同余与扩展欧几里得

裴蜀定理 + exgcd 通解与最小非负解 + 乘法逆元

HGNU-ACM 实验室算法组

零基础精讲：从第 2 章辗转相除出发，彻底掌握线性同余方程与模意义除法





本章目录

模运算、同余与裴蜀定理

扩展欧几里得 exgcd 与通解

模意义下的“除法”：乘法逆元

前三节逐层打通基础：模运算与裴蜀定理（判有解）、扩展欧几里得 exgcd（求特解与最小非负解）、乘法逆元（解决模运算除法）。

本章目录（续）

例题精讲：从建模到代码落地

常见错误、检查清单与练习安排

后两节精讲 3 道真题《青蛙的约会》《同余方程》《计算器》，并给出高频错因对照、六必查清单与第 4 章预告。

生活引入：钟表上的加法和“绕圈”

先看一个手算小例子

现在是上午 10 点，再过 5 小时是几点？再过 29 小时呢？

- $10+5=15$，时钟每 12 小时绕一圈，所以 $15-12=3$ 点。
- $10+29=39$，$39=3\times 12+3$，依然是 3 点！
- 在数学里，我们只关心“除以 12 后的余数”，记作 $15\equiv 3 \pmod{12}$。
- 新术语（同余）：如果两个整数 a, b 除以正整数 m 的余数相同（等价于 $m \mid (a-b)$），就说 a 和 b 模 m 同余，写成 $a\equiv b \pmod{m}$。

本页要点

同余就是“只看余数，绕圈后落在同一个刻度上”，$a\equiv b \pmod{m}$ 等价于 $a-b$ 是 m 的整数倍。

为什么不能算完再取模？兼谈负数取模

- 竞赛题常让求“方案数对 10^9+7 取模”。若先把 $100!$ 算完再取模，数字早就超出 `long long`（约 9×10^{18}）变成垃圾值！
- 好消息：加、减、乘满足“边算边取模”：

$$(a+b)\bmod m=((a\bmod m)+(b\bmod m))\bmod m$$

$$(a\times b)\bmod m=((a\bmod m)\times(b\bmod m))\bmod m$$

- 负数陷阱：在 C++ 中，`-7 % 5` 的结果是 `-2` 而不是 `3`！
- 若想把余数稳定修正到 $[0,m-1]$，必须写成 `(x % m + m) % m`。

本页要点

加减乘可以随时取模防溢出；遇到减法或负数，务必用 `(x % m + m) % m` 把余数拉回非负区间。

从水壶倒水看线性不定方程 $ax + by = c$

生活场景：两个水壶能倒出几升水？

你手里有一个 4 升水壶和一个 6 升水壶（水源无限，水可以倒掉），只用这两个水壶，能不能刚好量出 2 升水？能不能刚好量出 3 升水？

- 每次往里倒 4 升或 6 升，或者倒出，最终水量一定能写成：

$$4x + 6y = c \quad (x, y \text{ 为整数，正数表示倒进，负数表示倒出})$$

- 取 $x = -1, y = 1$：$4(-1) + 6(1) = 2$，所以 2 升可以量出！
- 不管 x, y 怎么选，$4x$ 和 $6y$ 都是偶数，它们的和 $4x + 6y$ 永远是偶数，不可能等于奇数 3！

本页要点

水壶问题本质是在问：关于整数 x, y 的方程 $ax + by = c$ 到底有没有整数解。

为什么不能用循环暴力枚举 (x, y)？

- 看到 $ax + by = c$，初学者常想：用一个 `for` 循环枚举 $x = 0, 1, 2, \ldots$，看看 $(c - ax)$ 能不能被 b 整除不就行了？
- 来算算操作次数：在第 3 章的例题《青蛙的约会》中，跑道长度 L 可达 2×10^9。
- 最小非负解 x 最多可能接近 $L = 2 \times 10^9$！
- 单层循环跑 2×10^9 次，按每秒约 10^8 次运算估算，需要约 20 秒，必定 **TLE**（超时）；如果答案根本无解，暴力循环甚至不知道什么时候该停！

本页要点

枚举 x 的暴力做法时间复杂度是 $O(L)$，面对 10^9 级别数据必 TLE，而且无法快速判定无解。

裴蜀定理：什么时候有整数解？

裴蜀定理（Bézout's Identity）

对于任意整数 a, b, c（a, b 不全为 0），关于整数 x, y 的方程

$$ax + by = c$$

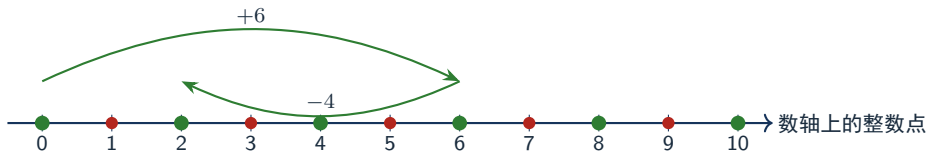
存在整数解，当且仅当 $\gcd(a, b) \mid c$（即 c 能被 a 和 b 的最大公约数整除）。

- 承接第 2 章：$\gcd(a, b)$ 就是 a 和 b 共同拥有的最大“积木块”。
- 回看刚才的水壶问题：$\gcd(4, 6) = 2$。
- 因为 $2 \mid 2$，所以 $4x + 6y = 2$ 有解；因为 $2 \nmid 3$，所以 $4x + 6y = 3$ 无解！

本页要点

用第 2 章学过的辗转相除算出 $g = \gcd(a, b)$，只需一行 `c % g == 0` 就能在 $O(\log n)$ 内判定有解还是无解！

图解裴蜀定理：格子间距就是 $g = \gcd(a, b)$



绿色点：$4x + 6y$ 能到达的所有位置（都

红色点：奇数 $1, 3, 5, \ldots$ 永远无法到达

- 无论怎么组合 a 步和 b 步，落脚点只能落在间距为 $g = \gcd(a, b)$ 的网格点上。

本页要点

$ax + by$ 能凑出的所有整数，恰好就是 $\gcd(a, b)$ 的全部整数倍，不多也不少。

为什么裴蜀定理一定对？（整除性与最小正整数）

论证方向	核心推导逻辑（以 a, b 不全为 0 为前提）
必要性（有解 $\Rightarrow$ $g \mid c$）	设 $g = \gcd(a, b)$，则 $g \mid a$ 且 $g \mid b$。若存在整数 x, y 使 $ax + by = c$，由整除性质，左边是 g 的倍数，所以右边 c 也必须是 g 的倍数。
充分性（$g \mid c \Rightarrow$ 有解）	在所有能写成 $ax+by$ 的正整数里，取最小的那一个，记作 $d = ax_0 + by_0$。用带余除法写 $a = qd + r$（$0 \le r < d$），则 $r = a - q(ax_0 + by_0) = a(1 - qx_0) + b(-qy_0)$。若 $r > 0$，它比最小正整数 d 还小，矛盾！故 $r = 0$，即 $d \mid a$；同理 $d \mid b$。所以 d 是公约数，必有 $d \le g$；又因 $g \mid d$，只能 $d = g$！既然能凑出 g，两边同乘 c/g 就凑出了 c。

