组合计数、容斥与期望

隔板法与阶乘逆元 + Lucas 定理 + 容斥原理 + 期望线性性

HGNU-ACM 实验室算法组

零基础精讲：从第 3 章逆元出发，打通竞赛计数、奇加偶减与期望拆解





本章目录

排列组合、隔板法与阶乘逆元

Lucas 定理：按 p 进制拆位算超大组合数

容斥原理：奇加偶减消去重叠

数学期望与期望线性性

前四节搭建竞赛计数的四大核心工具：隔板法与阶乘逆元、Lucas 定理拆位、容斥原理“奇加偶减”、期望线性性。

本章目录（续）

例题精讲：从模板到真题落地

常见错误、检查清单与练习安排

后两节精讲《超能粒子炮・改》《越狱》《集卡问题》三道真题与经典模型，并总结高频错因与第 5 章预告。

生活引入：选班委与分糖果

先看两个手算小例子

1) 从 5 名同学中选 2 人分别当正班长和副班长，有多少种选法？
2) 从 5 名同学中选 2 人参加打扫卫生（两人角色相同），有多少种选法？

- 排列 A_n^m（顺序有关）：正班长有 5 种，副班长剩 4 种，共 $5 \times 4 = 20$ 种。
- 组合 C_n^m 或 $\binom{n}{m}$（顺序无关）：上面每种两人组合被正副顺序算了 $2! = 2$ 次，故除掉重复，共 $\frac{5\times4}{2!} = 10$ 种。
- 公式总结：$A_n^m = \frac{n!}{(n-m)!}$，$\binom{n}{m} = \frac{n!}{m!(n-m)!}$（规定 $0! = 1$）。

本页要点

看清题目“选出的人/物是否区分顺序”：区分顺序用排列 A_n^m，不区分顺序用组合 $\binom{n}{m}$。

暴力算组合数为什么会溢出或超时？

- 做法一（直接套公式算阶乘）：先算 $n!$ 再除以 $m!(n-m)!$。可 $21!$ 就已经超过 `long long` 上限（约 9.22×10^{18}）直接溢出！
- 做法二（杨辉三角递推）：用 $\binom{n}{m} = \binom{n-1}{m-1} + \binom{n-1}{m}$ 边加边取模。可如果 $n = 10^5$，二维数组开到 $10^5 \times 10^5$ 需要 40 GB 内存，直接 **MLE**，时间 $O(n^2) = 10^{10}$ 必 **TLE**！
- 承接第 **3** 章：利用第 3 章学的乘法逆元，把组合数公式里的“除以 $k!(n-k)!$”变成“乘以阶乘的逆元”，就能在 $O(n)$ 预处理后 $O(1)$ 回答任意查询！

本页要点

杨辉三角适合 $n \leq 2000$；当 n 达到 $10^5 \sim 10^6$ 时，必须用“阶乘 + 阶乘逆元”边算边取模！

隔板法模型一：每盒至少 1 个（正整数解）

生活场景：把 5 颗相同的糖果分给 3 个不同的小朋友，每人至少分 1 颗

等价于求方程 $x_1 + x_2 + x_3 = 5$ 的正整数解个数（$x_i \geq 1$）。

- 把 5 颗相同的糖果排成一排，它们中间有 $5 - 1 = 4$ 个空隙。
- 只要往这 4 个空隙里插进 $3 - 1 = 2$ 块木板，就把糖果切成了 3 份，每份至少有 1 颗！
- 因此方案数就是从 $n - 1$ 个空隙选 $m - 1$ 个插板：$\binom{n-1}{m-1} = \binom{4}{2} = 6$。

本页要点

“n 个相同球放进 m 个不同盒，每盒至少 1 个” $\Longleftrightarrow$ $n - 1$ 个空隙插 $m - 1$ 块板，公式 $\binom{n-1}{m-1}$。

隔板法模型二：允许盒子为空（预借 m 个球转化）

变式场景：还是 5 颗相同糖果分给 3 个小朋友，但允许有人分到 0 颗！

等价于求方程 $x_1 + x_2 + x_3 = 5$ 的非负整数解个数（$x_i \geq 0$）。

- 直接插板不行，因为两块板插在同一个空隙里不好数。
- 绝妙转化（预借法）：先向每个小朋友各自“借”1 颗糖（一共借来 $m = 3$ 颗），变成一共 $n + m = 8$ 颗糖。
- 现在要求每人至少拿 1 颗（拿到后最后再还回刚才借的那 1 颗，允许最终剩 0 颗！）。
- 于是完美退化回模型一：总球数 $n + m$，插 $m - 1$ 块板 $\Longrightarrow \binom{n+m-1}{m-1}$！

本页要点

允许分到 0 个时，先给每个盒子预借 1 个球变成正整数问题，公式立刻变成 $\binom{n+m-1}{m-1}$。

图解隔板法：空隙插板与预借转化

情况一（至少 **1** 个）：$n = 5$ 颗糖，4 个空隙选 $m - 1 = 2$ 处插板



$\Longrightarrow$ $(2,\ 2,\ 1)$，共 $\binom{5-1}{3-1} = 6$ 种

情况二（允许为空）：先每盒借 1 颗（共借 3 颗），再按至少 1 个插板



$\Longrightarrow$ 归还后 $(3,\ \mathbf{0},\ 2)$

本页要点

两块板相邻意味着中间那个盒子只分到 1 个借来的球，归还后恰好剩下 0 个！

阶乘逆元：只需 1 次快速幂的 $O(N)$ 倒推神技

- 要算 $\binom{n}{k} \bmod p = \text{fac}[n] \cdot \text{inv}[k] \cdot \text{inv}[n-k] \bmod p$，其中 $\text{fac}[i] = i! \bmod p$，$\text{inv}[i] = (i!)^{-1} \bmod p$（$p$ 为质数且 $N < p$）。
- 如果对 $1! \sim N!$ 每个都调一次第 3 章学的费马快速幂 `qpow(fac[i], p-2)`，耗时 $O(N \log p)$。
- 倒推技巧（只需 1 次快速幂）：
  1. 正序算所有阶乘：$\text{fac}[0] = 1,\ \text{fac}[i] = \text{fac}[i-1] \cdot i \bmod p$；
  2. 只算最大那个阶乘的逆元：$\text{inv}[N] = \text{qpow}(\text{fac}[N], p-2, p)$；
  3. 从 N 往回倒推：因为 $\frac{1}{(i-1)!} = \frac{1}{i!} \cdot i$，所以 $\text{inv}[i-1] = \text{inv}[i] \cdot i \bmod p$！

本页要点

先用 1 次快速幂算出 $\text{inv}[N]$，再乘以 i 往回倒推到 $\text{inv}[0] = 1$，总复杂度仅 $O(N + \log p)$！

