阶乘取模

引入

本文讨论了某一模数下阶乘计算的相关结论，并提供一种时间复杂度线性相关于模数大小的计算方法，因而该方法主要适用于模数不太大（$\sim 10^6$）的情形。除了本文介绍的方法外，根据场景不同，还可以应用 多项式技术 进行快速计算。

根据 中国剩余定理，阶乘取模问题可以转化为模数为素数幂 p^α 的情形。在处理这类问题时，常常需要对于素数 p 和正整数 n，将阶乘 $n!$ 中的所有因子 p 都提取出来，进而得到分解：

$$n! = p^{\nu_p(n!)}(n!)_p.$$

其中，$\nu_p(n!)$ 表示阶乘 $n!$ 的素因数分解中 p 的幂次，$(n!)_p$ 表示在阶乘 $n!$ 的结果中去除所有 p 的幂次得到的整数。本文将讨论 $(n!)_p$ 在素数（幂）模下的余数以及幂次 $\nu_p(n!)$ 的具体计算方法。

这种分解在解决阶乘同时出现在所求表达式的分子和分母的问题时尤为有用，比如 计算某一模数下的二项式系数。对于这类问题，分子和分母中 p 的幂次可以直接相减，而与 p 互素的部分 $(n!)_p$ 则可以利用 乘法逆元 计算。

本文还介绍了与上述问题相关的 Wilson 定理及其推广、Legendre 公式和 Kummer 定理等内容。

Wilson 定理

Wilson 定理给出了判断某个自然数是素数的一个充分必要条件。

Wilson 定理

对于自然数 $n > 1$，当且仅当 n 是素数时，$(n-1)! \equiv -1 \pmod{n}$。

证明

首先，证明对于素数 p 有 $(p-1)! \equiv -1 \pmod{p}$。对于这一点，可以利用 同余方程 或 原根 得到两种简洁的证明，此处略去不表。下面提供前置知识较少的一种证明方法：

当 $p=2$ 时，命题显然成立。下面设 $p \geq 3$，继而要证明 Z_p 中所有非零元素（即同余类）的积为 $\overline{-1}$。因为 Z_p 中所有非零元素 $\bar{a}$ 都有逆元 $\bar{a}^{-1}$，于是 Z_p 中彼此互逆的元素乘积为 $\bar{1}$。但是要注意 $\bar{a}$ 和 $\bar{a}^{-1}$ 可能相等：$\bar{a}=\bar{a}^{-1}$，当且仅当 $a^2 \equiv 1 \pmod{p}$，即

$$0 \equiv a^2 - 1 \equiv (a+1)(a-1), \pmod{p}$$

从而，$a \equiv 1 \pmod{p}$ 或 $a \equiv -1 \pmod{p}$。这说明 $\mathsf{Z}_p \setminus \{\bar{0}, \bar{1}, \overline{-1}\}$ 中所有元素的乘积为 $\bar{1}$，进而 Z_p 中所有非零元素的积为 $\overline{-1}$。

反过来，对于合数 n 的情形，要证明 $(n-1)! \not\equiv -1 \pmod{n}$。利用反证法，不妨设 $(n-1)! \equiv -1 \pmod{n}$，亦即存在整数 k 使得 $(n-1)! = kn - 1$ 成立。因为 n 是合数，必然存在素数 $p < n$ 使得 $n = pm$，所以 $(n-1)! = kpm - 1 \equiv -1 \pmod{p}$。但是，乘积 $(n-1)!$ 中必然已经出现 p，故而一定有 $(n-1)! \equiv 0 \pmod{p}$。这一矛盾就说明了 $(n-1)! \not\equiv -1 \pmod{n}$。

利用本文的记号，Wilson 定理可以写作 $(p!)_p \equiv -1 \pmod{p}$。

推广

Wilson 定理可以推广到一般模数的情形。

定理（Gauss）

对于自然数 $m > 1$，有

$$\prod_{1 \leq k < m,\, k \perp m} k \equiv \pm 1 \pmod{m}.$$

而且，余数中的 ± 1 取值为 -1 当且仅当模 m 的 原根存在，即 $m = 2, 4, p^\alpha, 2p^\alpha$ 时，其中 p 是奇素数且 α 是正整数。