本页要点

充分性的灵魂在于“最小正线性组合 d 一定整除 a 和 b”，从而 d 就是最大公约数 g。

上机验证：用 gcd 秒判 $ax + by = c$ 是否有解

```cpp
#include <iostream>
#include <numeric>
using namespace std;
int main() {
    long long a, b, c; cin >> a >> b >> c;
    long long g = std::gcd(a, b);        // C++17 自带最大公约数
    cout << (c % g == 0 ? "YES\n" : "NO\n");
    return 0;
}
```

- 三组验证：`4 6 2` → `YES`；`4 6 3` → `NO`；`0 5 10` → `YES`（$\gcd(0,5)=5 \mid 10$）。
- 反事实追问：输入 `0 0 5` 时 $\gcd(0,0)=0$，执行 `c % g` 会怎样？（除以 0 触发 **RE**！允许 $a=b=0$ 需特判 $c==0$）。

本页要点

利用 C++17 的 `std::gcd` 可快速验证裴蜀定理，但 $a=b=0$ 时需警惕模 0 导致的 RE。

本节易错对照：同余与裴蜀定理

错误写法	正确写法	导致的评测结果与原因
`x % m` 直接输出	`(x % m + m) % m`	**WA**：当 $x < 0$ 时 C++ 取模得到负数，不符合余数 $\in [0, m-1]$ 的要求。
不判 `c % g != 0`	先判 `if (c % g != 0)` 输出无解	**WA**：无解时程序硬往下算，把本该输出 `Impossible` 的点输出一个数字。
暴力循环找 x	用 exgcd $O(\log n)$ 求解	**TLE**：当模数 $L \approx 2 \times 10^9$ 时，枚举 x 远超 1 秒时限。
`a * b % m` 直接乘	`(long long)a * b % m`	**WA**：两个 10^9 级别的 `int` 相乘先溢出再取模，结果变成负数。

本页要点

先判 `c % g == 0`，乘法先转 `long long`，负数取模用 `(x % m + m) % m`。

承接第 2 章：光知道“有解”还不够，怎么找 (x, y)？

- 第 2 章学过辗转相除法（欧几里得算法）：

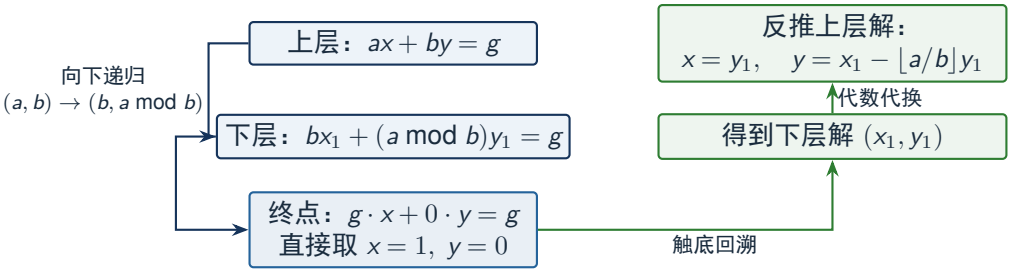
$$\gcd(a, b) = \gcd(b,\ a \bmod b)$$

- 它能迅速算出最大公约数 g，判断 $ax + by = c$ 是否有解。
- 但它只管“向下除”，到 $b = 0$ 时返回 g 就结束了，并没有告诉我们 x 和 y 是多少！
- 新术语（扩展欧几里得 `exgcd`）：在辗转相除向下递归的过程中，顺便在 ** 回溯返回 ** 时把整数系数 x, y 一步步反推出来，求出 $ax + by = \gcd(a, b)$ 的一组整数特解。

本页要点

普通 `gcd` 只算最大公约数；`exgcd` 在递归回溯的归途中把系数 (x, y) 一并讨回来。

图解 exgcd：下探求 g，回溯反推 (x, y)



本页要点

先一路递归到 $b = 0$ 拿到最简单的解 $(1, 0)$，再顺着调用栈一层层代换公式往上推。

核心公式推导：为什么 $x = y_1,\ y = x_1 - \lfloor a/b \rfloor y_1$？

- 假设递归下一层已经帮我们算出了方程的一组解 (x_1, y_1)，满足：

$$bx_1 + (a \bmod b)y_1 = g$$

- 回想整数取模的定义：$a \bmod b = a - \lfloor a/b \rfloor \cdot b$（其中 $\lfloor a/b \rfloor$ 就是 C++ 里的整除 `a / b`）。
- 把它代回下层方程并按 a 和 b 重新合并同类项：

$$bx_1 + \big(a - \lfloor a/b \rfloor b\big)y_1 = g \implies a \cdot \mathbf{y_1} + b \cdot (\mathbf{x_1} - \lfloor \mathbf{a}/\mathbf{b} \rfloor \mathbf{y_1}) = g$$

- 对比上层方程 $a \cdot x + b \cdot y = g$，立刻得出：

$$x = y_1, \qquad y = x_1 - \lfloor a/b \rfloor y_1$$

本页要点

利用 $a \bmod b = a - \lfloor a/b \rfloor b$ 展开并合并 a, b 项，就能直接读出上层系数 x 和 y。

手算跟踪表（一）：exgcd(14，6）递归下探过程

求 $14x+6y=\gcd(14,6)$，先跟踪每次向下调用的参数：

层数	当前 a	当前 b	$\lfloor a/b \rfloor$	下一步动作
第 1 层	14	6	2	计算 $14 \bmod 6=2$，递归 exgcd(6，2)
第 2 层	6	2	3	计算 $6 \bmod 2=0$，递归 exgcd(2，0)
第 3 层（终止）	2	0	—	触发 b == 0，令 $x=1,y=0$，返回 $g=2$

- 注意第 3 层就是递归终止条件：此时方程是 $2x+0y=2$，取 $x=1,y=0$ 显然成立（取 $y=0$ 最简单且不会溢出）。

本页要点

下探阶段只做取模并把 a / b 留在栈里备用，直到 $b=0$ 时触底赋值 $(1,0)$。

手算跟踪表（二）：exgcd(14，6）回溯反推过程

现在带着第 3 层的返回值，自底向上逐层套用 $x = y_1,\ y = x_1 - \lfloor a/b \rfloor y_1$：

层数	(a, b)	下层 (x_1, y_1)	$\lfloor a/b \rfloor$	新 $x = y_1$	新 $y = x_1 - \lfloor a/b \rfloor y_1$	验算 $ax + by$
第 3 层	$(2, 0)$	—	—	1	0	$2(1) + 0(0) = 2$
第 2 层	$(6, 2)$	$(1, 0)$	3	0	$1 - 3 \times 0 = 1$	$6(0) + 2(1) = 2$
第 1 层	$(14, 6)$	$(0, 1)$	2	1	$0 - 2 \times 1 = -2$	$14(1) + 6(-2) = 2$

- 最终第 1 层返回 $g = 2$，同时得到特解 $x = 1, y = -2$。验算：$14 \times 1 + 6 \times (-2) = 14 - 12 = 2$，完全正确！