手算跟踪表：取 $p = 7,\ N = 5$ 预处理阶乘与逆元

先正序算出 fac$[0..5]$，用费马算 inv$[5] = 1^{-1} \bmod 7 = 1$，再倒推 inv$[4..0]$：

i	fac$[i] = i! \bmod 7$	inv$[i]$ 的计算过程	验算 fac$[i] \cdot$ inv$[i] \bmod 7$
5	$120 \bmod 7 = 1$	qpow$(1, 5, 7) = 1$（终点起点）	$1 \times 1 \equiv 1$
4	$24 \bmod 7 = 3$	inv$[5] \times 5 \bmod 7 = 1 \times 5 = 5$	$3 \times 5 = 15 \equiv 1$
3	$6 \bmod 7 = 6$	inv$[4] \times 4 \bmod 7 = 5 \times 4 \bmod 7 = 6$	$6 \times 6 = 36 \equiv 1$
2	$2 \bmod 7 = 2$	inv$[3] \times 3 \bmod 7 = 6 \times 3 \bmod 7 = 4$	$2 \times 4 = 8 \equiv 1$
1	1	inv$[2] \times 2 \bmod 7 = 4 \times 2 \bmod 7 = 1$	$1 \times 1 \equiv 1$
0	1	inv$[1] \times 1 \bmod 7 = 1 \times 1 \bmod 7 = 1$	$1 \times 1 \equiv 1$

本页要点

倒推到 $i = 1$ 时会算出 inv$[0] =$ inv$[1] \times 1 = 1$，正好连 $0!$ 的逆元一起算对！

为什么阶乘逆元倒推一定对？（不变量论证）

论证环节	不变量：$\mathrm{inv}[i] \cdot \mathrm{fac}[i] \equiv 1 \pmod p$（前提：$N < p$ 且 p 为质数）
初始化（$i = N$）	由于 $N < p$ 且 p 是质数，$1 \sim N$ 都不是 p 的倍数，故 $N! \not\equiv 0 \pmod p$。由费马小定理，$\mathrm{inv}[N] = (N!)^{p-2} \bmod p$ 满足 $\mathrm{inv}[N] \cdot \mathrm{fac}[N] \equiv 1 \pmod p$。
保持（从 i 到 $i-1$）	假设 $\mathrm{inv}[i] \cdot i! \equiv 1 \pmod p$ 成立。因为 $i! = (i-1)! \cdot i$，代入得 $(\mathrm{inv}[i] \cdot i) \cdot (i-1)! \equiv 1 \pmod p$。因此 $\mathrm{inv}[i-1] = \mathrm{inv}[i] \cdot i \bmod p$ 依然满足不变量！
终止（$i = 0$）	倒推至 $i = 0$ 时得到 $\mathrm{inv}[0] = 1$，刚好与 $0! = 1$ 匹配，可用于算 $\binom{n}{0}$ 和 $\binom{n}{n}$。

本页要点

注意前提条件：必须保证 $N < p$！一旦 $N \geq p$，就会导致 $\mathrm{fac}[N] \equiv 0 \pmod p$，整个倒推失效。

模板代码：预处理阶乘逆元与 $O(1)$ 求组合数

```
void init_comb(int N, long long p) {
    fac[0] = 1;
    for (int i = 1; i <= N; ++i) fac[i] = fac[i - 1] * i % p;
    inv[N] = qpow(fac[N], p - 2, p);        // 仅需 1 次快速幂
    for (int i = N; i >= 1; --i)
        inv[i - 1] = inv[i] * i % p;        // O(N) 倒推全部阶乘逆元
}
long long C(int n, int k, long long p) {
    if (k < 0 || k > n) return 0;           // 越界立即返回 0
    return fac[n] * inv[k] % p * inv[n - k] % p;
}
```

本页要点

在 C(n, k) 中连续相乘两项时，中间一定要先 % p 再乘第三项，否则会爆 long long！

上机验证与反事实追问

```
// 假设已调用 init_comb(10, 1000000007)
cout << C(5, 2, 1000000007) << '\n'; // 普通：输出 10
cout << C(5, 0, 1000000007) << '\n'; // 最小边界：输出 1
cout << C(5, 6, 1000000007) << '\n'; // 特殊边界 k>n：输出 0
```

- 三组验证：C(5,2) $= 10$；C(5,0) $= 1$；C(5,6) $= 0$。
- 反事实追问：如果不小心把倒推循环 `for (int i = N; i >= 1; --i)` 写成了正序从 `inv[0]=1` 往后乘 `inv[i] = inv[i-1] * i % p`，会发生什么？（算出来的根本不是逆元而是普通阶乘 `fac[i]`，所有组合数全错！）。

本页要点

记清方向：阶乘是正序乘 i，阶乘逆元是从 N 倒推乘 $i!$

本节易错对照：隔板法与阶乘逆元

错误写法	正确写法	导致的评测结果与原因
`C(n, k)` 漏判 `k > n`	开头加 `if(k<0||k>n) return 0`	**RE / WA**：访问 `inv[n-k]` 时出现负下标越界。
连续乘两项不取模	`fac[n] * inv[k] % p * inv[n-k] % p`	**WA**：三项连乘最大可达 $(10^9)^3 = 10^{27}$，必溢出。
允许为空错用 $\binom{n-1}{m-1}$	允许为空用 $\binom{n+m-1}{m-1}$	**WA**：漏掉了先预借 m 个球的转化。
`N >= p` 还用阶乘逆元	当 $n \geq p$ 时改用 Lucas 定理	**WA**：$\mathrm{fac}[p] \equiv 0 \pmod p$，逆元全变成 0。

本页要点

“`k>n` 立刻归零，中间连乘先取模，隔板可空先加 m，$n \geq p$ 找 Lucas”。

痛点引入：当 $n = 10^{18}$、模数 $p = 2333$ 时怎么办？

- 前一节我们学会了用 `fac[]` 和 `inv[]` 在 $O(1)$ 内算 $\binom{n}{k} \bmod p$。
- 可如果遇到题目给出 $n, k \le 10^{18}$，而模数是一个很小的质数 $p = 2333$：
  1. 开不下数组：`fac[10^18]` 需要 8×10^{18} 字节，直接 **MLE**；循环 10^{18} 次直接 **TLE**！
  2. 数学失效（零的逆元）：因为 $n \ge p$，$n!$ 里一定包含因子 p，所以 $n! \equiv 0 \pmod p$。0 没有逆元，直接算 $\frac{n!}{k!(n-k)!}$ 会得到 $0/0$ 的假结果！
- 我们需要一种方法：只预处理 $0 \sim p-1$（不到 2333 个数），就能算出 $\binom{10^{18}}{k} \bmod p$！

本页要点

只要 $n \ge p$，普通阶乘逆元就会因为“数组开不下 $+$ $p! \equiv 0$ 没有逆元”而彻底失效。

Lucas 定理：把大组合数拆成 p 进制一位位算

Lucas 定理（p 必须为质数）

对于非负整数 n, k 和质数 p，有：

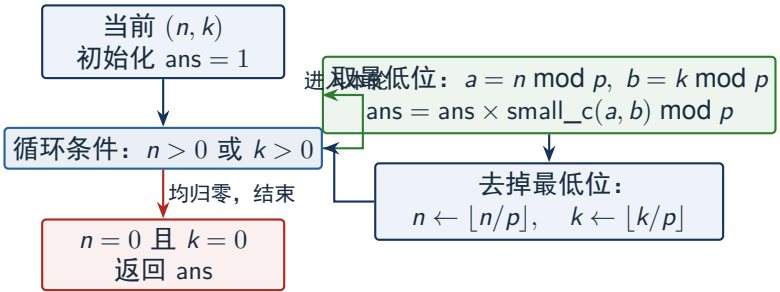
$$\binom{n}{k} \equiv \binom{n \bmod p}{k \bmod p} \cdot \binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor} \pmod{p}$$