证明

这个定理可以通过 模 n 整数乘法群 的结构简单地证明。此处给出思路相仿，但是较为初等的证明。

对于 $m=2$ 的情形，有 $1!=1\equiv -1 \pmod 2$。对于其他存在原根的情形，设原根为 g，则所有满足小于 m 且与它互素的正整数 k 都可以唯一地表示为 $g^i \bmod m$ 的形式，其中 $0\le i<\varphi(m)$ 且 $\varphi(m)$ 是 Euler 函数。直接验证可知，$\varphi(m)$ 一定是偶数。因为 g^i 和 $g^{\varphi(m)-i}$ 互为乘法逆元，所以在乘积中将它们两两配对，就有

$$\prod_{1\le k<m,\, k\perp m} k \equiv \prod_{i=0}^{\varphi(m)-1} g^i = g^{\varphi(m)/2} \prod_{i=1}^{\varphi(m)/2-1} g^i g^{\varphi(m)-i} \equiv g^{\varphi(m)/2} \pmod m.$$

因为 $g^{\varphi(m)/2} \bmod m$ 是唯一的不等于 $1 \bmod m$ 且乘法逆元就是它自身的元素，所以它就等于 $-1 \bmod m$。这就说明了此时的余数等于 -1。

对于模 m 的原根不存在的情形，要证明余数等于 1。为此，可以首先做质因数分解 $m=p_1^{e_1}p_2^{e_2}\cdots p_s^{e_s}$，然后应用 中国剩余定理 可知，只需要证明

$$\prod_{1\le k<m,\, k\perp m} k \equiv 1 \pmod{p_j^{e_j}}$$

对所有因子 $p_j^{e_j}$ 都成立。中国剩余定理说明，每一个可能的余数组合 $(r_1, r_2, \cdots, r_s)$，其中，$1\le r_j<p_j^{e_j}$ 且 $p_j\perp r_j$，都唯一地对应着一个 $1\le k<m$ 且 $k\perp m$ 使得 $k\equiv r_j \pmod{p_j^{e_j}}$ 成立。所以，对于某个余数 r_j，都恰好有 $\varphi(m)/\varphi(p_j^{e_j})$ 个 k 使得 $k\equiv r_j \pmod{p_j^{e_j}}$ 成立。利用这一点，可以对乘积进行分组，就有

$$\prod_{1\le k<m,\, k\perp m} k \equiv \left(\prod_{1\le r_j<p_j^{e_j},\, r_j\perp p_j} r_j\right)^{\varphi(m)/\varphi(p_j^{e_j})} \pmod{p_j^{e_j}}.$$

此处的指数 $\varphi(m)/\varphi(p_j^{e_j})=\varphi(m/p_j^{e_j})$ 要成为奇数，必然要求 $m/p_j^{e_j}=1,2$，因为欧拉函数 $\varphi(n)$ 对于 $n\ge 3$ 都是偶数。如果 p_j 是奇素数，因为模 m 的原根不存在，必然有 $m/p_j^{e_j}\ne 1,2$；如果 $p_j^{e_j}=2,4$，因为模 m 的原根不存在，必然有 $m/p_j^{e_j}$ 含有某个奇素因子，故而大于 2：这两种情形指数 $\varphi(m)/\varphi(p_j^{e_j})$ 都是偶数。而上式中括号里的项已经证明是模 $p_j^{e_j}$ 余 -1 的，所以这个幂模 $p_j^{e_j}$ 的余数一定是 1。剩余的情形只有 $p_j=2$ 且 $e_j>2$ 时，对于这个情形，可以直接证明 `

