卢卡斯定理

前置知识：阶乘取模

引入

本文讨论大组合数取模的求解。组合数，又称二项式系数，指表达式：

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

规模不大时，组合数可以通过 递推公式 求解，时间复杂度为 $O(nk)$；也可以在较大的素数模数 $p > n$ 下，通过计算分子和分母的阶乘在 $O(n)$ 时间内求解。但当问题规模很大（$n \sim 10^{18}$）时，这些方法不再适用。

基于 Lucas 定理及其推广，本文讨论一种可以在模数不太大 ($m \sim 10^6$) 时求解组合数的方法。更准确地说，只要模数的唯一分解 $m = \prod p_i^{e_i}$ 中所有素数幂的和（即 $\sum p_i^{e_i}$）在 10^6 规模时就可以使用该方法，因为算法的预处理大致相当于这一规模。

Lucas 定理

首先讨论模数为素数 p 的情形。此时，有 Lucas 定理：

Lucas 定理

对于素数 p，有

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

其中，当 $n < k$ 时，二项式系数 $\binom{n}{k}$ 规定为 0。

利用生成函数证明

考虑 $\binom{p}{n} \bmod p$ 的取值。因为

$$\binom{p}{n} = \frac{p!}{n!(p-n)!},$$

所以，当 $n \neq 0, p$ 时，分母中都没有因子 p，但分子中有因子 p，所以分式一定是 p 的倍数，模 p 的余数是 0；当 $n = 0, p$ 时，分式就是 1。因此，

$$\binom{p}{n} \equiv [n = 0 \vee n = p] \pmod{p}.$$

记 $f(x) = ax^n + bx^m$。一般地，由 二项式展开 和 费马小定理 有

$$\begin{aligned}(f(x))^p &= (ax^n + bx^m)^p \\ &= \sum_{k=0}^{p} \binom{p}{k} (ax^n)^k (bx^m)^{p-k} \\ &\equiv a^p x^{pn} + b^p x^{pm} \\ &\equiv a(x^p)^n + b(x^p)^m \\ &= f(x^p) \pmod{p}.\end{aligned}$$

其中，第三行的同余利用了前文说明的结论，即只有 $k = 0, p$ 时，组合数才不是 p 的倍数。

利用这一结论，考察二项式展开：

$$\begin{aligned}(1+x)^n &= (1+x)^{p\lfloor n/p \rfloor}(1+x)^{n \bmod p} \\ &\equiv (1+x^p)^{\lfloor n/p \rfloor}(1+x)^{n \bmod p} \pmod{p}.\end{aligned}$$

等式左侧中，项 x^k 的系数为

$$\binom{n}{k} \bmod p.$$

转而计算等式右侧中项 x^k 的系数。第一个因子中各项的次数必然是 p 的倍数，第二个因子中各项的次数必然小于 p，而 k 分解成这样两部分的和的方式是唯一的，即带余除法：$k = p\lfloor k/p \rfloor + (k \bmod p)$。因此，第一个因子只能贡献其 $p\lfloor k/p \rfloor$ 次项，第二个因子只能贡献其 $k \bmod p$ 次项。所以，右侧等式中 x^k 系数为两个因子各自贡献的项的系数的乘积：

$$\binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor}\binom{n \bmod p}{k \bmod p} \bmod p.$$

令两侧系数相等，就得到 Lucas 定理。

利用阶乘取模的结论证明

此处提供一种基于 阶乘取模 相关结论的证明方法，以方便和后文 exLucas 部分的方法建立联系。已知二项式系数

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

将阶乘 $n!$ 中 p 的幂次和其他因子分离，得到分解：

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

就得到二项式系数的表达式：

$$\binom{n}{k} = p^{\nu_p(n!)-\nu_p(k!)-\nu_p((n-k)!)}\frac{(n!)_p}{(k!)_p((n-k)!)_p}.$$

幂次 $\nu_p(n!)$ 和阶乘余数 $(n!)_p \bmod p$ 都有递推公式：

$$\begin{aligned}\nu_p(n!) &= \lfloor n/p\rfloor + \nu_p(\lfloor n/p\rfloor!),\\ (n!)_p &\equiv (-1)^{\lfloor n/p\rfloor}\cdot(n \bmod p)!\cdot(\lfloor n/p\rfloor!)_p \pmod p.\end{aligned}$$