- 大白话理解：把 n 和 k 写成 p 进制数（就像十进制拆个位、十位、百位一样）：
  - $n \bmod p$ 和 $k \bmod p$ 就是它们的“p 进制最低位”；
  - 算完最低位的小组合数 $\binom{n \bmod p}{k \bmod p}$ 后，把 n, k 同时除以 p（右移一位），继续算下一位，把所有位的结果乘起来！
- 每次都只算 $< p$ 的小组合数，规模从 10^{18} 瞬间降到 $O(\log_p n)$ 步！

本页要点

Lucas 定理就是把 n 和 k 写成 p 进制，各位分别算小组合数再连乘。

图解 Lucas 定理：p 进制逐位拆解流程



本页要点

循环条件写成 `while (n || k)`，每轮取最低位算 `small_c` 并右移一位，直到两数全为 0。

手算跟踪表：用 Lucas 定理算 $\binom{13}{5} \bmod 3$

取 $n=13, k=5, p=3$（已知 $13=(111)_3,\ 5=(012)_3$，真值 $\binom{13}{5}=1287\equiv 0 \pmod 3$）：

轮次	n	k	$n \bmod 3$	$k \bmod 3$	small_c($n \bmod 3, k \bmod 3$)	累计 ans
第 1 轮	13	5	1	2	$\binom{1}{2}=\mathbf{0}$（因 $2>1$!）	$1\times 0 \bmod 3=0$
第 2 轮	4	1	1	1	$\binom{1}{1}=1$	$0\times 1 \bmod 3=0$
第 3 轮	1	0	1	0	$\binom{1}{0}=1$	$0\times 1 \bmod 3=0$
终止步	0	0	—	—	`n || k` 为假，退出循环	最终返回 **0**

- 再算一个非零例子：$\binom{10}{3} \bmod 7$。第 1 轮 $(10,3)\to\binom{3}{3}=1$；第 2 轮 $(1,0)\to\binom{1}{0}=1$；终止 $(0,0)$，返回 $1\times 1=\mathbf{1}$（验算 $\binom{10}{3}=120=17\times 7+1\equiv 1$）。

本页要点

只要在任意一位出现 $k \bmod p > n \bmod p$，该位组合数为 0，整个乘积立刻归 0!

为什么 Lucas 定理一定对？（二项式展开直观理解）

- 当 p 是质数且 $1 \le i \le p-1$ 时，$\binom{p}{i} = \frac{p!}{i!(p-i)!}$ 分子有因子 p 而分母没有，所以 $\binom{p}{i} \equiv 0 \pmod p$！
- 因此在模 p 意义下，$(1+x)^p \equiv 1 + x^p \pmod p$（中间交叉项全被 p 整除消掉了！）。
- 把 n 写成 $n = qp + r$（其中 $q = \lfloor n/p \rfloor$，$r = n \bmod p$），于是：

$$(1+x)^n = (1+x)^{qp} \cdot (1+x)^r \equiv (1+x^p)^q \cdot (1+x)^r \pmod p$$

- 左边 x^k 的系数是 $\binom{n}{k}$；右边要凑出 x^k（$k = q'p + r'$），只能从 $(1+x^p)^q$ 里取 $x^{q'p}$（系数 $\binom{q}{q'}$），从 $(1+x)^r$ 里取 $x^{r'}$（系数 $\binom{r}{r'}$），两者相乘即得 $\binom{q}{q'}\binom{r}{r'}$！

本页要点

核心桥梁是质数 p 满足 $(1+x)^p \equiv 1 + x^p \pmod p$，对比两边 x^k 系数即证 Lucas 定理。

Lucas 定理模板代码与 k > n 保护

```
long long small_c(long long n, long long k, long long p) {
    if (k > n) return 0; // 绝对不能删！否则访问 inv[n-k] 越界
    return fac[n] * inv[k] % p * inv[n - k] % p;
}
long long lucas(long long n, long long k, long long p) {
    long long ans = 1;
    while (n || k) {
        ans = ans * small_c(n % p, k % p, p) % p; // 当前 p 进制位
        n /= p; k /= p;                           // 右移一位
    }
    return ans;
}
```

本页要点

即使输入 $k > n$，在某一位也必会出现 $k \bmod p > n \bmod p$，由 small_c 自动归 0！

上机验证与反事实追问

```
// 假设 p = 7，已预处理 fac[0..6] 与 inv[0..6]
cout << lucas(10, 3, 7) << '\n'; // 普通：输出 1（120 % 7 = 1）
cout << lucas(5, 0, 7)  << '\n'; // 最小边界：输出 1
cout << lucas(13, 5, 3) << '\n'; // 某位 k%p > n%p：输出 0
```

- 三组验证：`lucas(10,3,7)` $= 1$；`lucas(5,0,7)` $= 1$；`lucas(13,5,3)` $= 0$。
- 反事实追问：如果把 `small_c` 里的 `if (k > n) return 0;` 删掉，跑第三组 `lucas(13, 5, 3)` 会发生什么？（第 1 轮传入 $n \bmod 3 = 1$，$k \bmod 3 = 2$，程序会去访问 `inv[1 - 2] = inv[-1]`，引发数组负下标越界导致 **RE** 或输出随机垃圾值！）。

本页要点

测试 `lucas(13, 5, 3)` 是检验 `small_c` 是否正确处理 $k \bmod p > n \bmod p$ 的最佳试金石。

本节易错对照：Lucas 定理

错误写法	正确写法	导致的评测结果与原因
`small_c` 漏判 `k > n`	开头写 `if (k > n) return 0;`	**RE / WA**：某一位 $k \bmod p > n \bmod p$ 时访问负下标。
合数模数套用 Lucas	合数模数需拆质因子（exLucas）	**WA**：$(1+x)^m \equiv 1+x^m$ 仅在 m 为质数时成立。
预处理数组只开到 $p-1$	数组大小至少开到 p（下标 $0 \ldots p-1$）	**RE**：若把 `vector` 开成 `p-1`，访问 `fac[p-1]` 越界。
p 太大还预处理 $O(p)$	当 $p \le 10^5$ 预处理；p 很大但 k 小直接算	**TLE / MLE**：$p = 10^9$ 时开不下 `fac[p]`。

本页要点

Lucas 定理适用画像：“n, k 极大（10^{18}）+ 模数 p 是小质数（$\le 10^5$）”。

生活引入：班里到底有多少人会编程？

先看一个手算小例子

一个班里会 C++ 的有 20 人，会 Python 的有 15 人，两种语言都会的有 6 人。问至少会一门语言的有多少人？

- 如果直接算 $20 + 15 = 35$ 人，那 6 个“两种都会”的同学被算了 ** 两次 **！
- 把多算的那一次减掉：$20 + 15 - 6 = 29$ 人。
- 如果再加一门 Java（3 个集合），两两减完之后，** 三种都会 ** 的人又被减多了，得再补加回来！
- 这套“先把单个集合全加上，再减掉两两重叠，再加回三三重叠……”的规则，就叫容斥原理！

