数论、组合选讲

丁尧尧

上海交通大学

February 13, 2017

目录

组合数、排列数

从 n 个对象中选 m 个排成一列的方案数称作排列数，记作 $P(n,m)$。
如果 $n=m$，则称为全排列，记作 $P(n)$。
从 n 个对象中选择 m 个的方案数称作组合数，记作 $C(n,m)$ 或 $\binom{n}{m}$。
它们满足：[1]

- $P(n,m)=\frac{n!}{(n-m)!}$
- $\binom{n}{m}=\frac{n!}{m!(n-m)!}$ （常用来算模意义下组合数）
- $\binom{n}{m}=\binom{n-1}{m-1}\binom{n-1}{m}$ （常用来算一般意义下组合数）
- $(x+y)^n=\sum_{i=0}^{n}\binom{n}{i}x^iy^{n-i}$ （二项式定理）
- $\sum_{i=0}^{n}\binom{n}{i}=2^n$ （令上面 $x=y=1$）

还有一些重要的东西：

- 可重排列
- 循环排列
- 不区分球，区分盒子（夹棍法）
- 卡特兰数列

[1] 我们将 $0!$ 看成 1

容斥原理、鸽巢原理

容斥原理：

$$|\bigcup_{i=1}^{n} A_i| = \sum_{i} |A_i| - \sum_{1\le i<j\le n} |A_i \cap A_j| + \sum_{1\le i<j<k\le n} |A_i \cap A_j \cap A_k| \cdots$$

鸽巢原理：将 n 个鸽子塞进 $n-1$ 个巢中，那么必定有一个巢有至少两个鸽子。

快速幂

如果我们要计算 a^n，那么我们可以将 n 写成二进制形式，然后将 a^n 拆成一些 a^{2^i} 的乘积，而后者可以递推来算。
如果我们的 a 特别大，大到 64 位的整型都存不下，并且是以十进制的形式输入的，那么我们可以弄十进制快速幂（类比二进制快速幂）。
快速乘也是类似的思想。

最大公约数

两个整数公共的约数称为公约数，如果这两个数不同时为 0, 那么他们中就存在最大的一个公约数，称为最大公约数，记作 $gcd(a,b)$。
两个不为 0 的整数公共的倍数称为公倍数，其中最小的正公倍数记为最小公倍数，记作 $lcm(a,b)$。
它们有如下性质:

- $gcd(a,b) = gcd(b,a)$
- $gcd(a,0) = abs(a)$
- $gcd(a,b) = gcd(a,b+ax), x \in Z$ 以上两条是我们求 gcd 的主要途径
- $ab = gcd(a,b)lcm(a,b)$ 用来求 lcm。

关于 $gcd(a,b)$ 有一个重要的事实，那就是存在整数 x,y 使得：

$$gcd(a,b) = ax + by$$

我们可以用辗转相除法给出构造性的证明。
从而我们也有了求 x,y 的方法。
事实上，如果我们让 x,y 遍历整个整数集合，那么 $ax+by$ 就会遍历所有 $gcd(a,b)$ 的倍数。
有了上面事实，我们就可以证明（虽然它们感觉很显然）：

$$a \mid bc,\ gcd(a,b) = 1 \Rightarrow a \mid c$$

$$p \mid a_1 a_2 \Rightarrow p \mid a_1 \ or\ p \mid a_2$$

从而证明唯一分解定理。

二元一次不定方程

问题： 给定 a, b，讨论下面这个二元一次不定方程解的情况：

$$ax + by = c$$

我们设 $d = gcd(a, b)$。那么：

如果 $d \nmid c$，无解

如果 $d \mid c$，那么有无数解，并且解集和方程 $\frac{a}{d}x + \frac{b}{d}y = \frac{c}{d}$ 的解集相同。这时我们可以找到 $1 = gcd(\frac{a}{d}, \frac{b}{d}) = \frac{a}{d}x_0' + \frac{b}{d}y_0'$ 的解 x_0', y_0'，从而得到原方程的一个特解 $x_0 = \frac{c}{d}x_0', y_0 = \frac{c}{d}y_0'$。整个解集就是 $x = x_0 + \frac{b}{d}t, \quad y = y_0 - \frac{a}{d}t$，其中 t 取遍整个 Z。