本页要点

每层回溯都保持 $ax + by = g$ 成立，最终回到顶层就得到原方程的特解。

为什么 exgcd 一定对？（递归不变量论证）

证明环节	不变量：$ax+by=\gcd(a,b)$
终止步（基础）	当 $b=0$ 时，$\gcd(a,0)=a$（前提 $a\geq 0$）。代码令 $x=1,y=0$，左边 $a\cdot 1+0\cdot 0=a=\gcd(a,0)$，成立。
保持步（归纳）	假设下层调用 `exgcd(b, a % b, x1, y1)` 返回了正确的 $g=\gcd(b,a \bmod b)$ 且满足 $bx_1+(a \bmod b)y_1=g$。代入 $a \bmod b=a-\lfloor a/b\rfloor b$ 得 $ay_1+b(x_1-\lfloor a/b\rfloor y_1)=g$，故更新后的 x,y 仍满足 $ax+by=g$。
结束（终止性）	每次递归第二个参数严格变小（$0\leq a \bmod b<b$），由良序性必然在 $O(\log b)$ 步内到达 $b=0$ 并返回。

本页要点

数学归纳法三步严丝合缝，前提是传入的 a,b 非负（若 $a<0$ 需提前将方程两边取反）。

exgcd 标准模板代码与“为什么这样写”

```
long long exgcd(long long a, long long b, long long &x, long long &y) {
    if (b == 0) { x = 1; y = 0; return a; } // 递归终点：ax + 0y = a
    long long x1, y1;                       // 接收下层返回的系数
    long long g = exgcd(b, a % b, x1, y1); // 向下递归求 gcd(b, a%b)
    x = y1;                                 // 回溯更新当前层 x
    y = x1 - (a / b) * y1;                  // 回溯更新当前层 y
    return g;                               // 返回最大公约数 g
}
```

- 为什么传引用 `&x`，`&y`？函数返回值只能带回一个数 g，用引用参数才能同时把 x, y 带回上层。
- 为什么用局部变量 `x1`，`y1`？防止覆盖当前层的计算顺序，写起来最不容易错。

本页要点

exgcd 只有 7 行核心代码，记住“终点 `1, 0`，回溯 `x = y1, y = x1 - (a/b)*y1`”。

已知一组特解，怎么求出 $ax + by = c$ 的所有解？

- 第一步：用 `exgcd(a, b, x0, y0)` 求出 $ax_0 + by_0 = g$。
- 第二步：若 $c \bmod g \neq 0$，无解；否则方程两边同乘 c/g，得到 $ax + by = c$ 的一组特解：

$$x^* = x_0 \cdot \frac{c}{g}, \qquad y^* = y_0 \cdot \frac{c}{g}$$

- 第三步：如何得到全部整数解（通解）？我们想给 x^* 加上一个正步长 Δx，同时给 y^* 减去 Δy，使总和不变：

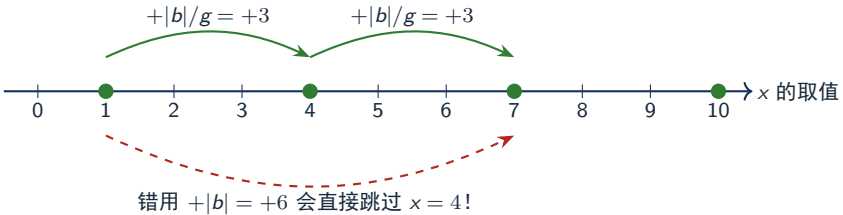
$$a(x^* + \Delta x) + b(y^* - \Delta y) = c \implies a\Delta x = b\Delta y$$

- 两边同除以 g：$\frac{a}{g}\Delta x = \frac{b}{g}\Delta y$。因为 $\frac{a}{g}$ 和 $\frac{b}{g}$ 互质，所以 Δx 必须是 $\frac{b}{g}$ 的整数倍！

本页要点

特解先乘 c/g 扩大到右边等于 c；通解里 x 每次变化的最小步长是 $|b|/g$，而不是 $|b|$！

图解步长为什么是 $|b|/g$ 而不是 $|b|$



- 例如 $14x+6y=2$（$g=2$），步长是 $6/2=3$。特解 $x^*=1$，所有解为 $x=1+3k$（如 $1,4,7,\ldots$ 和 $-2,-5,\ldots$）。
- 如果把模数错写成 $b=6$，从 $x=7$ 往左减 6 只能得到 1，遇到 $x=4$ 这样的解就会彻底漏掉！

本页要点

求 x 的最小非负整数解：令 `mod = abs(b) / g`，结果为 `(x_star % mod + mod) % mod`。

扩大 c/g 倍时的防溢出警告：__int128

- 在很多题目中（例如《青蛙的约会》），输入 a, b, c 可达 2×10^9。
- 此时 exgcd 算出的特解 x_0 绝对值最大可达 $|b| \approx 2 \times 10^9$，而 c/g 也可达 2×10^9。
- 执行 `x0 * (c / g)` 时，乘积最大可达 $(2 \times 10^9) \times (2 \times 10^9) = 4 \times 10^{18}$！
- 若题目数据达到 10^{10}，long long（上限约 9.22×10^{18}）极易溢出变负。
- 最佳实践：把 exgcd 中的 x, y 设为 `__int128_t`（简写 `using i128 = __int128_t;`），算完 `(x0 * (c / g) % mod + mod) % mod` 后再强转回 long long 输出！

本页要点

64 位评测机（GCC）原生支持 `__int128_t`，能轻松兜住 10^{38} 以内的乘法，防止特解放大时溢出。

上机验证：求 $ax + by = c$ 的最小非负整数解 x

```
// 接前页 exgcd，传入前先保证 a >= 0, b > 0
long long solve(long long a, long long b, long long c) {
    long long x0, y0;
    long long g = exgcd(a, b, x0, y0);
    if (c % g != 0) return -1;              // 无整数解
    long long mod = b / g;                  // x 的最小正周期
    __int128 xs = (__int128)x0 * (c / g);  // 放大 c/g 倍防溢出
    return (long long)((xs % mod + mod) % mod);
}
```

- 三组验证：$(14,\ 6,\ 2) \to 1$；$(14,\ 6,\ 4) \to 2$；$(4,\ 6,\ 3) \to -1$（无解）。
- 反事实追问：如果把 `mod = b / g` 错写成 `mod = b`，测试 `14x + 6y = 2` 时若特解恰好算出 $x^* = 4$，程序会输出什么？（会错误地输出 4 而不是真正的最小非负解 1!）。

本页要点

用 $(14,\ 6,\ 2)$ 和 $(4,\ 6,\ 3)$ 两组数据可快速检验“步长 b/g”与“无解判定”是否写对。

本节易错对照：exgcd 与通解

错误写法	正确写法	导致的评测结果与原因
`mod = b`	`mod = abs(b) / g`	**WA**：周期算大 g 倍，最小非负解可能错成较大的正整数。
把负数 a 直接传给 exgcd	提前 `if(a<0){a=-a; b=-b;}`	**WA**：算出负的 g，导致 mod 为负甚至方向反转。
`x0 * (c / g)` 用 `long long`	用 `__int128` 中转后再取模	**WA**：两项乘积逼近或超过 9×10^{18} 溢出变负。
忘记写 `c / g` 直接用 `x0`	`x0 * (c / g)`	**WA**：只解了 $ax+by=g$ 而非原方程 $ax+by=c$。