$$\prod_{1\le r_j<2^{e_j},\, r_j\perp 2} r_j \equiv 1 \pmod{2^{e_j}}.$$

仿照前文的证明思路，可以将所有 $1\le r_j<2^{e_j}$ 的奇数 r_j 两两配对而消去，那些无法配对的必然是方程 $x^2\equiv 1 \pmod{2^{e_j}}$ 的解。该方程意味着 $2^{e_j}\mid (x-1)(x+1)$。令 $x=2y+1$，就必然有 $2^{e_j-2}\mid y(y+1)$，而 y 和 $y+1$ 必然一奇一偶，所以 $y=t2^{e_j-2}$ 或 $y=t2^{e_j-2}-1$。故而，有 $x=t2^{e_j-1}\pm 1$ 且 t 是整数。模 2^{e_j} 的余数中，只有 ± 1 和 $2^{e_j-1}\pm 1$ 四个。因此，有

$$\prod_{1\le r_j<2^{e_j},\, r_j\perp 2} r_j \equiv (-1)(2^{e_j-1}-1)(2^{e_j-1}+1)\equiv 1 \pmod{2^{e_j}}.$$

这就完成了所有情形的证明。

在计算中，尤为重要的是模数为素数幂的情形：

推论

对于素数 p 和正整数 α，有

$$\prod_{1\le k<p^\alpha,\ k\perp p} k \equiv \begin{cases} 1, & p=2 \text{ and } \alpha \ge 3, \\ -1, & \text{otherwise} \end{cases} \pmod{p^\alpha}.$$

注意，左侧并非 $(p^\alpha!)_p$，因为后者还需要统计 p 的倍数的贡献。

阶乘余数的计算

本节讨论余数 $(n!)_p \bmod p^\alpha$ 的计算。

素数模的情形

算式 $(n!)_p$ 有明显的递归结构。为注意到这一点，首先考察一个具体的例子：

例子

要计算 $(32!)_5 \bmod 5$，可以做如下递归计算：

$$\begin{aligned}(32!)_5 &= 1\times 2\times 3\times 4\times \underbrace{1}_{5}\times 6\times 7\times 8\times 9\times \underbrace{2}_{10} \\ &\quad \times 11\times 12\times 13\times 14\times \underbrace{3}_{15}\times 16\times 17\times 18\times 19\times \underbrace{4}_{20} \\ &\quad \times 21\times 22\times 23\times 24\times \underbrace{1}_{25}\times 26\times 27\times 28\times 29\times \underbrace{6}_{30}\times 31\times 32 \\ &\equiv 1\times 2\times 3\times 4\times \underbrace{1}_{5}\times 1\times 2\times 3\times 4\times \underbrace{2}_{10} \\ &\quad \times 1\times 2\times 3\times 4\times \underbrace{3}_{15}\times 1\times 2\times 3\times 4\times \underbrace{4}_{20} \\ &\quad \times 1\times 2\times 3\times 4\times \underbrace{1}_{25}\times 1\times 2\times 3\times 4\times \underbrace{1}_{30}\times 1\times 2 \\ &= (1\times 2\times 3\times 4)^6\times (1\times 2)\times (\underbrace{1}_{5}\times \underbrace{2}_{10}\times \underbrace{3}_{15}\times \underbrace{4}_{20}\times \underbrace{1}_{25}\times \underbrace{1}_{30}) \pmod 5\end{aligned}$$

可以看出，利用模 5 余数的周期性，可以将这一乘积划分为若干个长度为 5 的块，每一块的唯一差异就是最后一个元素的余数。因为 32 除以 5 得到的商是 6 且余数是 2，所以，该乘积可以划分为 6 个完整的块和最后一段长度为 2 的不完整的块。因此，可以将前 6 个块除了最后一个元素之外的部分提取出来（这一部分恰好是 Wilson 定理能够解决的），再乘上最后一个不完整的块的乘积，最后乘上前 6 个块的最后一个元素的连乘积。每个块的最后一个元素都是 5 的倍数，去掉 5 的幂次后，它们的连乘积恰好是 $(6!)_5 \pmod 5$。这就将原来的问题转化为了规模更小的问题。

将该例子中的递归的结构一般化，就得到如下递推公式：

递推公式

对于素数 p 和正整数 n，有

$$(n!)_p \equiv (-1)^{\lfloor n/p \rfloor} \cdot (n \bmod p)! \cdot (\lfloor n/p \rfloor !)_p \pmod{p}.$$