本页要点

容斥原理的核心口诀只有四个字：“奇加偶减”——奇数个集合的交集加，偶数个集合的交集减！

暴力做法为何会 TLE？以“整除计数”为例

经典问题

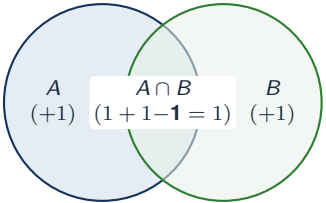
求 $[1, N]$ 中有多少个整数能被给定 m 个质数 $p_1, p_2, \ldots, p_m$ 中的至少一个整除？（$N \le 10^{18},\ m \le 15$）

- 暴力做法：用 `for (long long x = 1; x <= N; ++x)` 枚举每个数，检查它能不能被某个 p_i 整除。
- 操作次数估算：当 $N = 10^{18}$ 时，循环要跑 10^{18} 次，即使用超级计算机也要跑几百年，绝对 **TLE**！
- 容斥突破：$1 \sim N$ 中能被 d 整除的个数直接就是 $\lfloor N/d \rfloor$（$O(1)$ 算完！）。用容斥原理枚举这 m 个质数的非空子集，只需 $2^m - 1 \le 32767$ 次运算，瞬间出解！

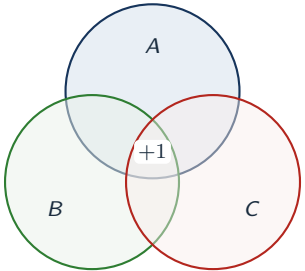
本页要点

当全集 N 极大（10^{18}）而限制条件数 m 很小（$m \le 20$）时，用容斥原理把 $O(N)$ 降成 $O(2^m)$！

图解容斥原理：两集合与三集合的重叠消重



$$|A \cup B| = |A| + |B| - |A \cap B|$$



$$|A \cup B \cup C| = \sum |A_i| - \sum |A_i \cap A_j| + |A \cap B \cap C|$$

本页要点

正中间三圆交汇的区域：被 A, B, C 加了 3 次，被两两交集减了 3 次，最后靠 $+|A \cap B \cap C|$ 补回成 1 次！

怎样写代码枚举所有子集？二进制状态压缩

- 容斥公式写成统一形式：

$$\left|\bigcup_{i=1}^{m} A_i\right| = \sum_{\emptyset \neq S \subseteq \{1,\ldots,m\}} (-1)^{|S|-1} \left|\bigcap_{i \in S} A_i\right|$$

- 新术语（二进制状态压缩）：用一个 m 位二进制数 mask（从 1 到 $2^m - 1$）表示选了哪些集合——第 i 位是 1 就表示选了第 i 个集合，是 0 表示没选。
- 统计 mask 中有多少个 1（即子集大小 $|S|$）：
  - 若 1 的个数是 ** 奇数 **，符号为 $+1$（加）；
  - 若 1 的个数是 ** 偶数 **，符号为 -1（减）。

本页要点

从 mask = 1 枚举到 (1 << m) - 1，自动跳过空集 0，轻松实现任意 m 的容斥！

手算跟踪表：$N = 20$ 内能被 $\{2, 3, 5\}$ 之一整除的个数

令 $p = [2, 3, 5]$，枚举 `mask = 1 .. 7`（即 $2^3 - 1$ 个非空子集）：

`mask`（二进制）	选中的质数	乘积 d	奇偶符号	贡献 $\pm\lfloor 20/d \rfloor$	累计 `ans`
001（1）	$\{2\}$	2	$+$（1 个）	$+10$	10
010（2）	$\{3\}$	3	$+$（1 个）	$+6$	16
011（3）	$\{2, 3\}$	6	$-$（2 个）	-3	13
100（4）	$\{5\}$	5	$+$（1 个）	$+4$	17
101（5）	$\{2, 5\}$	10	$-$（2 个）	-2	15
110（6）	$\{3, 5\}$	15	$-$（2 个）	-1	14
111（7）	$\{2, 3, 5\}$	30	$+$（3 个）	$+\lfloor 20/30 \rfloor = 0$	**14**

本页要点

数一数 $1 \sim 20$ 中能被 $2, 3, 5$ 之一整除的数：只剩下 $1, 7, 11, 13, 17, 19$ 这 6 个互质的，其余 $20 - 6 = \mathbf{14}$ 个全中！

为什么“奇加偶减”一定不多不少恰好算 1 次？

- 假设某个元素 x 同时属于这 m 个集合中的 t 个集合（$t \geq 1$）。
- 在我们的枚举中，只有当选出的子集 S 完全落在那 t 个集合里时，x 才会被计入。
- 于是 x 被统计的总次数等于：

$$\binom{t}{1} - \binom{t}{2} + \binom{t}{3} - \cdots + (-1)^{t-1}\binom{t}{t}$$

- 回想二项式定理：$(1-1)^t = \binom{t}{0} - \binom{t}{1} + \binom{t}{2} - \cdots + (-1)^t\binom{t}{t} = 0$。
- 移项即得：$\binom{t}{1} - \binom{t}{2} + \cdots = \binom{t}{0} = \mathbf{1}$！无论 t 是几，每个符合条件的元素最终净贡献永远是 1！

本页要点

二项式定理 $(1-1)^t = 0$ 保证了：只要元素至少属于 1 个集合，它在容斥中不多不少恰好被算 1 次！

模板代码：二进制状态压缩容斥（整除计数）

```
long long count_divisible(long long N, const vector<long long> &p) {
    int m = p.size(); long long ans = 0;
    for (int mask = 1; mask < (1 << m); ++mask) {
        long long prod = 1; int cnt = 0;
        for (int i = 0; i < m; ++i) if (mask >> i & 1) {
            ++cnt;
            if (prod > N / p[i]) { prod = N + 1; break; } // 防溢出
            prod *= p[i];
        }
        ans += (cnt & 1 ? 1 : -1) * (N / prod); // 奇加偶减
    }
    return ans;
}
```

本页要点

用 `prod > N / p[i]` 提前截断，既防 `prod` 乘爆 `long long`，又让 `N / prod == 0`！

上机验证与反事实追问

```
cout << count_divisible(20, {2, 3, 5}) << '\n'; // 普通：输出 14
cout << count_divisible(1, {2, 3}) << '\n';     // 最小边界：输出 0
cout << count_divisible(10, {2}) << '\n';       // 单集合：输出 5
```

- 三组验证：(20, {2,3,5}) $\to 14$；(1, {2,3}) $\to 0$；(10, {2}) $\to 5$。
- 反事实追问：如果把外层循环写成 `for (int mask = 0; mask < (1 << m); ++mask)`（从 0 开始包含空集），输出会变成什么？（当 `mask=0` 时 `cnt=0` 为偶数，`prod=1`，程序会多执行一次 `ans -= N / 1`，结果全部错减了 $N!$）。