本页要点

牢记四步口诀：“系数先翻正，特解乘 c/g，周期模 |b|/g，乘法用 __int128”。

痛点引入：为什么模运算不能直接做除法？

先看一个手算小例子

算 $(6 \div 2) \bmod 5$：先除再取模，显然等于 $3 \bmod 5 = 3$。
那如果先对分子分母分别取模呢？$(6 \bmod 5) \div (2 \bmod 5) = 1 \div 2 = 0.5$，连整数都不是！

- 为什么？因为取模只对 ** 加法、减法、乘法 ** 保持封闭（分配律），对 ** 除法 ** 根本不成立！
- 在组合数 $C_n^k = \frac{n!}{k!(n-k)!} \pmod p$ 等问题中，$n!$ 动辄几百位，必须边算边取模，不能最后再除。
- 怎么办？把“除以 a”变成“乘以某个整数 x”，使它们在模 m 下的效果完全一样！

本页要点

模运算里没有直接的除法，需要找一个整数 x 来“冒充” $\frac{1}{a}$，把除法变成乘法。

暴力找逆元为何会 TLE？兼谈逆元定义

新术语：乘法逆元

如果存在整数 $x \in [0, m-1]$，满足 $a \cdot x \equiv 1 \pmod m$，就称 x 是 a 模 m 的乘法逆元，记作 a^{-1}。有了它，$(b \div a) \bmod m$ 就可以算成 $(b \times a^{-1}) \bmod m$！

- 验证例子：模 5 下，$2 \times 3 = 6 \equiv 1 \pmod 5$，所以 2 的逆元是 3。刚才的 $(6 \div 2) \bmod 5 = (6 \times 3) \bmod 5 = 18 \bmod 5 = 3$，完美吻合！
- 暴力做法为何 **TLE**？从 $x = 1$ 到 $m - 1$ 挨个试 `a * x % m == 1`。当 $m = 10^9 + 7$ 时，单次查询就要跑 10^9 次循环（约 10 秒），必然 **TLE**！
- 存在条件：$ax \equiv 1 \pmod m \iff ax + my = 1$。由裴蜀定理，逆元存在的 ** 充要条件 ** 是 $\gcd(a, m) = 1$（a 与 m 互质）！

本页要点

a 在模 m 下有逆元当且仅当 $\gcd(a, m) = 1$；模数为 $10^9 + 7$ 时暴力枚举必 TLE。

方法一：费马小定理 + 快速幂（质数模数专用）

费马小定理

如果 p 是 ** 质数 **，且 $p \nmid a$（即 a 不是 p 的倍数），那么：

$$a^{p-1} \equiv 1 \pmod{p} \quad \Longrightarrow \quad a \cdot a^{p-2} \equiv 1 \pmod{p}$$

- 对比逆元定义 $a \cdot x \equiv 1 \pmod{p}$，直接得到 $a^{-1} \equiv a^{p-2} \pmod{p}$！
- 承接第 1 章《快速幂》：用快速幂计算 `qpow(a, p - 2, p)`，时间复杂度仅需 $O(\log p)$（模 10^9+7 时只需约 30 次乘法）！
- 致命限制：要求 p 必须是 ** 质数 **（比如 10^9+7、998244353）。如果模数 m 是合数（如 $m=10$ 或 $m=10000$），费马小定理 ** 不成立 **！

本页要点

模数是质数时，用 `qpow(a, p - 2, p)` 是最快的写法；合数模数严禁套用费马小定理！

方法二：exgcd 求逆元（通吃质数与合数模数）

- 当模数 m 是 ** 合数 **（例如 $m = 10$，求 3 的逆元）时，费马小定理失效（$3^{10-2} \bmod 10 = 3^8 \bmod 10 = 1$ 只是巧合，试 $m = 9, a = 2$：$2^7 \bmod 9 = 128 \bmod 9 = 2 \neq 1$，而真正的逆元是 5!）。
- 这时用刚刚学过的 exgcd！把 $ax \equiv 1 \pmod m$ 改写为线性不定方程：

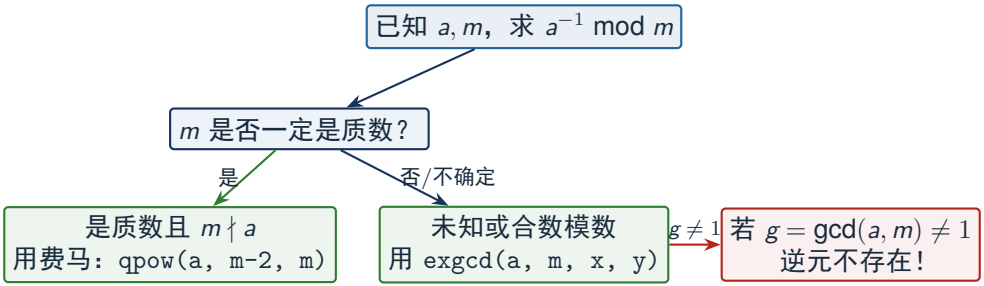
$$ax + my = 1$$

- 调用 `exgcd(a, m, x, y)`：
  - 若返回的 $g \neq 1$，说明 $\gcd(a, m) \neq 1$，逆元 ** 不存在 **；
  - 若 $g = 1$，逆元就是 `(x % m + m) % m`，复杂度同样是 $O(\log m)$！

本页要点

exgcd 只需 $\gcd(a, m) = 1$ 即可求逆元，既能算合数模数，还能顺手判定逆元存不存在。

图解决策：求单个逆元该选哪种方法？



本页要点

质数模数写费马快速幂最短；模数不保证质数或需要判无解时，一律用 exgcd。

方法三：线性递推 $O(n)$ 批量求 $1 \sim n$ 的全部逆元

- 如果题目要频繁用到 $1, 2, \ldots, n$（如 $n = 10^6$）每个数模质数 p 的逆元，对每个数都跑一次 $O(\log p)$ 快速幂需要 3×10^7 次运算，偏慢。
- 线性递推公式：设 p 为质数，$1 < i < p$，令 $k = \lfloor p/i \rfloor$，$r = p \bmod i$，则有：

$$k \cdot i + r = p \equiv 0 \pmod{p}$$

- 两边同时乘以 $i^{-1} \cdot r^{-1}$（因为 $0 < r < i < p$ 且 p 是质数，它们都有逆元）：

$$k \cdot r^{-1} + i^{-1} \equiv 0 \pmod{p} \implies i^{-1} \equiv -k \cdot r^{-1} \equiv (p - \lfloor p/i \rfloor) \cdot \mathrm{inv}[p \bmod i] \pmod{p}$$

- 递推边界：$\mathrm{inv}[1] = 1$；注意 $p \bmod i < i$，所以算到 i 时 $\mathrm{inv}[p \bmod i]$ 一定已经算好！