前者是 Legendre 公式的推论，后者是 Wilson 定理的推论。

将递推公式代入二项式系数的表达式并整理，就得到：

$$\begin{aligned}\binom{n}{k} \equiv\ &(-p)^{\lfloor n/p\rfloor-\lfloor k/p\rfloor-\lfloor (n-k)/p\rfloor}\cdot\frac{(n \bmod p)!}{(k \bmod p)!((n-k) \bmod p)!}\\ &\cdot p^{\nu_p(\lfloor n/p\rfloor!)-\nu_p(\lfloor k/p\rfloor!)-\nu_p(\lfloor (n-k)/p\rfloor!)}\frac{(\lfloor n/p\rfloor!)_p}{(\lfloor k/p\rfloor!)_p(\lfloor (n-k)/p\rfloor!)_p} \pmod p.\end{aligned}$$

现在考察 $\lfloor n/p\rfloor-\lfloor k/p\rfloor-\lfloor (n-k)/p\rfloor$ 的取值。因为有

$$\begin{aligned}n &= \lfloor n/p\rfloor p + (n \bmod p),\\ k &= \lfloor k/p\rfloor p + (k \bmod p),\\ n-k &= \lfloor (n-k)/p\rfloor p + ((n-k) \bmod p),\end{aligned}$$

所以，利用第一式减去后两式，就得到

$$(\lfloor n/p\rfloor-\lfloor k/p\rfloor-\lfloor (n-k)/p\rfloor)p = (k \bmod p)+((n-k) \bmod p)-(n \bmod p).$$

等式右侧，前两项的和严格小于 $2p$，而第三项 $n \bmod p$ 正是前两项的和的余数，所以右侧必然非负，但小于 $2p$，又需要是 p 的倍数，就只能是 0 或 p。这说明 $\lfloor n/p\rfloor-\lfloor k/p\rfloor-\lfloor (n-k)/p\rfloor$ 只能是 0 或 1：

- 如果它是 0，那么此时也成立 $(n \bmod p) = (k \bmod p) + ((n-k) \bmod p)$。因此，上式中的第一个因子的指数为 0，该因子就等于一；第二个因子就是 $\binom{n \bmod p}{k \bmod p}$；第三个因子则由前文的展开式可知，就等于 $\binom{\lfloor n/p\rfloor}{\lfloor k/p\rfloor}$。此时，Lucas 公式成立；
- 如果它是 1，那么第一个因子的指数为 1，该因子就等于零，所以二项式系数的余数为零。同时，Lucas 定理所要证明的等式右侧的 $\binom{n \bmod p}{k \bmod p}$ 也必然是零，因为此时必然有 $(n \bmod p) < (k \bmod p)$；否则，将有

$$((n-k) \bmod p) = p + (n \bmod p) - (k \bmod p) \geq p.$$

这显然与余数的定义矛盾。

综合两种情形，就得到了所要求证的 Lucas 定理。这一证明说明，在求解素数模下组合数时，利用 Lucas 定理和利用 exLucas 算法得到的结果是等价的。

Lucas 定理指出，模数为素数 p 时，大组合数的计算可以转化为规模更小的组合数的计算。在式中，第一个组合数可以继续递归，直到 $n, k < p$ 为止；第二个组合数则可以直接计算，或者提前预处理出来。写成代码的形式就是：

示意

```
long long Lucas(long long n, long long k, long long p) {
  if (k == 0) return 1;
  return (C(n % p, k % p, p) * Lucas(n / p, k / p, p)) % p;
}
```

其中，`C(n, k, p)` 用于计算小规模的组合数。

递归至多进行 $O(\log_p n)$ 次，因而算法的复杂度为 $O(f(p) + g(p)\log_p n)$，其中，$f(p)$ 为预处理组合数的复杂度，$g(p)$ 为单次计算组合数的复杂度。