本页要点

二进制容斥务必从 `mask = 1` 开始；若各数不互质，把 `prod *= p[i]` 换成求最小公倍数 `lcm(prod, p[i])`！

本节易错对照：容斥原理

错误写法	正确写法	导致的评测结果与原因
`mask` 从 0 开始	`for (int mask = 1; ...)`	**WA**：空集对应偶数个集合且乘积为 1，会多减去 N。
奇偶符号写反	奇数个加，偶数个减	**WA**：算出的答案变成负数或严重偏小。
直接连乘 `prod *= p[i]`	乘前判断 `prod > N / p[i]`	**WA / RE**：15 个质数连乘轻松超过 10^{18} 溢出变负。
给非质数直接相乘	若集合不互质，用 `lcm` 计算交集	**WA**：例如能被 4 和 6 同时整除的是 12 的倍数，不是 24。

本页要点

“从 1 枚举非空集，奇数加来偶数减，乘前判溢不爆仓，非质数交集用 LCM”。

生活引入：掷骰子与“集卡”平均要抽几次？

先看两个生活问题

1) 掷一枚均匀六面骰子，每次可能掷出 $1 \sim 6$，平均每次能掷出多少点？
2) 方便面袋里有 n 种不同的水浒卡，每袋等概率随机送 1 张，集齐全部 n 种平均需要买几袋？

- 第 1 题很简单：每种点数概率都是 $\frac{1}{6}$，平均点数 $E(X) = 1 \cdot \frac{1}{6} + \cdots + 6 \cdot \frac{1}{6} = 3.5$。
- 新术语（数学期望 $E(X)$）：随机结果“按概率加权的平均值”，公式为 $E(X) = \sum x_i \cdot P(X = x_i)$。
- 可第 2 题“集卡”可能买 n 袋就齐，也可能买 1000 袋还没齐（无穷多种可能！），硬套定义算无穷级数非常痛苦！

本页要点

期望就是“带概率的平均值”；遇到无穷多步的随机过程，硬套定义算总分布会非常困难。

核心武器：期望线性性 $E(X+Y)=E(X)+E(Y)$

竞赛概率题第一定理：期望的线性性

对于任意两个随机变量 X 和 Y（完全不需要它们互相独立！），永远成立：

$$E(X+Y)=E(X)+E(Y), \qquad E(aX+b)=aE(X)+b$$

- 为什么这东西这么神？很多同学会和乘法公式 $E(XY)=E(X)E(Y)$（需要独立）搞混。
- 求和的期望 ** 永远等于 ** 期望的和！就算后一步受前一步影响（不独立），依然可以直接把各步期望加起来！
- 因此解题套路是：** 把总步数/总得分 S 拆成小阶段 $X_1+\cdots+X_n$，分别算 $E(X_i)$ 再相加！**

本页要点

期望线性性 $E(X+Y)=E(X)+E(Y)$ 不依赖独立性！把大过程拆成小阶段增量往往一秒出解。

图解集卡问题：把总步数拆成 n 个小阶段



- 设 X_i 为“从已有 $i-1$ 种卡片，到首次抽到第 i 种新卡片”还需要买的袋数。
- 此时袋子外还有 $n-(i-1) = n-i+1$ 种没集齐的卡，所以每次买中 ** 新卡 ** 的概率固定为 $p_i = \frac{n-i+1}{n}$。
- 几何分布结论：每次成功概率为 p，直到第一次成功的期望尝试次数就是 $\frac{1}{p}$！所以 $E(X_i) = \frac{n}{n-i+1}$。

本页要点

成功概率为 p 时，平均需要 $\frac{1}{p}$ 次才能成功一次（例如 $p = \frac{1}{6}$ 的骰子平均 6 次掷出 6）。

手算跟踪表：$n = 3$ 种卡片的期望步数拆解

当 $n = 3$ 时，总期望步数 $E(S) = E(X_1) + E(X_2) + E(X_3)$：

阶段 i	手头已有种类	抽中新卡概率 $p_i = \frac{3-i+1}{3}$	本段期望 $E(X_i) = \frac{1}{p_i}$	累计期望步数
$i = 1$	0 种	$3/3 = 1$	1（第 1 袋必是新卡）	1.00
$i = 2$	1 种	$2/3$	$3/2 = 1.50$	2.50
$i = 3$	2 种	$1/3$	$3/1 = 3.00$	**5.50**

- 推广到任意 n：总期望步数为 $E(S) = \sum_{i=1}^{n} \frac{n}{n-i+1} = n\left(1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right)$。

本页要点

集齐 3 种卡片平均只要 5.50 袋；集齐 100 种卡片则约需 $100 \times \ln 100 \approx 519$ 袋。

为什么不独立也能直接加？（小例子验证线性性）

反直觉验证：两个明显不独立的事件

袋里有 1 个红球、1 个黑球，不放回地依次摸出第 1 个球和第 2 个球。设 $X = 1$ 表示第 1 个是红球（否则 0），$Y = 1$ 表示第 2 个是红球（否则 0）。

- 显然 X 和 Y ** 极度不独立 **（只要 $X = 1$，就注定 $Y = 0$!）。
- 可只看单独的期望：$E(X) = \frac{1}{2}$，$E(Y) = \frac{1}{2}$。
- 它们的和 $X + Y$ 永远等于 1（总共就 1 个红球），所以 $E(X + Y) = 1$。
- 验算：$E(X) + E(Y) = \frac{1}{2} + \frac{1}{2} = 1 = E(X + Y)$，分毫不差！

本页要点

期望线性性对任意相关甚至互斥的随机变量都成立，这是它比概率乘法强大得多的原因。

期望递推入门：为什么总是从终点倒推？

- 另一类常见期望题无法直接拆成独立小段，需要设状态 $f[i]$ 表示“从状态 i 走到终点的期望步数”，且规定终点 $f[\text{end}] = 0$。
- 拿集卡问题举例：设 $f[i]$ 为“已集齐 i 种卡，还需买几袋才能集齐 n 种”，显然 $f[n] = 0$。
- 在状态 i 买一袋（花费 $+1$ 步）：
  - 有 $\frac{i}{n}$ 概率抽到旧卡，停在状态 i；有 $\frac{n-i}{n}$ 概率抽到新卡，走到状态 $i+1$。
- 列出方程：$f[i] = 1 + \frac{i}{n} f[i] + \frac{n-i}{n} f[i+1]$。
- 关键一步（移项消去自环）：把 $\frac{i}{n} f[i]$ 移到左边得 $\frac{n-i}{n} f[i] = 1 + \frac{n-i}{n} f[i+1]$，两边同除以 $\frac{n-i}{n}$：

$$f[i] = f[i+1] + \frac{n}{n-i}$$

本页要点

期望递推统一设“到终点的期望”，从 $f[n] = 0$ 倒推回 $f[0]$；遇到停留在原地（自环），先把 $f[i]$ 移项合并！

模板代码：集卡问题线性性计算与输出

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

double expected_draws(int n) {
    double ans = 0.0;
    for (int i = 1; i <= n; ++i) {
        ans += (double)n / (n - i + 1); // 线性叠加第 i 阶段期望步数
    }
    return ans;
}
```

- 只需 $O(n)$ 时间、$O(1)$ 空间；若要求对质数 p 取模（竞赛常见“分数取模”），则把除以 $n-i+1$ 换成乘以第 3 章学的乘法逆元 `inv[n - i + 1]`！

本页要点

注意 `(double)n / (n - i + 1)` 必须先把分子或分母强转为 `double`，否则整数除法会直接向下取整丢精度！

上机验证与反事实追问