本页要点

用 `inv[i] = (p - p / i) * inv[p % i] % p`，每个数只需 $O(1)$ 时间就能推出 $1 \sim n$ 全部逆元！

手算跟踪表：用线性递推求模 $p = 7$ 下 $1 \sim 5$ 的逆元

已知边界 $\mathrm{inv}[1] = 1$，依次计算 $i = 2, 3, 4, 5$：

i	$k = \lfloor 7/i \rfloor$	$r = 7 \bmod i$	$\mathrm{inv}[r]$	$\mathrm{inv}[i] = (7-k) \cdot \mathrm{inv}[r] \bmod 7$	验算 $i \cdot \mathrm{inv}[i] \bmod 7$
1	—	—	—	1（初值）	$1 \times 1 \bmod 7 = 1$
2	3	1	$\mathrm{inv}[1] = 1$	$(7-3) \times 1 \bmod 7 = 4$	$2 \times 4 = 8 \equiv 1$
3	2	1	$\mathrm{inv}[1] = 1$	$(7-2) \times 1 \bmod 7 = 5$	$3 \times 5 = 15 \equiv 1$
4	1	3	$\mathrm{inv}[3] = 5$	$(7-1) \times 5 \bmod 7 = 30 \bmod 7 = 2$	$4 \times 2 = 8 \equiv 1$
5	1	2	$\mathrm{inv}[2] = 4$	$(7-1) \times 4 \bmod 7 = 24 \bmod 7 = 3$	$5 \times 3 = 15 \equiv 1$

本页要点

注意 $i = 4$ 用到了刚算出的 $\mathrm{inv}[3] = 5$，严格从小到大递推就不会依赖未计算的值。

正确性论证：为什么线性递推公式不会出错？

论证步骤	逻辑依据（前提：p 为质数且 $1 \le i < p$）
基础步（$i = 1$）	$1 \times 1 = 1 \equiv 1 \pmod p$，故 $\mathrm{inv}[1] = 1$ 必对。
保持步（归纳）	对任意 $2 \le i < p$，由于 p 是质数，$p \bmod i \neq 0$，且 $1 \le r = p \bmod i < i$。由归纳假设，$\mathrm{inv}[r]$ 已经正确。从 $k \cdot i + r = p \equiv 0 \pmod p$ 同乘 $i^{-1} r^{-1}$ 得 $i^{-1} \equiv (p - k) \cdot r^{-1} \pmod p$。
防负数与防爆	因为 $k = \lfloor p/i \rfloor < p$，所以 $p - k > 0$ 永不为负；只需把 `(p - p / i)` 转成 `long long` 再乘 `inv[p % i]`，就不会溢出且结果落在 $[1, p-1]$。

本页要点

因为 p 是质数，$p \bmod i$ 永远大于 0 且严格小于 i，保证递推链不断裂。

模板代码：exgcd 求逆元与线性递推求逆元

```cpp
// 1) exgcd 求单个逆元（支持合数模数，无解返回 -1）
long long inv_exgcd(long long a, long long m) {
    long long x, y;
    long long g = exgcd(a, m, x, y);
    return (g == 1) ? (x % m + m) % m : -1;
}
// 2) 线性递推求 1..n 模质数 p 的逆元（O(n)）
void init_inv(int n, long long p, vector<long long> &inv) {
    inv[1] = 1;
    for (int i = 2; i <= n; ++i)
        inv[i] = (p - p / i) * inv[p % i] % p; // p-p/i 保证正数
}
```

本页要点

单个逆元（尤其合数模数）用 inv_exgcd，批量求 $1 \sim n$ 模质数逆元用 init_inv。

上机验证：对比费马小定理与 exgcd 的输出

```cpp
int main() {
    long long a, m; cin >> a >> m;
    cout << "exgcd: " << inv_exgcd(a, m) << '\n';
    return 0; // 若套费马小定理：qpow(a, m - 2, m)
}
```

- 三组验证：3 7（质数模数）$\to 5$；2 9（合数模数，互质）$\to 5$；4 6（不互质）$\to -1$。
- 反事实追问：对第 2 组 2 9 强行套费马小定理算 $2^{9-2} \bmod 9$ 会输出什么？（$128 \bmod 9 = \mathbf{2}$，而 $2 \times 2 = 4 \not\equiv 1 \pmod 9$，直接 **WA**！）。

本页要点

用（2，9）和（4，6）两组测试数据，能一眼验出“合数模数”与“不互质无解”的处理是否正确。

本节易错对照：乘法逆元

错误写法	正确写法	导致的评测结果与原因
合数模数用费马小定理	改用 `exgcd(a, m, x, y)`	**WA**：费马小定理仅对质数模数成立。
a 是 p 的倍数还用费马	先判 `a % p != 0`	**WA**：$a \equiv 0 \pmod p$ 时逆元不存在，费马会算出 0。
线性递推用 `int` 相乘	`(p - p / i)` 用 `long long`	**WA**：当 $p = 10^9 + 7$ 时两项相乘会爆 `int`。
线性递推漏写 `inv[1] = 1`	循环前先赋值 `inv[1] = 1`	**WA**：`inv` 全为 0，导致后续递推全部算出 0。

本页要点

使用费马小定理前先确认“p 是质数”且“a 不是 p 的倍数”；批量递推别忘 `inv[1]=1`。

例 3.1（[基础] 青蛙的约会｜一本通 1631）：先读懂要求

原题题意（ ）

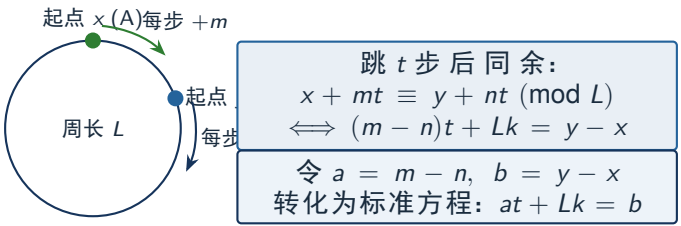
长为 L 的环形纬度线上，青蛙 A 从位置 x 出发，每次向西跳 m 米；青蛙 B 从位置 y 出发，每次向西跳 n 米。两只青蛙同时起跳且每次花费时间相同。问最少跳多少次后两只青蛙恰好落在同一点？若永远不能碰面，输出 `Impossible`。

- 输入：一行 5 个整数 x, y, m, n, L（$0 \le x, y < 2 \times 10^9,\ 0 < m, n, L < 2 \times 10^9$，$x \neq y$）。
- 输出：最少跳跃次数（非负整数）或 `Impossible`。
- 原题样例：输入 `1 2 3 4 5` $\Longrightarrow$ 输出 `4`。

本页要点

环形跑道上的“落在同一点”，翻译成数学语言就是“坐标对 L 取模后相等”。

例 3.1 | 怎么想到的：环形追及到线性同余方程



- 暴力模拟一步步跳最多要跳 $L \approx 2 \times 10^9$ 次，必 TLE；化成 $at + Lk = b$ 后可直接用 exgcd $O(\log L)$ 求解！

本页要点

设跳了 t 步相遇，直接列出 $x + mt \equiv y + nt \pmod{L}$，移项即得标准的 $at + Lk = b$。

例 3.1 | 样例手推跟踪表（1 2 3 4 5）

代入 $x=1, y=2, m=3, n=4, L=5$：

$a=m-n=-1,\ b=y-x=1 \xrightarrow{\text{把}a\text{翻正}} a=1,\ b=-1,\ L=5$，解方程 $1\cdot t+5\cdot k=-1$。

