X, Z)$ 不一定为 1.

代码:

```
1. #include <cmath>
2. #include <cstdio>
3.
4. int X, Y, Z, K;
5.
6. int Gcd(int a, int b) { return !b ? a : Gcd(b, a % b); }
7.
8. int exGcd(int a, int b, int &x, int &y) {
9.     if (b == 0) return x = 1, y = 0, a;
10.    int r = exGcd(b, a % b, y, x);
11.    y -= (long long)(a / b) * x;
12.    return r;
13. }
14.
15. int inv(int a, int m) {
16.    int x, y;
17.    exGcd(a, m, x, y);
18.    return (x % m + m) % m;
19. }
20.
21. namespace Hash {
22.    const int N = 50000;
23.    const int H = 999979;
24.    int tot, adj[H], nxt[N], num[N], val[N];
25.    int top, stk[N];
26.
27.    void init() {
28.        tot = 0;
29.        while (top) adj[stk[top--]] = 0;
30.    }
31.
32.    void insert(int x, int y) {
33.        int h = x % H;
34.        for (int e = adj[h]; e; e = nxt[e]) {
35.            if (num[e] == x) {
36.                val[e] = y;
37.                return ;
38.            }
39.        }
```

```

40.     if (!adj[h]) stk[++top] = h;
41.     nxt[++tot] = adj[h], adj[h] = tot;
42.     num[tot] = x, val[tot] = y;
43. }
44.
45. int query(int x) {
46.     int h = x % H;
47.     for (int e = adj[h]; e; e = nxt[e])
48.         if (num[e] == x) return val[e];
49.     return -1;
50. }
51. }
52.
53. // a^x = b (mod c)
54. int BSGS(int a, int b, int c) {
55.     int cnt = 0, G, d = 1;
56.     while ((G = Gcd(a, c)) != 1) {
57.         if (b % G != 0) return -1;
58.         ++cnt, b /= G, c /= G;
59.         d = (long long)d * (a / G) % c;
60.     }
61.     b = (long long)b * inv(d, c) % c;
62.
63.     Hash::init();
64.     int s = sqrt(c);
65.     int p = 1;
66.     for (int i = 0; i < s; ++i) {
67.         if (p == b) return i + cnt;
68.         Hash::insert((long long)p * b % c, i);
69.         p = (long long)p * a % c;
70.     }
71.     int q = p, t;
72.     for (int i = s; i - s + 1 <= c - 1; i += s) {
73.         t = Hash::query(q);
74.         if (t != -1) return i - t + cnt;
75.         q = (long long)q * p % c;
76.     }
77.     return -1;
78. }
79.
80. bool check() {
81.     for (int i = 0, j = 1; i <= 10; ++i) {
82.         if (j == K) {
83.             printf("%d\n", i);

```

```
84.     return true;
85. }
86. j = (long long)j * X % Z;
87. }
88. if (X == 0) {
89.     puts("No Solution");
90.     return true;
91. }
92. return false;
93. }
94.
95. int main() {
96.     //  $X^Y = K \pmod Z$ 
97.     while (scanf("%d%d%d", &X, &Z, &K), (long long)X + Z + K > 0) {
98.         X %= Z, K %= Z;
99.         if (check()) continue;
100.        int ans = BSGS(X, K, Z);
101.        if (ans == -1) puts("No Solution");
102.        else printf("%d\n", ans);
103.    }
104.    return 0;
105. }
```

原根

阶

定义 1

设 $n > 1$, a 是满足 $(a, n) = 1$ 的整数, 则必有一个 $r (1 \leq r \leq n)$ 使得 $a^r \equiv 1 \pmod{n}$

满足 $a^r \equiv 1 \pmod{n}$ 的最小整数 r , 称为 a 模 n 的阶, 记为 $Ord_n(a)$ 。

注意, 只有 $(a, n) = 1$ 情况下才有阶, 否则对于 $a^x \equiv 1 \pmod{n}$, x 无解, 即不存在阶。

阶的性质

(1) 设 $(a, n) = 1$, a 模 n 的阶为 r . 若正整数 N 使得 $a^N \equiv 1 \pmod{n}$, 则 $r \mid N$ 。

(2) 设 $(a, n) = 1$, 则 a 模 n 的阶 r 整除 $\varphi(n)$. 特别的, 若 n 是素数 p , 则 a 模 p 的阶整除 $p-1$ (素数 p 的欧拉函数值为 $p-1$)。

原根

定义 2

设 m 是正整数, a 是整数, 若 a 模 m 的阶等于 $\varphi(m)$, 则称 a 为模 m 的一个原根。

显然, 由阶的定义可知原根和 m 必然互质。

解性

原根是一个模 m 的剩余类, 因此我们只要研究在 $0 < x < m$ 范围内的解即可。

性质

记 $\delta = Ord_m(a)$, 则 $a^0, a^1, a^2, \dots, a^{\delta-1}$ 模 m 两两不同余。因此当 a 是模 m 的原根时, $a^0, a^1, a^2, \dots, a^{\delta-1}$ 构成模 m 的简化剩余系, 反之亦然。特殊地当 m 为质数时, $a^0, a^1, a^2, \dots, a^{\delta-1}$ 对 m 取模后对应为 $\{1, 2, \dots, m-1\}$

求质数 p 的原根算法

算法说明

原根的分布比较广，并且最小的原根通常也比较小，故可以通过从小到大枚举正整数来快速地寻找一个原根。对于一个待检查的 p ，对于 $p-1$ 个每一个素因子 a ，检查 $g^{\frac{p-1}{a}} \equiv 1 \pmod{p}$ 是否成立，如果成立则说明 g 不是原根。

N 次剩余

目标

我们已经会解同余方程 $ax \equiv b \pmod{c}$ ， $a^x \equiv b \pmod{c}$ ，那么我们本节来研究 $x^N \equiv a \pmod{c}$ 的解 x 。由于该方程较复杂，我们只考虑 c 为素数的情况。

方程 $x^N \equiv a \pmod{p}$ 的解性

显然，解出的 x 是模 p 意义下的剩余类。

算法

令 g 为 p 的原根，因为 p 为素数，则 $\varphi(p) = p-1$ ，所以找到原根 g 就可以将 $\{1, 2, \dots, p-1\}$ 的数与 $\{g^1, g^2, \dots, g^{p-1}\}$ 建立一一对应关系。

令 $g^y = x, g^t = a$ ，则有 $g^{y \times N} \equiv g^t \pmod{p}$

因为 p 是素数，所以方程左右都不可以为 0。这样就可以将这 $p-1$ 个取值与指数建立对应关系。原问题转化为： $N \times y \equiv t \pmod{p-1}$

对于 y 解模线性方程就可以解决。而 $g^t = a$ 则可以用解离散对数的方法求出。

练习题

POJ 1284 ; BZOJ 2242 ; HDU 3223