证明

记 $(n)_p$ 为 n 的素因数分解中去除所有 p 的幂次的结果。于是，有

$$\begin{aligned}(n!)_p &= \prod_{k=1}^{n}(k)_p = \left(\prod_{1\le k\le n,\, k\perp p}(k)_p\right)\left(\prod_{1\le k\le \lfloor n/p\rfloor}(pk)_p\right)\\ &= \left(\prod_{i=0}^{\lfloor n/p\rfloor-1}\prod_{j=1}^{p-1}(ip+j)\right)\left(\prod_{j=1}^{n \bmod p}(\lfloor n/k\rfloor k+j)\right)\left(\prod_{1\le k\le \lfloor n/p\rfloor}(k)_p\right)\\ &\equiv \left(\prod_{j=1}^{p-1} j\right)^{\lfloor n/p\rfloor}\left(\prod_{j=1}^{n \bmod p} j\right)(\lfloor n/p\rfloor!)_p\\ &\equiv (-1)^{\lfloor n/p\rfloor}\cdot(n \bmod p)!\cdot(\lfloor n/p\rfloor!)_p \pmod p.\end{aligned}$$

这就完成了证明。下面对于该形式证明提供具体的解释。

要计算 $(n!)_p \bmod p$ 的值。仿照上面的例子，有

$$\begin{aligned}(n!)_p &= 1\cdot 2\cdot 3\cdot \ldots\cdot(p-2)\cdot(p-1)\cdot\underbrace{1}_{p}\cdot(p+1)\cdot(p+2)\cdot\ldots\cdot(2p-1)\cdot\underbrace{2}_{2p}\\ &\quad\cdot(2p+1)\cdot\ldots\cdot(p^2-1)\cdot\underbrace{1}_{p^2}\cdot(p^2+1)\cdot\ldots\cdot n \pmod p\\ &= 1\cdot 2\cdot 3\cdot\ldots\cdot(p-2)\cdot(p-1)\cdot\underbrace{1}_{p}\cdot 1\cdot 2\cdot\ldots\cdot(p-1)\cdot\underbrace{2}_{2p}\cdot 1\cdot 2\\ &\quad\cdot\ldots\cdot(p-1)\cdot\underbrace{1}_{p^2}\cdot 1\cdot 2\cdot\ldots\cdot(n \bmod p) \pmod p.\end{aligned}$$

可以清楚地看到，除了最后一个块外，阶乘被划分为几个长度相同的完整的块。

$$\begin{aligned}(n!)_p &= \underbrace{1\cdot 2\cdot 3\cdot\ldots\cdot(p-2)\cdot(p-1)\cdot 1}_{1\text{st}}\cdot\underbrace{1\cdot 2\cdot 3\cdot\ldots\cdot(p-2)\cdot(p-1)\cdot 2}_{2\text{nd}}\cdot\ldots\\ &\quad\cdot\underbrace{1\cdot 2\cdot 3\cdot\ldots\cdot(p-2)\cdot(p-1)\cdot 1}_{p\text{th}}\cdot\ldots\cdot\underbrace{1\cdot 2\cdot\ldots\cdot(n \bmod p)}_{\text{tail}} \pmod p.\end{aligned}$$

除了块的最后一个元素外，完整的块的主要部分 $(p-1)! \bmod p$ 很容易计算，可以应用 Wilson 定理：

$$(p-1)!\equiv -1 \pmod p.$$

总共有 $\left\lfloor\frac{n}{p}\right\rfloor$ 个完整的块，因此需要将 $\left\lfloor\frac{n}{p}\right\rfloor$ 写到 -1 的指数上。

最后一个部分块的值就是 $(n \bmod p)! \bmod p$，可以单独计算。

剩下的就是每个块的最后一个元素。如果隐藏已处理的元素，可以看到以下模式：

$$(n!)_p = \underbrace{\ldots\cdot 1}\cdot\underbrace{\ldots\cdot 2}\cdot\ldots\cdot\underbrace{\ldots\cdot(p-1)}\cdot\underbrace{\ldots\cdot 1}\cdot\underbrace{\ldots\cdot 1}\cdot\underbrace{\ldots\cdot 2}\cdot\cdots$$