步骤	(a, L)	(t_0, k_0)	返回 g	计算与说明
调用 1	$(1,5)$	—	—	$1 \bmod 5=1$，向下调用 `exgcd(5, 1)`
调用 2	$(5,1)$	—	—	$5 \bmod 1=0$，向下调用 `exgcd(1, 0)`
终止层	$(1,0)$	$(1,0)$	1	触发 `b == 0`，直接赋值 $t_0=1, k_0=0$
回溯 2	$(5,1)$	$(0,1)$	1	$t_0=0,\ k_0=1-\lfloor 5/1\rfloor\cdot 0=1$
回溯 1	$(1,5)$	$(1,0)$	1	$t_0=1,\ k_0=0-\lfloor 1/5\rfloor\cdot 1=0$

- 验算：$b \bmod g=(-1) \bmod 1=0$（有解）。周期 $\text{mod}=L/g=5$，最小非负解 $(1\times(-1) \bmod 5+5) \bmod 5=\mathbf{4}$！

本页要点

遇到 $m-n<0$ 时必须将 a 和 b 同时取反，保证传入 exgcd 的系数非负。

例 3.1 | 核心步骤与相对模板改动

按顺序计算	纸上得到什么	核对什么问题
1. 算 $a=m-n,\ b=y-x$	方程左、右端系数	若 $a<0$，必须同时令 $a=-a, b=-b$
2. 调用 `exgcd(a, L, t0, k0)`	最大公约数 g 与特解 t_0	若 $b \bmod g \neq 0$，立刻输出 `Impossible`
3. 算 $\mathrm{mod}=L/g$	最小非负解周期	周期是 L/g，千万别写错成 L
4. 算 $(t_0 \cdot (b/g) \bmod \mathrm{mod}+\mathrm{mod}) \bmod \mathrm{mod}$	最少跳跃次数	t_0 与 b/g 相乘可能达 4×10^{18}，用 `__int128_t`

- 相对标准 `exgcd` 模板的改动：增加了 $a<0$ 翻正处理，并将引用参数 x, y 改为 `__int128_t` 防溢出。

本页要点

两处核心防坑：第一处是 $a<0$ 翻正，第二处是 `__int128_t` 承接 $t_0 \times (b/g)$。

例 3.1 | 核心代码（1/2）：exgcd 部分（code/frog_exgcd.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;

// 一本通 1631 青蛙的约会：化简成 (m-n)*t ≡ (y-x) (mod L)
using i128 = __int128_t;

long long exgcd(long long a, long long b, i128 &x, i128 &y) {
    if (b == 0) { x = 1; y = 0; return a; } // 递归终点
    i128 x1, y1;
    long long g = exgcd(b, a % b, x1, y1); // 向下求 gcd
    x = y1; y = x1 - (a / b) * y1;          // 回溯反推系数
    return g;
}
```

本页要点

使用 i128 作为 exgcd 的引用参数类型，从源头上杜绝中间结果溢出。

例 3.1 | 核心代码（2/2）：主函数部分（code/frog_exgcd.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    long long x, y, m, n, L;
    if (!(cin >> x >> y >> m >> n >> L)) return 0;
    long long a = m - n, b = y - x;
    if (a < 0) { a = -a; b = -b; }          // 保证系数 a 非负
    i128 t0, k0; long long g = exgcd(a, L, t0, k0);
    if (b % g != 0) { cout << "Impossible\n"; return 0; }
    long long mod = L / g;
    long long ans = (long long)((t0 * (b / g) % mod + mod) % mod);
    cout << ans << '\n'; return 0;
}
```

本页要点

当 $m = n$ 时 $a = 0$，返回 $g = L$；因 $x \neq y$ 且 $0 < |y - x| < L$，故 $b \bmod L \neq 0$ 自动判无解，不会除以 0！

例 3.1 | 易错点与 3 组自测边界数据

自测输入	预期输出	测试意图与容易踩中的坑
`1 2 3 4 5`	`4`	普通样例：$m < n$ 出现负系数 $a = -1$，检验翻正与负余数修正。
`0 2 2 2 6`	`Impossible`	速度相同（$a = 0$）永不相遇，检验是否提前判 `b % g != 0` 退出。
`0 1 2000000000 1 1999999999`	`1999999998`	极限边界数据（接近 2×10^9），检验 `__int128_t` 防溢出是否生效。

- 常见错误：把 `mod = L / g` 写成 `L`；忘记写 `if (a < 0)` 导致 g 为负；`t0 * (b / g)` 未用 `__int128_t`。

本页要点

提交前必测“速度相等 $m = n$”和“2×10^9 极大值”两组边界点。

例 3.2（[基础] 同余方程 | 一本通 1632）：先读懂要求

原题题意（ ）

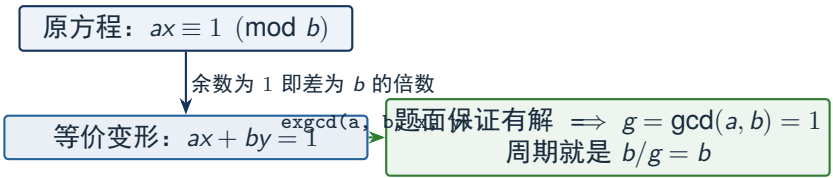
求关于 x 的同余方程 $ax \equiv 1 \pmod{b}$ 的最小正整数解。输入数据保证一定有解。

- 输入：一行两个正整数 a, b（$2 \le a, b \le 2 \times 10^9$）。
- 输出：一行一个正整数 x_0，即最小正整数解。
- 原题样例：输入 3 10 $\Longrightarrow$ 输出 7（因为 $3 \times 7 = 21 \equiv 1 \pmod{10}$）。

本页要点

注意题目求的是“最小正整数解”，且 b 未必是质数，不能用费马小定理，必须用 exgcd！

例 3.2 | 怎么想到的：同余方程与求逆元等价



- 这道题其实就是求 a 模合数 b 的乘法逆元！
- 因为 $b \le 2 \times 10^9$，暴力枚举 $x = 1 \dots b-1$ 必 TLE；而用 exgcd 只需要约 30 步递归就能出解。

本页要点

把 $ax \equiv 1 \pmod b$ 改写成 $ax + by = 1$，直接调用 `exgcd(a, b, x, y)` 即可。

例 3.2 | 样例手推跟踪表（3 10）

求解 $3x+10y=1$ 的 exgcd(3，10) 完整过程：

阶段	(a,b)	$\lfloor a/b \rfloor$	x	y	说明
下探 1	$(3,10)$	0	—	—	$3 \bmod 10 = 3$，调用 exgcd(10，3)
下探 2	$(10,3)$	3	—	—	$10 \bmod 3 = 1$，调用 exgcd(3，1)
下探 3	$(3,1)$	3	—	—	$3 \bmod 1 = 0$，调用 exgcd(1，0)
终止步	$(1,0)$	—	1	0	b == 0 返回 $g=1$
回溯 3	$(3,1)$	3	0	$1-3\times 0=1$	验算：$3(0)+1(1)=1$
回溯 2	$(10,3)$	3	1	$0-3\times 1=-3$	验算：$10(1)+3(-3)=1$
回溯 1	$(3,10)$	0	-3	$1-0\times(-3)=1$	验算：$3(-3)+10(1)=1$