```
cout << fixed << setprecision(2)
     << expected_draws(1) << ' '  // 最小边界：输出 1.00
     << expected_draws(2) << ' '  // 小数据：输出 3.00
     << expected_draws(3) << '\n';// 经典值：输出 5.50
```

- 三组验证：$n=1 \to$ **1.00**；$n=2 \to$ **3.00**；$n=3 \to$ **5.50**。
- 反事实追问：如果把 `(double)n / (n - i + 1)` 漏掉强转写成 `n / (n - i + 1)`，输入 3 会输出什么？（$\lfloor 3/3 \rfloor + \lfloor 3/2 \rfloor + \lfloor 3/1 \rfloor = 1+1+3 =$ **5.00**，白白丢掉 0.50!）。

本页要点

浮点期望题第一防线：整数相除前一定先转 `double`；模意义期望题第一防线：除法改乘逆元！

本节易错对照：数学期望与期望线性性

错误写法	正确写法	导致的评测结果与原因
`n / (n - i + 1)` 整数除	`(double)n / (n - i + 1)`	**WA**：C++ 两个整数相除自动截断小数部分。
递推遇到自环直接算	移项合并 `f[i]` 后再除系数	**WA**：等式右边用了还没算出的 `f[i]`。
误以为不独立不能相加	直接用 $E(\sum X_i) = \sum E(X_i)$	导致思路卡死或强行算复杂联合分布超时。
输出没控制小数位数	`cout << fixed << setprecision(2)`	**WA**：科学计数法或默认精度位数不足。

本页要点

“期望可加不靠独立，自环移项再倒推，浮点除法先强转，输出固定小数位”。

例 4.1（[提高] 超能粒子炮・改 | 一本通 1658）：先读懂要求

原题题意（ ）

给定 t 组询问（$t \le 10^6$），每组给出两个极大整数 n, k（$0 \le k, n \le 10^{18}$），求：

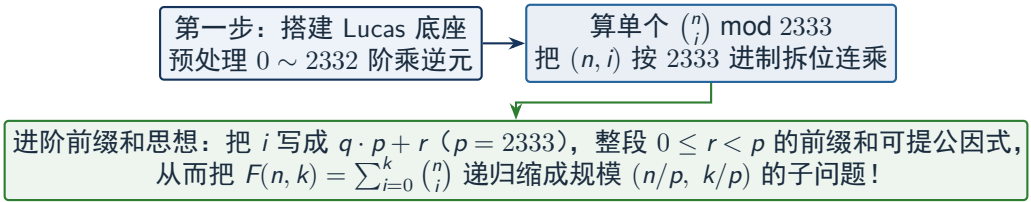
$$\sum_{i=0}^{k} \binom{n}{i} \bmod 2333$$

- 输入：第一行 t；接下来 t 行每行两个整数 n, k。
- 输出：t 行，每行一个整数表示威力之和模 2333 的余数。
- 原题样例：输入 3 组：`5 5` $\to$ `32`；`10 7` $\to$ `968`；`1145 14` $\to$ `763`。
- 本题先决核心底座：如何 $O(\log_{2333} n)$ 精准求出任意单个超大组合数 $\binom{n}{i} \bmod 2333$（即 `code/lucas_comb.cpp`）！

本页要点

模数 $p = 2333$ 是小质数而 $n, k \le 10^{18}$，第一反应就是 Lucas 定理底座！

例 4.1 | 怎么想到的：从 Lucas 底座到前缀和分块



- 无论算单点还是算分块前缀和，都离不开一个干净、零越界的 `Lucas` 结构体作为底层引擎。

本页要点

先把 `Lucas` 结构体写稳（$O(p)$ 预处理 + `small_c` + `comb`），整道题就攻克了一大半。

例 4.1 | 样例手推跟踪表（n=5，k=5）

第一组样例 `5 5`：因 $n = 5 < 2333$，此时 Lucas 只有 1 位，即 $\sum_{i=0}^{5} \binom{5}{i}$：

i	`small_c(5, i)`	累计前缀和	说明
0	$\binom{5}{0} = 1$	1	依赖 `fac[0]=1, inv[0]=1`
1	$\binom{5}{1} = 5$	6	$\text{fac}[5] \cdot \text{inv}[1] \cdot \text{inv}[4] \bmod 2333$
2	$\binom{5}{2} = 10$	16	组合对称性
3	$\binom{5}{3} = 10$	26	组合对称性
4	$\binom{5}{4} = 5$	31	—
5	$\binom{5}{5} = 1$	**32**	恰好等于 $2^5 = 32$，与样例一致！

本页要点

当 $n < p$ 时，Lucas 自动退化成单步 `small_c(n, k)`，无需任何额外特判！

例 4.1 | 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. 构造 `Lucas(p)`	`fac[0..p-1]` 与 `inv[0..p-1]`	`inv[p-1]` 用一次快速幂，再倒推回 `inv[0]=1`
2. 编写 `small_c(n, k)`	小组合数 $\binom{n}{k} \bmod p$	必须先判 `if (k > n) return 0;`
3. 编写 `comb(n, k)`	大组合数 $\binom{n}{k} \bmod p$	`while (n || k)` 逐位取模并整除 p

- 相对基础模板的改动：将快速幂、预处理数组和 `small_c/comb` 封装进 `struct Lucas`，支持任意小质数 p 复用。

本页要点

用 `struct Lucas` 封装后，换一道小质数模数题只需改一行构造参数即可复用！

例 4.1 | 核心代码（1/2）：预处理部分（code/lucas_comb.cpp）

```cpp
struct Lucas {
    long long p;
    vector<long long> fac, inv;
    long long qpow(long long a, long long b) const {
        long long r = 1 % p; a %= p;
        for (; b; b >>= 1, a = a * a % p) if (b & 1) r = r * a % p;
        return r;
    }
    explicit Lucas(long long p) : p(p), fac(p, 1), inv(p, 1) {
        for (int i = 1; i < p; ++i) fac[i] = fac[i - 1] * i % p;
        inv[p - 1] = qpow(fac[p - 1], p - 2);        // 费马小定理
        for (int i = p - 1; i > 0; --i) inv[i - 1] = inv[i] * i % p;
    }
```

本页要点

构造函数里用 fac(p, 1), inv(p, 1) 自动把第 0 项初始化为 1，省心又安全。

例 4.1 | 核心代码（2/2）：查询部分（code/lucas_comb.cpp）