同余

两个数除以某个数有相同的余数是一个重要的关系，所以我们引进同余符号：

$$a \equiv b \ (mod\ m) \Leftrightarrow m \mid a - b$$

有些性质（如果后面没有模数，默认为 $(mod\ m)$）：

- $a \equiv b, b \equiv c \Rightarrow a \equiv c$
- $a \equiv b, c \equiv d \Rightarrow a + c \equiv b + d, ac \equiv bd$
- 对于非 0 的整数 c，有 $a \equiv b \ (mod\ m) \Leftrightarrow ac \equiv bc \ (mod\ mc)$
- $ac \equiv bc \ (mod\ m) \Leftrightarrow a \equiv b \ (mod\ \frac{m}{gcd(m,c)})$

欧拉函数

我们定义欧拉函数 $\varphi(n)$:

$$\varphi(n) = |\{a \in Z \mid 1 \leq a \leq n \ and \ gcd(a,n) = 1\}|$$

即 $\varphi(n)$ 表示 1 到 n 中和 n 互质的数的个数。
关于它，有以下事实:

- $\varphi(nm) = \varphi(n)\varphi(m) \quad (gcd(m,n) = 1)$ 积性函数
- $\varphi(n) = n\prod_{p|n}(1 - \frac{1}{p})$ 用于手算
- $n = \sum_{d|n}\varphi(d)$

欧拉定理

关于欧拉函数，我们还有一个重要的定理：

$$a^{\varphi(n)} \equiv 1 \ (mod\ m)\ (gcd(a, m) = 1)$$

这个定理的一个直接推论就是费马小定理：

$$a^{p-1} \equiv 1 \ (mod\ p)\ (p\ is\ a\ prime)$$

欧拉定理的证明，大致过程是说 m 的一个缩系中每个数乘以一个与 m 互素的数后还是 m 的一个缩系，然后两个缩系中每个数乘起来同余，然后就有了欧拉定理。
我们一般用欧拉定理去求逆元，或将大指数变小。

有这样一个问题，对于整数数 a，求一个数 b 满足：

$$ab \equiv 1\ (mod\ m)$$

我们可以证明上面这个式子成立当且仅当有：

$$gcd(a, m) = 1$$

并且将 b 记作 a^{-1}，称为 a 在模 m 意义下的逆。
有了这个，我们就可以在模意义下做除法操作了。

中国剩余定理

还有一类方程我们需要求解：

$$x \equiv a_i \ (mod\ m_i) \quad (i = 1, 2, \cdots, n)$$

这是一个同余方程组，x 的系数都是 1，并且满足 m_i 两两互质。
我们设 $M = m_1 m_2 m_3 \ldots m_n$，上面那个方程就等价于：

$$x \equiv \sum_{i=1}^{n} \frac{M}{m_i} (\frac{M}{m_i})^{-1} a_i \ (modM)$$

其中 $(\frac{M}{m_i})^{-1}$ 指的是关于 m_i 的逆元。
中国剩余定理除了单纯的解方程外，还为我们提供了一种思路，就是当题目中给出的模数不是质数时，我们可以把它质因分解成 $p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_s^{\alpha_s}$ 的形式，然后对每个 $p_i^{\alpha_i}$ 做（这个时候会有一些比合数更好的性质），然后再用中国剩余定理合并起来。

Lucas 定理

求组合数 $\binom{n}{m}$ 是我们经常遇到的问题，如果是要求出其具体值，我们一般是用组合数的递推公式来直接做（因为组合数增长很快，所以规模一般很小）。如果是在摸意义下，那么就可以弄得很大，而 $Lucas$ 定理就是用来处理模数是小素数（$p \leq 10^6$），但 n, m 可以很大（$n, m \leq 10^{18}$）的情况，它的定理内容是：

$$\binom{n}{m} \equiv \binom{qp+r}{sp+t} \equiv \binom{q}{s}\binom{r}{t} \ (mod\ p)$$

其中

$$n = qp + r, m = sp + t, 0 \leq r, t < p$$

我们可以对 $\binom{q}{s}$ 继续用定理，从而将 $\binom{n}{m}$ 分解成一些小的数对应组合数的乘积，并且将后者排成一排，可以看出上面部分是 n 的 p 进制分解，下面是 m 的 p 进制分解。

筛素数

我们怎么去把一定范围内的素数全部搞出来？

- $Eraosthenes$ 筛，思路是从前往后枚举每个数，每枚举到一个素数，就用它把比他大的倍数都标记为“非素数”。复杂度 $O(nlog(log(n)))$
- $Euler$ 筛，一个数一定是被它的最小素因子筛掉的。复杂度 $O(n)$

其实在 10^6 范围内，两个的速度差不多（我试验了下，10^6 时前者基本操作大概是后者的六倍，但是后者中有取模运算，所以弄得差不多快），前者比后者更显然一些，后者比前者更容易计算积性函数一些。