- 得到特解 $x=-3$，修正到正整数区间：(-3 % 10 + 10) % 10 = 7，与样例输出一致！

本页要点

当 $a<b$ 时第一层 $\lfloor a/b \rfloor = 0$ 会自动把 (a,b) 交换成 (b,a)，无需手动交换。

例 3.2 | 核心步骤与相对模板改动

按顺序计算	纸上得到什么	核对什么问题
1. 调用 `exgcd(a, b, x, y)`	特解 x	题面保证有解，故无需乘 c/g（因为 $c = g = 1$）
2. 算 `(x % b + b) % b`	最小非负解 $\in [0, b-1]$	因为 $b \geq 2$ 且 $a \cdot 0 \not\equiv 1 \pmod b$，结果必 > 0
3. 防守判断 `if (ans == 0)`	若为 0 则改为 b	确保万一出现非负解为 0 时能输出正整数

- 相对标准 `exgcd` 模板的改动：省去了 c/g 放大步骤，因为右边常数恰好就是 $1 = g$；x 绝对值不超过 b，普通 `long long` 即可装下。

本页要点

本题 $c = 1$ 且保证有解，直接对 `b` 取正余数即可，代码极其简洁。

例 3.2 | 核心代码（1/2）：exgcd 部分（code/congruence_eq.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;

// 一本通 1632 [NOIP2012] 同余方程：求 ax  1 (mod b) 的最小正整数解
long long exgcd(long long a, long long b, long long &x, long long &y) {
    if (b == 0) { x = 1; y = 0; return a; }
    long long x1, y1;
    long long g = exgcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return g;
}
```

本页要点

标准 exgcd 函数一字不改即可直接拿来使用。

例 3.2 | 核心代码（2/2）：主函数部分（code/congruence_eq.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    long long a, b;
    if (!(cin >> a >> b)) return 0;
    long long x, y;
    exgcd(a, b, x, y);
    long long ans = (x % b + b) % b;
    if (ans == 0) ans = b;                // 题目保证 b >= 2 且要求正整数解
    cout << ans << '\n';
    return 0;
}
```

本页要点

算完 `(x % b + b) % b` 后加上 `if (ans == 0) ans = b;` 是求“最小正整数解”的好习惯。

例 3.2 | 易错点与 3 组自测边界数据

自测输入	预期输出	测试意图与容易踩中的坑
3 10	7	原题样例，检验合数模数 10 不能用费马小定理。
1 2	1	最小边界数据：$a=1, b=2$，检验输出最小正整数 1。
1999999999 2000000000	1999999999	极大互质数据（$a=b-1$），检验 $a\cdot x\equiv 1 \pmod b$ 时特解为 -1 的负余数修正。

本页要点

用 1999999999 2000000000 这种 $a=b-1$ 的数据自测，答案一定也是 $b-1$。

例 3.3（[提高] 计算器前两问 | 一本通 1636）：先读懂要求

原题题意（ ，聚焦 $K = 1, 2$）

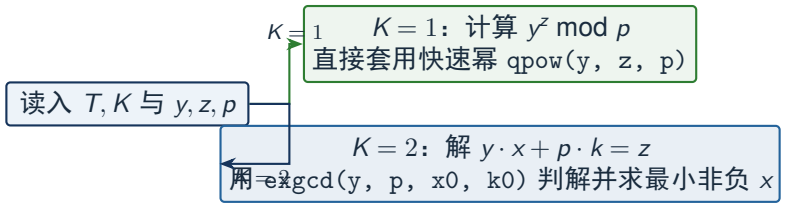
给定 T 组询问（同组测试点询问类型 K 相同），每组给出正整数 y, z, p（$1 \le y, z, p \le 10^9$，p 为质数）：
$K = 1$：计算 $y^z \bmod p$；　　$K = 2$：求满足 $x \cdot y \equiv z \pmod p$ 的最小非负整数 x（无解输出 `Orz,I cannot find x!`）。

- 原题样例（$K = 2$）：输入 3 2 后接 2 1 3、2 2 3、2 3 3 $\Longrightarrow$ 输出 2、1、0。
- 注意 $K = 2$ 的第三组 2 3 3：因为 $z = 3$ 是 $p = 3$ 的倍数，方程变成 $2x \equiv 0 \pmod 3$，最小非负解是 $x = 0$！

本页要点

这题把第 1 章快速幂与本章 exgcd 线性同余拼在一起，注意无解提示语的标点空格不能错。

例 3.3 | 怎么想到的：快速幂与 exgcd 的分工



- 为什么 $K=2$ 推荐 exgcd 而不是直接算 $z \cdot y^{p-2} \bmod p$? 因为当 y 是 p 的倍数而 z 也是 p 的倍数时（如 $y=3, z=3, p=3$），y 没有逆元，但方程 $3x \equiv 3 \pmod 3$ 依然有解 $x=0$！用 exgcd 能统一正确处理！

本页要点

不要在 $K=2$ 里无脑套逆元公式 $z \cdot y^{-1}$！当 $y \equiv 0 \pmod p$ 且 $z \equiv 0 \pmod p$ 时逆元不存在但方程有解 0。

例 3.3 | 样例手推跟踪表（K=2 三组询问）

逐组代入 $y \cdot x + p \cdot k = z$，用 `exgcd(y, p, x0, k0)` 求解：

(y, z, p)	$g = \gcd(y, p)$	$z \bmod g$	exgcd 得 x_0	周期 $\bmod = p/g$	最小非负解
$(2, 1, 3)$	1	0（有解）	-1	3	$(-1 \times 1 \bmod 3 + 3) \bmod 3 = \mathbf{2}$
$(2, 2, 3)$	1	0（有解）	-1	3	$(-1 \times 2 \bmod 3 + 3) \bmod 3 = \mathbf{1}$
$(2, 3, 3)$	1	0（有解）	-1	3	$(-1 \times 3 \bmod 3 + 3) \bmod 3 = \mathbf{0}$

- 假如再加一组 $(3, 2, 3)$：$g = \gcd(3, 3) = 3$，$z \bmod g = 2 \neq 0$，立刻判无解输出 `Orz,I cannot find x!`。

本页要点

三组输出 2，1，0 与样例分毫不差；exgcd 天然覆盖了 z 为 p 倍数和 y 为 p 倍数的边界。

例 3.3 | 核心步骤与相对模板改动

按顺序计算	纸上得到什么	核对什么问题
1. 若 $K = 1$，跑 `qpow(y, z, p)`	$y^z \bmod p$	每次乘法用 (i128) 防 $10^9 \times 10^9$ 溢出
2. 若 $K = 2$，跑 `exgcd(y, p, x0, k0)`	g 与特解 x_0	若 `z % g != 0` 输出无解字符串
3. 算 `mod = p / g` 与最小非负解	$x \in [0, \mathrm{mod} - 1]$	乘法 `x0 * (z / g)` 同样用 i128 保护