```
    long long small_c(long long n, long long k) const {
        if (k > n) return 0;
        return fac[n] * inv[k] % p * inv[n - k] % p;
    }
    long long comb(long long n, long long k) const {
        long long ans = 1;
        while (n || k) {
            ans = ans * small_c(n % p, k % p) % p;  // p 进制拆位
            n /= p; k /= p;
        }
        return ans;
    }
};
```

本页要点

连乘时先算 `fac[n] * inv[k] % p` 再乘 `inv[n - k]`，避免三数连乘溢出。

例 4.1 | 易错点与 3 组自测边界数据

自测输入 (n, k, p)	预期输出	测试意图与容易踩中的坑
`(10, 3, 2333)`	`120`	小数据普通验证，检验阶乘逆元倒推是否全对。
`(2333, 2333, 2333)`	`1`	恰好 $n = p$ 边界：拆成 $(1, 1)$ 与 $(0, 0)$ 两位，检验 `while(n || k)`。
`(1000000000000000000, 1, 2333)`	`1000000000000000000 % 2333 = 1859`	10^{18} 极大值边界，检验 `long long` 与 p 进制拆位不超时。

本页要点

重点验证 $n = p$ 边界和 $n = 10^{18}$ 极大值，确保数组不越界且无溢出。

例 4.2（[基础] 越狱 | 一本通 1618）：先读懂要求

原题题意（ ）

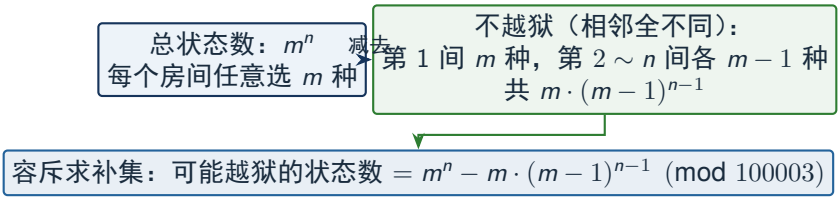
监狱有连续编号为 $1 \sim n$ 的 n 个房间，每个房间关押一个犯人。有 m 种宗教，每个犯人可能信仰其中一种。只要 ** 有相邻房间 ** 的犯人信仰相同，就可能发生越狱。求有多少种状态可能发生越狱？结果对 100003 取余。

- 输入：两个整数 m, n（$1 \le m \le 10^8,\ 1 \le n \le 10^{12}$）。
- 输出：可能越狱的状态数 $\bmod 100003$。
- 原题样例：输入 2 3 $\Longrightarrow$ 输出 6（总共 $2^3 = 8$ 种，只有 $\{0,1,0\}$ 和 $\{1,0,1\}$ 不越狱，剩下 6 种均越狱）。

本页要点

“至少有一对相邻相同”正面算非常麻烦（要用容斥枚举哪里相同），反过来算“全都不同”却只需一秒！

例 4.2 | 怎么想到的：正难则反的容斥思想



- 这正是容斥原理最经典的特例——** 正难则反（全集减去补集）**！
- 因 $n \le 10^{12}$，直接用快速幂即可在 $O(\log n)$ 内算完两项幂次。

本页要点

遇到“至少发生一次……”的计数题，先想想它的反面“一次都不发生”好不好算！

例 4.2 | 样例手推与边界检验（m=2，n=3）

代入 $m = 2, n = 3$，模数 MOD $= 100003$：

计算项	算式与结果	说明
总状态数 total	$2^3 \bmod 100003 = 8$	3 个房间每个 2 种选择
不越狱状态数 safe	$2 \times (2-1)^2 \bmod 100003 = 2$	$\{0,1,0\}$ 和 $\{1,0,1\}$
越狱状态数 ans	$(8 - 2 + 100003) \bmod 100003 = \mathbf{6}$	与样例输出完全一致！

- 如果只算一次：当 $n = 1$ 时，$m^1 - m(m-1)^0 = m - m = 0$（只有 1 个房间没有相邻房间，不可能越狱），公式依然完美成立！

本页要点

即使在 $m = 1$（$1 - 1 = 0$）或 $n = 1$（指数为 0）时，$(m-1)^{n-1}$ 的公式依然完全正确！

例 4.2 | 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. `total = qpow(m, n)`	总状态数 mod100003	内部 `a %= MOD` 防止 $m = 10^8$ 初乘溢出
2. `safe = (m%MOD) * qpow(m-1, n-1) % MOD`	合法状态数 mod100003	`m` 先取模再乘，防止 $10^8 \times 10^5$ 爆 `int`
3. `(total - safe + MOD) % MOD`	越狱状态数	必须 `+ MOD`，防止 `total < safe` 出现负数！

- 相对快速幂模板的改动：只增加了一步容斥减法，并用 `+ MOD` 处理取模后的减法负数。

本页要点

模意义下做减法 `A - B`，永远记得写成 `(A - B + MOD) % MOD`！

例 4.2 | 核心代码（code/jailbreak.cpp）

```cpp
const long long MOD = 100003;
long long qpow(long long a, long long b) {
    long long res = 1;
    for (a %= MOD; b > 0; b >>= 1, a = a * a % MOD)
        if (b & 1) res = res * a % MOD;
    return res;
}
int main() {
    long long m, n; cin >> m >> n;
    long long total = qpow(m, n);
    long long safe = (m % MOD) * qpow(m - 1, n - 1) % MOD;
    cout << (total - safe + MOD) % MOD << '\n'; // 防负数取模
}
```

本页要点

虽然数学上 total $\geq$ safe，但取模后 total 可能小于 safe，+ MOD 绝对不能省！

例 4.2 | 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
`2 3`	`6`	原题样例，检验正难则反公式。
`5 1`	`0`	$n = 1$ 最小边界，只有 1 个房间无相邻，必须输出 0。
`100000000` `1000000000000`	`69546`	$m = 10^8, n = 10^{12}$ 极大值，检验 `long long` 与 `+ MOD` 防负。

本页要点

用 `5 1`（输出 0）和极大值测一遍，确认 `qpow` 内 `b` 为 `long long`（10^{12} 会爆 `int`!）。

例 4.3（[基础] 集卡问题 / 收集邮票基础版）：先读懂要求

经典题意（对应 code/coupon_collector.cpp）

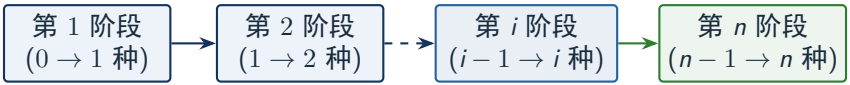
共有 n 种不同的卡片（$1 \le n \le 10^5$），每次抽奖都会 ** 等概率 ** 地随机获得其中一种（抽完放回，即每种概率始终为 $\frac{1}{n}$）。问平均需要抽多少次，才能集齐全部 n 种卡片？结果保留两位小数。

- 输入：一行一个正整数 n。
- 输出：一个浮点数（保留两位小数），表示集齐全部 n 种卡片的期望抽取次数。
- 经典样例：输入 3 $\Longrightarrow$ 输出 5.50。

本页要点

这是所有期望 DP 与概率题的母题——把“无穷步的随机过程”拆成 n 段“等待下一张新卡”。

例 4.3 | 怎么想到的：按“已集齐种类数”分段



- 无论抽了多少次，只要手头已经有 $i-1$ 种卡片，下一张抽到 ** 还没见过的新卡 ** 的概率只取决于还剩 $n-i+1$ 种，即 $p_i = \frac{n-i+1}{n}$。
- 由几何分布知该阶段期望步数 $E(X_i) = \frac{1}{p_i} = \frac{n}{n-i+1}$。
- 利用 ** 期望线性性 ** 直接累加：$E(S) = \sum_{i=1}^{n} \frac{n}{n-i+1}$。

本页要点

虽然第 i 阶段花了多少步会影响总步数，但期望线性性允许我们把各阶段期望直接相加！

例 4.3 | 样例手推与核心步骤表

当 $n=3$ 时，循环 $i=1,2,3$ 逐步累加：

循环轮次 i	分母 $n-i+1$	本轮增加 `(double)n / (n - i + 1)`	累计 `ans`
$i=1$	3	$3.0/3=1.0000$	1.0000
$i=2$	2	$3.0/2=1.5000$	2.5000
$i=3$	1	$3.0/1=3.0000$	**5.5000**

- 如果题目改成每次抽卡花 k 元钱（进阶《收集邮票》）？第 k 次抽卡花 k 元，总花费是 $\frac{S(S+1)}{2}$，就不能只算 $E(S)$，而要同时递推步数期望 $f[i]$ 和花费期望 $g[i]$！

本页要点

基础版只需要一重循环累加调和级数；进阶版在理解分段思想后也能顺理成章列出递推式。

例 4.3 | 核心代码（code/coupon_collector.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;
// 集卡问题：利用期望线性性计算集齐全部 n 种卡片的期望步数
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    int n; if (!(cin >> n)) return 0;
    double ans = 0.0;
    for (int i = 1; i <= n; ++i) {
        ans += (double)n / (n - i + 1); // 抽到第 i 种新卡的期望步数
    }
    cout << fixed << setprecision(2) << ans << '\n';
    return 0;
}
```