这也是一个修正的阶乘，只是长度短得多。它是：

$$\left(\left\lfloor\frac{n}{p}\right\rfloor!\right)_p.$$

将各部分乘起来，就得到上面的递推公式。

利用该递推式做计算，递归深度为 $O(\log_p n)$。如果每次都重新计算中间那一项，那么每层计算的复杂度都是 $O(p)$ 的，总的时间复杂度是 $O(p \log_p n)$；如果对所有 $n = 1, 2, \cdots, p-1$ 都预先处理了 $n! \bmod p$，那么预处理的复杂度是 $O(p)$ 的，每层计算的复杂度都是 $O(1)$ 的，总的时间复杂度是 $O(p + \log_p n)$ 的。

在实现时，因为是尾递归，可以用迭代实现。下面的实现对前 $p-1$ 个阶乘做了预计算，如果要多次调用，可以将预计算放到函数外进行。

参考实现

```cpp
// Calculate (n!)_p mod p.
int factmod(int n, int p) {
  // Pretreatment.
  std::vector<int> f(p);
  f[0] = 1;
  for (int i = 1; i < p; ++i) {
    f[i] = (long long)f[i - 1] * i % p;
  }
  // Recursion.
  int res = 1;
  while (n > 1) {
    if ((n / p) & 1) res = p - res;
    res = (long long)res * f[n % p] % p;
    n /= p;
  }
  return res;
}
```

如果空间有限，无法存储所有阶乘，也可以将函数调用中实际用到的阶乘 $n! \bmod p$ 中的 n 都计算出来，然后对它们进行排序，从而可以在最后一次性计算出来这些阶乘的值，汇总到最终结果中，而避免存储所有阶乘的值。

素数幂模的情形

对于素数幂模的情形，可以仿照素数模的情形解决，只需要将 Wilson 定理替换成它的推广形式。本节两个结论中的 ± 1，均特指这样的定义：当模数 $p = 2$ 且 $\alpha \geq 3$ 时取 1，其余情形取 -1。

递推公式

对于素数 p 和正整数 α, n，有

$$(n!)_p \equiv (\pm 1)^{\lfloor n/p^\alpha \rfloor} \cdot \left(\prod_{1 \le j \le (n \bmod p^\alpha),\, j \perp p} j \right) \cdot (\lfloor n/p \rfloor !)_p \pmod{p^\alpha}.$$

其中，± 1 的取值如同 Wilson 定理的推广 中规定的那样。

证明

证明思路和素数模的情形完全一致。记 $(k)_p$ 为去除 k 的素因数分解中 p 的全部幂次的结果，则

$$\begin{aligned}
(n!)_p &= \prod_{1 \le k \le n} (k)_p = \left(\prod_{1 \le k \le n,\, k \perp p} (k)_p \right) \left(\prod_{1 \le k \le \lfloor n/p \rfloor} (pk)_p \right) \\
&= \left(\prod_{i=0}^{\lfloor n/p^\alpha \rfloor - 1} \prod_{1 \le j \le p^\alpha,\, j \perp p} (ip^\alpha + j)_p \right) \left(\prod_{1 \le j \le (n \bmod p^\alpha),\, j \perp p} (\lfloor n/p^\alpha \rfloor p^\alpha + j)_p \right) \left(\prod_{1 \le k \le \lfloor n/p \rfloor} (k)_p \right) \\
&\equiv \left(\prod_{1 \le j \le p^\alpha,\, j \perp p} j \right)^{\lfloor n/p^\alpha \rfloor} \cdot \left(\prod_{1 \le j \le (n \bmod p^\alpha),\, j \perp p} j \right) \cdot (\lfloor n/p \rfloor !)_p \\
&\equiv (\pm 1)^{\lfloor n/p^\alpha \rfloor} \cdot \left(\prod_{1 \le j \le (n \bmod p^\alpha),\, j \perp p} j \right) \cdot (\lfloor n/p \rfloor !)_p \pmod{p^\alpha}.
\end{aligned}$$