参考实现

此处给出的参考实现在 $O(p)$ 时间内预处理 p 以内的阶乘及其逆元后，可以在 $O(1)$ 时间内计算单个组合数：

参考实现

```cpp
#include <iostream>
#include <vector>

class BinomModPrime {
  int p;
  std::vector<int> fa, ifa;

  // Calculate binom(n, k) mod p for n, k < p.
  int calc(int n, int k) {
    if (n < k) return 0;
    long long res = fa[n];
    res = (res * ifa[k]) % p;
    res = (res * ifa[n - k]) % p;
    return res;
  }

 public:
  BinomModPrime(int p) : p(p), fa(p), ifa(p) {
    // Factorials mod p till p.
    fa[0] = 1;
    for (int i = 1; i < p; ++i) {
      fa[i] = (long long)fa[i - 1] * i % p;
    }
    // Inverse of factorials mod p till p.
    ifa[p - 1] = p - 1;  // Wilson's theorem.
    for (int i = p - 1; i; --i) {
      ifa[i - 1] = (long long)ifa[i] * i % p;
    }
  }

  // Calculate binom(n, k) mod p.
  int binomial(long long n, long long k) {
    long long res = 1;
    while (n || k) {
      res = (res * calc(n % p, k % p)) % p;
      n /= p;
      k /= p;
    }
    return res;
  }
};

int main() {
  int t, p;
  std::cin >> t >> p;
  BinomModPrime bm(p);
  for (; t; --t) {
    long long n, k;
    std::cin >> n >> k;
```

```cpp
    std::cout << bm.binomial(n, k) << '\n';
  }
  return 0;
}
```

该实现的时间复杂度为 $O(p+T\log_p n)$，其中，T 为询问次数。

exLucas 算法

Lucas 定理中对于模数 p 要求必须为素数，那么对于 p 不是素数的情况，就需要用到 exLucas 算法。虽然名字如此，该算法实际操作时并没有用到 Lucas 定理。它的关键步骤是 计算素数幂模下的阶乘。上文的第二个证明指出了它与 Lucas 定理的联系。

素数幂模的情形

首先考虑模数为素数幂 p^α 的情形。将阶乘 $n!$ 中的 p 的幂次和其他幂次分开，可以得到分解：

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

其中，$\nu_p(n!)$ 为 $n!$ 的素因数分解中 p 的幂次，而 $(n!)_p$ 显然与 p 互素。因此，组合数可以写作：

$$\binom{n}{k} = p^{\nu_p(n!)-\nu_p(k!)-\nu_p((n-k)!)}\frac{(n!)_p}{(k!)_p((n-k)!)_p}.$$

式子中的 $\nu_p(n!)$ 等可以通过 Legendre 公式 计算，$(n!)_p$ 等则可以通过 递推关系 计算。因为后者与 p^α 互素，所以分母上的乘积的逆元可以通过 扩展欧几里得算法 计算。问题就得以解决。

注意，如果幂次 $\nu_p(n!)-\nu_p(k!)-\nu_p((n-k)!)\geq\alpha$，余数一定为零，不必再做更多计算。

一般模数的情形

对于 m 是一般的合数的情形，只需要首先对它做 素因数分解：

$$m = p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_s^{\alpha_s}.$$

然后，分别计算出模 $p_i^{\alpha_i}$ 下组合数 $\binom{n}{k}$ 的余数，就得到 s 个同余方程：

$$\begin{cases}\binom{n}{k}\equiv r_1, & \pmod{p_1^{\alpha_1}},\\ \binom{n}{k}\equiv r_2, & \pmod{p_2^{\alpha_2}},\\ \cdots \\ \binom{n}{k}\equiv r_s, & \pmod{p_s^{\alpha_s}}.\end{cases}$$