本页要点

代码虽短，背后的“几何分布 + 期望线性性”是解决概率期望题最常用的核心武器。

例 4.3 | 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	1.00	最小边界：只有 1 种卡，抽 1 次必齐。
3	5.50	小数据检验浮点除法 `(double)n / (n - i + 1)` 是否误写成整数除法。
100000	1209014.61	$n = 10^5$ 大数据，$O(n)$ 毫秒级出解，检验 `double` 精度与格式控制。

本页要点

注意不要把 `fixed << setprecision(2)` 漏掉，否则 10^5 级别输出会变成科学计数法 `1.20901e+06`！

高频错误对照表（组合、Lucas、容斥与期望）

题目 / 场景	常见错误写法	正确做法与原因
small_c 漏判 k > n	直接计算 inv[n - k]	下标为负导致 **RE / WA**！一旦 k > n 必须立刻 return 0。
阶乘逆元写正序	从 inv[0]=1 往后乘 i	倒数关系方向相反！必须从 inv[N] 倒推回 inv[0]=1。
容斥包含空集或符号反	mask 从 0 开始或全减	从 1 到 (1<<m)-1 枚举，记住“奇加偶减”。
隔板法漏处理 0 约束	允许为空却用 $\binom{n-1}{m-1}$	先每人预借 1 个变成正整数再插板，公式 $\binom{n+m-1}{m-1}$。
期望浮点除法截断	n / (n - i + 1)	整数除法向下取整！必须强转为 (double)n / ...。

本页要点

“倒推逆元 inv[0]=1，k>n 立刻归零，隔板可空先加 m，容斥紧记奇加偶减”！

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
RE（运行时错）	`small_c` 未判 `k > n` 导致访问 `inv[-1]`；或 `fac` 数组开小。	检查 `if (k < 0 || k > n) return 0;` 与数组大小 `>= p`。
WA（部分错）	1) 连续相乘未先取模；2) 容斥乘积 `prod` 溢出；3) 减法未 `+ MOD`。	检查 `fac[n]*inv[k]%p*inv[n-k]%p`、防溢出截断与负数取模。
WA（全错）	1) 合数模数套用 Lucas；2) 隔板法少加 m；3) 期望输出科学计数法。	核对模数是否质数，加上 `fixed << setprecision(2)`。
TLE / MLE	$n \geq 10^5$ 还用二维杨辉三角，或对每个数调用快速幂求逆元。	改用 $O(N)$ 倒推阶乘逆元或 $O(\log_p n)$ 的 Lucas 定理。

本页要点

组合数 RE 查 `k>n` 负下标；容斥错查空集与溢出；期望 WA 查整数除法和科学计数法！

提交前检查清单（六必查）

1. 预处理阶乘时，是否让 `fac[0] = 1` 与 `inv[0] = 1`（应对取 0 个或全取的情况）？
2. 求单次组合数 `small_c(n, k)` 前，是否先判断了 `k < 0 || k > n` 并直接返回 0？
3. 如果 $n \geq p$ 且 p 是小质数，是否严格套用了 Lucas 定理（而非被 0 污染的普通 `fac[]`）？
4. 组合数连乘三项时（`fac[n] * inv[k] % p * inv[n-k] % p`），有没有中间先取模防溢出？
5. 容斥原理枚举子集时，有没有剔除空集（从 `1` 到 `(1<<m)-1`）并严格对应奇加偶减？
6. 期望题中是否利用线性性 $E(\sum X_i) = \sum E(X_i)$ 拆解（不依赖独立性），并避免整数除法？

最后一步

用 $\binom{5}{0} = 1$，$\binom{5}{6} = 0$ 以及 Lucas 例 $\binom{10}{3} \bmod 7 = 1$ 自测后再提交！

本章全部练习与完成标准

1. 学校 **OJ：1658** ｜超能粒子炮・改（提高，`code/lucas_comb.cpp`）：掌握 Lucas 底座与分块前缀和递归。
2. 学校 **OJ：1618** ｜ **[HNOI2008]** 越狱（必做，`code/jailbreak.cpp`）：熟练应用“正难则反”容斥思维与防负取模。
3. 经典集卡期望（必做，`code/coupon_collector.cpp`）：掌握期望线性性拆解与几何分布 $\frac{1}{p}$；进阶可练习《收集邮票》双递推。

完成标准

能默写 $O(p)$ 阶乘逆元与 Lucas 模板；会区分“分到 0 个与至少 1 个”对应的隔板公式；理解期望线性性为何不依赖独立性。

课后安排与下一章预告

- 复习：手算验证一次 $p = 5$ 时 $0! \sim 4!$ 及其逆元，确认 `inv[i-1] = inv[i] * i % p` 的正确性。
- 练习：把《超能粒子炮・改》的 Lucas 底座封装好，并独立补全前缀和分块递归代码。
- 下一章预告：第 5 章《博弈论与 SG 函数》（全系列收官之章！）——只看石子堆异或和就能判定必胜必败，用 SG 函数秒杀所有组合博弈！

本章一句话

大组合数用 p 进制拆成小数，复杂集合用奇加偶减消重，随机过程借期望线性性化繁为简。

谢谢！