与素数模的情形不同之处，除了 -1 可能需要替换为 ± 1 之外，还需要注意预处理的数据的不同。对于素数幂模的情形，需要对所有不超过 p^α 的正整数 n 预处理自 1 至 n 但并非 p 的倍数的所有整数的乘积，即

$$\prod_{1 \le k \le n,\, k \perp p} k \bmod p^\alpha.$$

在素数模的情形，它退化为 $n! \bmod p$，但是该表达式在一般的素数幂的情形不再适用。

下面提供了在素数幂模的情形下计算阶乘余数的例子，以便理解上述方法：

例子

要计算 $(32!)_3 \bmod 9$，可以做如下递归计算：

$$
\begin{aligned}
(32!)_3 &= 1\times 2\times \underbrace{1}_{3}\times 4\times 5\times \underbrace{2}_{6}\times 7\times 8\times \underbrace{1}_{9}\\
&\quad\times 10\times 11\times \underbrace{4}_{12}\times 13\times 14\times \underbrace{5}_{15}\times 16\times 17\times \underbrace{2}_{18}\\
&\quad\times 19\times 20\times \underbrace{7}_{21}\times 22\times 23\times \underbrace{8}_{24}\times 25\times 26\times \underbrace{1}_{27}\\
&\quad\times 28\times 29\times \underbrace{10}_{30}\times 31\times 32\\
&\equiv 1\times 2\times \underbrace{1}_{3}\times 4\times 5\times \underbrace{2}_{6}\times 7\times 8\times \underbrace{1}_{9}\\
&\quad\times 1\times 2\times \underbrace{4}_{12}\times 4\times 5\times \underbrace{5}_{15}\times 7\times 8\times \underbrace{2}_{18}\\
&\quad\times 1\times 2\times \underbrace{7}_{21}\times 4\times 5\times \underbrace{8}_{24}\times 7\times 8\times \underbrace{1}_{27}\\
&\quad\times 1\times 2\times \underbrace{1}_{30}\times 4\times 5\\
&= (1\times 2\times 4\times 5\times 7\times 8)^3\times (1\times 2\times 4\times 5)\\
&\quad\times \left(\underbrace{1}_{3}\times \underbrace{2}_{6}\times \underbrace{1}_{9}\times \underbrace{4}_{12}\times \underbrace{5}_{15}\times \underbrace{2}_{18}\right.\\
&\qquad\left.\times \underbrace{7}_{21}\times \underbrace{8}_{24}\times \underbrace{1}_{27}\times \underbrace{1}_{30}\right) \pmod 9.
\end{aligned}
$$

将 $(32!)_3 \bmod 9$ 的算式分解的结果同样可以分为三部分：

- 完整的块：由 $1\sim 9$ 之间所有不被 3 整除的整数的乘积，共 $\lfloor 32/9\rfloor = 3$ 块；
- 尾部不完整的块：所有不被 3 整除的整数从 1 一直乘到 $32 \bmod 9$；
- 所有被 3 整除的整数的乘积，对比倒数第二个等号的结果可知，这就是它的前 $\lfloor 32/3\rfloor = 10$ 项，亦即 $(\lfloor 32/3\rfloor !)_3 \bmod 9$。

最后一个括号里的递归求解即可，这样就将原问题转化为了更小的问题。

由此，就可以得到如下递推结果：

递推结果

对于素数 p 和正整数 α, n，有

$$
(n!)_p \equiv (\pm 1)^{\sum_{j\geq \alpha}\lfloor n/p^j\rfloor} \prod_{j\geq 0} F(\lfloor n/p^j\rfloor \bmod p^\alpha),
$$

其中，$F(m) = \prod_{1\leq k\leq m,\, k\perp p} k \bmod p^\alpha$ 且 ± 1 的取值与上文所述相同。

素数幂模的情形的实现和素数模的情形类似，只有一些细节上的区别。与上文类似，同样可以将预处理放到函数外进行。

参考实现

```cpp
// Calculate (n!)_p mod pa.
int factmod(int n, int p, int pa) {
  // Pretreatment.
  std::vector<int> f(pa);
  f[0] = 1;
  for (int i = 1; i < pa; ++i) {
    f[i] = i % p ? (long long)f[i - 1] * i % pa : f[i - 1];
  }
  // Recursion.
  bool neg = p != 2 || pa <= 4;
  int res = 1;
  while (n > 1) {
    if ((n / pa) & neg) res = pa - res;
    res = (long long)res * f[n % pa] % pa;
    n /= p;
  }
  return res;
}
```