- 相对标准模板的改动：把第 1 章快速幂与本章 exgcd 整合在同一个分支结构里，并在 qpow 内也用 i128 防乘法溢出。

本页要点

题面未要求 $K = 3$（BSGS 离散对数）在本章展开，聚焦 $K = 1, 2$ 把快速幂与 exgcd 写稳。

例 3.3 | 核心代码（1/3）：快速幂部分（code/calculator_p12.cpp）

```
using i128 = __int128_t;

long long qpow(long long a, long long b, long long p) {
    long long res = 1 % p;
    for (a %= p; b > 0; b >>= 1) {
        if (b & 1) res = (i128)res * a % p;
        a = (i128)a * a % p;
    }
    return res;
}
```

本页要点

`res = 1 % p` 能顺便处理 $p = 1$ 时答案应为 0 的极端边界。

例 3.3 | 核心代码（2/3）：exgcd 部分（code/calculator_p12.cpp）

```
long long exgcd(long long a, long long b, i128 &x, i128 &y) {
    if (b == 0) { x = 1; y = 0; return a; }
    i128 x1, y1;
    long long g = exgcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return g;
}
```

本页要点

与前两题完全相同的 exgcd 骨架，使用 i128 防止系数放大溢出。

例 3.3 | 核心代码（3/3）：主函数分支（code/calculator_p12.cpp）

```
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    int T, K; if (!(cin >> T >> K)) return 0;
    while (T--) {
        long long y, z, p; cin >> y >> z >> p;
        if (K == 1) cout << qpow(y, z, p) << '\n';
        else if (K == 2) {
            i128 x0, k0; long long g = exgcd(y, p, x0, k0);
            if (z % g != 0) cout << "Orz,I cannot find x!\n";
            else { long long mod = p / g;
                cout << (long long)((x0 * (z / g) % mod + mod) % mod) << '\n'; }
        }
    } return 0;
}
```

本页要点

无解提示语 `Orz,I cannot find x!` 注意逗号后面没有空格，必须逐字核对题面！

例 3.3 | 易错点与 3 组自测边界数据

自测输入	预期输出	测试意图与容易踩中的坑
1 2 后接 3 2 3	`Orz,I cannot find x!`	y 是 p 倍数但 z 不是，检验无解分支与字符串拼写。
1 2 后接 3 6 3	`0`	y 与 z 同为 p 倍数，若误用费马求逆元会错，exgcd 正确输出 0。
1 1 后接 2 0 1	`0`	$p = 1$ 的快速幂最小边界，检验 `res = 1 % p`。

本页要点

用 3 2 3 和 3 6 3 两组数据，能完美检验你的 $K = 2$ 分支是否真正吃透了同余方程。

高频错误对照表（同余、exgcd 与逆元）

题目 / 场景	常见错误写法	正确做法与原因
最小非负解周期错	对 `L` 或 `b` 取模	周期是 `|b| / g` 而不是 `|b|`！漏除 g 会使答案偏大。
方程左端出现负数	直接把 $a = m - n < 0$ 传入 exgcd	先 `if(a<0){a=-a; b=-b;}`，否则 g 可能变负。
特解放大乘积溢出	`long long t0 * (b / g)`	两项各达 2×10^9，乘积接近或超过上限，用 `__int128_t`。
合数模数乱用费马	模数非质数还算 `qpow(a, m-2, m)`	只有 m 是质数才成立！合数模数必须改用 exgcd。
负余数直接输出	`(x0 * (c / g)) % mod`	C++ 负数取模仍为负，必须写成 `(x % mod + mod) % mod`。

本页要点

“系数先翻正，特解乘 c/g 莫忘记，最小非负模 |b|/g，大数乘法用 __int128_t”！

按评测结果快速定位 Bug（CE / WA / RE / TLE）

评测结果	本章最常见元凶	排查第一动作
WA（部分错）	1) 周期写成 b 而非 b/g；2) 乘法溢出变负；3) 负余数未加 mod。	检查 `mod = abs(b)/g`、`__int128_t` 和 `(x%m+m)%m`。
WA（全错）	1) 合数模数套用费马小定理；2) 无解字符串少空格或大小写错。	核对模数是否质数，逐字比对无解提示语。
RE（运行时错）	$a = b = 0$ 时执行了 `c % g`（除以 0）。	确认 exgcd 输入不同时为 0，或提前特判 $a = b = 0$。
TLE（超时）	用 `for` 循环暴力枚举 x 或枚举找逆元。	换成 $O(\log n)$ 的 exgcd 或 $O(n)$ 线性递推。

本页要点

答案偏大一倍查 `L/g`；输出负数查 `+mod`；大样例挂掉查 `__int128_t`！

提交前检查清单（六必查）

1. 调用 `exgcd(a, b, x, y)` 前，传入的系数 `a`，`b` 是否已保证非负（若 $a < 0$ 是否方程两边同时取反）？
2. 是否在判无解条件 `c % g != 0` 成立时立刻输出无解并退出（不进入后续乘法）？
3. 从 $ax_0 + by_0 = g$ 扩大到 $ax + by = c$ 时，特解有没有先乘以 `c / g`，且乘积使用了 `__int128_t`？
4. 求 x 的最小非负整数解时，取模周期写的是 `abs(b) / g` 还是错写成了 `b`？
5. 若代码出现 C++ 取余运算，负数结果有没有用 `(x % m + m) % m` 修正到 $[0, m)$？
6. 使用费马小定理求逆元前，是否确认过模数一定是质数且与底数互质？

最后一步

用 `1 2 3 4 5`（输出 4）和 `0 2 2 2 6`（输出 `Impossible`）验证后再提交！

本章全部练习与完成标准

1. 学校 **OJ：1631** ｜青蛙的约会（必做，`code/frog_exgcd.cpp`）：熟练处理负系数与 `__int128_t` 防溢出。
2. 学校 **OJ：1632** ｜ **[NOIP2012]** 同余方程（必做，`code/congruence_eq.cpp`）：掌握合数模数下的最小正整数解。
3. 学校 **OJ：1636** ｜计算器（提高，`code/calculator_p12.cpp`）：先独立 AC 前两问，后续可挑战第三问 BSGS 算法。

完成标准

能手工推导 `x = y1, y = x1 - (a/b)*y1`；能 1 分钟写出带 `|b|/g` 最小非负解求解与线性逆元递推，并独立 AC《青蛙的约会》和《同余方程》。

课后安排与下一章预告

- 复习：用 exgcd 手推一次 $14x + 6y = 2$ 的递归与回溯赋值过程，验证得到的 (x_0, y_0)。
- 练习：把线性递推求逆元代码敲一遍，输出 $p = 11$ 时 $1 \sim 10$ 的全部逆元并手算验算。
- 下一章预告：第 4 章《组合计数、容斥与期望》——用本章学的乘法逆元 $O(p)$ 预处理阶乘逆元，再用 Lucas 定理拆位算超大组合数，最后用期望线性性一秒算穿概率题！

本章一句话

辗转相除只管算最大公约数，扩展欧几里得顺便把回溯路线上的整数系数一并讨回来。

谢谢！