最后，利用 中国剩余定理 求出模 m 的余数。

参考实现

最后，给出模板题目 二项式系数 的参考实现。

参考实现

```cpp
#include <iostream>
#include <vector>

// Extended Euclid.
void ex_gcd(int a, int b, int& x, int& y) {
  if (!b) {
    x = 1;
    y = 0;
  } else {
    ex_gcd(b, a % b, y, x);
    y -= a / b * x;
  }
}

// Inverse of a mod m.
int inverse(int a, int m) {
  int x, y;
  ex_gcd(a, m, x, y);
  return (x % m + m) % m;
}

// Coefficient in CRT.
int crt_coeff(int m_i, int m) {
  long long mm = m / m_i;
  mm *= inverse(mm, m_i);
  return mm % m;
}

// Binominal Coefficient Calculator Modulo Prime Power.
class BinomModPrimePower {
  int p, a, pa;
  std::vector<int> f;

  // Obtain multiplicity of p in n!.
  long long nu(long long n) {
    long long count = 0;
    do {
      n /= p;
      count += n;
    } while (n);
    return count;
  }

  // Calculate (n!)_p mod pa.
  long long fact_mod(long long n) {
    bool neg = p != 2 || pa <= 4;
    long long res = 1;
    while (n > 1) {
      if ((n / pa) & neg) res = pa - res;
```

```
      res = res * f[n % pa] % pa;
      n /= p;
    }
    return res;
  }

 public:
  BinomModPrimePower(int p, int a, int pa) : p(p), a(a), pa(pa),
f(pa) {
    // Pretreatment.
    f[0] = 1;
    for (int i = 1; i < pa; ++i) {
      f[i] = i % p ? (long long)f[i - 1] * i % pa : f[i - 1];
    }
  }

  // Calculate Binom(n, k) mod pa.
  int binomial(long long n, long long k) {
    long long v = nu(n) - nu(n - k) - nu(k);
    if (v >= a) return 0;
    auto res = fact_mod(n - k) * fact_mod(k) % pa;
    res = fact_mod(n) * inverse(res, pa) % pa;
    for (; v; --v) res *= p;
    return res % pa;
  }
};

// Binominal Coefficient Calculator.
class BinomMod {
  int m;
  std::vector<BinomModPrimePower> bp;
  std::vector<long long> crt_m;

 public:
  BinomMod(int n) : m(n) {
    // Factorize.
    for (int p = 2; p * p <= n; ++p) {
      if (n % p == 0) {
        int a = 0, pa = 1;
        for (; n % p == 0; n /= p, ++a, pa *= p);
        bp.emplace_back(p, a, pa);
        crt_m.emplace_back(crt_coeff(pa, m));
      }
    }
    if (n > 1) {
      bp.emplace_back(n, 1, n);
      crt_m.emplace_back(crt_coeff(n, m));
    }
  }

  // Calculate Binom(n, k) mod m.
  int binomial(long long n, long long k) {
```

```
    long long res = 0;
    for (size_t i = 0; i != bp.size(); ++i) {
      res = (bp[i].binomial(n, k) * crt_m[i] + res) % m;
    }
    return res;
  }
};

int main() {
  int t, m;
  std::cin >> t >> m;
  BinomMod bm(m);
  for (; t; --t) {
    long long n, k;
    std::cin >> n >> k;
    std::cout << bm.binomial(n, k) << '\n';
  }
  return 0;
}
```

该算法在预处理时将模数 m 分解为素数幂，然后对所有 p^α 预处理了自 1 至 p^α 所有非 p 倍数的自然数的乘积，以及它在中国剩余定理合并答案时对应的系数。预处理的时间复杂度为 $O(\sqrt{m}+\sum_i p_i^{\alpha_i})$。每次询问时，复杂度为 $O(\log m+\sum_i \log_{p_i} n)$，复杂度中的两项分别是计算逆元和计算幂次、阶乘余数的复杂度。

习题

- Luogu3807【模板】卢卡斯定理
- SDOI2010 古代猪文 卢卡斯定理
- Luogu4720【模板】扩展卢卡斯
- Ceizenpok's formula

本页面最近更新：2025/8/24 15:29:16，更新历史

发现错误？想一起完善？ 在 GitHub 上编辑此页！

本页面贡献者：Enter-tainer, c-forrest, GitPinkRabbit, Great-designer, TonyYin0418, Xeonacid, EntropyIncreaser, ksyx, MegaOwIer, sshwy, Henry-ZHR, iamtwz, ouuan, Sheng-Horizon, CornWorld, IceySakura, Ir1d, LuoYisu, Marcythm, megakite, Menci, shawlleyw, StudyingFather, Tiphereth-A, whongzhong, YOYO-UIAT