预处理的时间复杂度为 $O(p^\alpha)$，单次询问的时间复杂度为 $O(\log_p n)$。

幂次的计算

本节讨论阶乘 $n!$ 中 p 的幂次 $\nu_p(n!)$ 的计算，它可以用于计算二项式系数的余数。因为二项式系数中，分子和分母都会出现阶乘，而分子和分母中素数 p 能否互相抵消，就成为了决定最后的余数的重要因素。

Legendre 公式

阶乘 $n!$ 中素数 p 的幂次可以通过 Legendre 公式计算，而且与 n 在 p 进制下的表示有关。

Legendre 公式

对于正整数 n，阶乘 $n!$ 中含有的素数 p 的幂次 $\nu_p(n!)$ 为

$$\nu_p(n!) = \sum_{i=1}^{\infty} \left\lfloor \frac{n}{p^i} \right\rfloor = \frac{n - S_p(n)}{p-1},$$

其中，$S_p(n)$ 为 p 进制下 n 的各个数位的和。特别地，阶乘中 2 的幂次是 $\nu_2(n!) = n - S_2(n)$。

证明

因为

$$n! = 1 \times 2 \times \cdots \times p \times \cdots \times 2p \times \cdots \times \lfloor n/p \rfloor p \times \cdots \times n.$$

其中，p 的倍数的乘积为 $p \times 2p \times \cdots \times \lfloor n/p \rfloor p = p^{\lfloor n/p \rfloor} \lfloor n/p \rfloor !$，而 $\lfloor n/p \rfloor !$ 可能继续出现 p 的倍数。所以，对于幂次，有递推关系：

$$\nu_p(n!) = \lfloor n/p \rfloor + \nu_p(\lfloor n/p \rfloor !).$$

将它展开就得到 Legendre 公式。

要证明第二个等号，首先将 n 展开为 p 进制，这相当于将它写作如下和式：

$$n = n_\ell p^\ell + \cdots + n_1 p + n_0 = \sum_{k=0}^{\ell} n_k p^k.$$

因此，有

$$\begin{aligned} \nu_p(n!) &= \sum_{i=1}^{\ell} \left\lfloor \frac{n}{p^i} \right\rfloor = \sum_{i=1}^{\ell} \sum_{k=i}^{\ell} n_k p^{k-i} = \sum_{k=1}^{\ell} n_k \sum_{i=1}^{k} p^{k-i} \\ &= \sum_{k=1}^{\ell} n_k \frac{p^k - 1}{p-1} = \frac{\sum_{k=0}^{\ell} n_k p^k - \sum_{k=0}^{\ell} n_k}{p-1} = \frac{n - S_p(n)}{p-1}. \end{aligned}$$

求阶乘中素数幂次的参考实现如下：

参考实现

```c
// Obtain multiplicity of p in n!.
int multiplicity_factorial(int n, int p) {
  int count = 0;
  do {
    n /= p;
    count += n;
  } while (n);
  return count;
}
```

它的时间复杂度为 $O(\log n)$。

Kummer 定理

组合数对一个数取模的结果，往往构成分形结构，例如谢尔宾斯基三角形就可以通过组合数模 2 得到。

如果仔细分析，p 是否整除组合数其实和上下标在 p 进制下减法是否需要借位有关。这就有了 **Kummer 定理**。

Kummer 定理

素数 p 在组合数 $\binom{m}{n}$ 中的幂次，恰好是 p 进制下 m 减掉 n 需要借位的次数，亦即

$$\nu_p\left(\binom{m}{n}\right)=\frac{S_p(n)+S_p(m-n)-S_p(m)}{p-1}.$$

特别地，组合数中 2 的幂次是 $\nu_2\left(\binom{m}{n}\right)=S_2(n)+S_2(m-n)-S_2(m).$

证明

首先证明下面的表达式。为此，利用 Legendre 公式，有

$$\begin{aligned}\nu_p\left(\binom{m}{n}\right)&=\nu_p(m!)-\nu_p(n!)-\nu_p((m-n)!)\\&=\sum_{i=1}^{\infty}\left(\left\lfloor\frac{m}{p^i}\right\rfloor-\left\lfloor\frac{n}{p^i}\right\rfloor-\left\lfloor\frac{m-n}{p^i}\right\rfloor\right)\\&=\frac{S_p(n)+S_p(m-n)-S_p(m)}{p-1}.\end{aligned}$$

该表达式可以理解为 p 进制下 m 减掉 n 需要借位的次数。因为如果在计算第 i 位（最低位下标是 1）时存在不够减需要借位的情况，那么相减的结果中第 i 位之前的数字 $\left\lfloor\frac{m-n}{p^i}\right\rfloor$，其实是 m 中第 i 位之前的数字 $\left\lfloor\frac{m}{p^i}\right\rfloor$，减去一（即借掉的一），再减去 n 中第 i 位之前的数字得到的差值 $\left\lfloor\frac{n}{p^i}\right\rfloor$，所以，差值

$$\left\lfloor\frac{m}{p^i}\right\rfloor-\left\lfloor\frac{n}{p^i}\right\rfloor-\left\lfloor\frac{m-n}{p^i}\right\rfloor=1$$

当且仅当发生了一次借位；否则，该差值为 0。因此，上述表达式中的求和式就可以理解为借位发生的次数。这就得到了 Kummer 定理的文字表述。

例题

例题 HDU 2973 - YAPTCHA

给定 n, 计算

$$\sum_{k=1}^{n}\left\lfloor\frac{(3k+6)!+1}{3k+7}-\left\lfloor\frac{(3k+6)!}{3k+7}\right\rfloor\right\rfloor$$

解题思路

若 $3k+7$ 是质数，则

$$(3k+6)! \equiv -1 \pmod{3k+7}$$

设 $(3k+6)!+1 = k(3k+7)$

则

$$\left\lfloor \frac{(3k+6)!+1}{3k+7} - \left\lfloor \frac{(3k+6)!}{3k+7} \right\rfloor \right\rfloor = \left\lfloor k - \left\lfloor k - \frac{1}{3k+7} \right\rfloor \right\rfloor = 1$$

若 $3k+7$ 不是质数，则有 $(3k+7) \mid (3k+6)!$，即

$$(3k+6)! \equiv 0 \pmod{3k+7}$$

设 $(3k+6)! = k(3k+7)$，则

$$\left\lfloor \frac{(3k+6)!+1}{3k+7} - \left\lfloor \frac{(3k+6)!}{3k+7} \right\rfloor \right\rfloor = \left\lfloor k + \frac{1}{3k+7} - k \right\rfloor = 0$$

因此

$$\sum_{k=1}^{n} \left\lfloor \frac{(3k+6)!+1}{3k+7} - \left\lfloor \frac{(3k+6)!}{3k+7} \right\rfloor \right\rfloor = \sum_{k=1}^{n} [3k+7 \text{ is prime}]$$

参考代码

```cpp
#include <iostream>

constexpr int M = 1e6 + 5, N = 3 * M + 7;

bool not_prime[N];
int sum[M];

int main() {
  for (int i = 2; i < N; ++i)
    if (!not_prime[i])
      for (int j = 2; j * i < N; ++j) not_prime[j * i] = true;
  for (int i = 1; i < M; ++i) sum[i] = sum[i - 1] + !not_prime[3
* i + 7];

  int t;
  std::cin >> t;
  while (t--) {
    int n;
    std::cin >> n;
    std::cout << sum[n] << std::endl;
  }
}
```

参考资料

- 冯克勤。《初等数论及其应用》。
- Wilson's theorem - Wikipedia
- Legendre's formula - Wikipedia

本页面主要译自博文 Вычисление факториала по модулю 与其英文翻译版 Factorial modulo p。其中俄文版版权协议为 Public Domain + Leave a Link；英文版版权协议为 CC-BY-SA 4.0。内容有改动。
