模逆元

本文介绍模意义下乘法运算的逆元，并讨论它的常见求解方法。

基本概念

非零实数 $a \in \mathbf{R}$ 的乘法逆元就是它的倒数 a^{-1}。类似地，数论中也可以定义一个整数 a 在模 m 意义下的逆元 $a^{-1} \bmod m$，或简单地记作 a^{-1}。这就是 **模逆元**（modular multiplicative inverse），也称作 **数论倒数**。

> **逆元**
>
> 对于非零整数 a, m，如果存在 b 使得 $ab \equiv 1 \pmod m$，就称 b 是 a 在模 m 意义下的 **逆元**（inverse）。

这相当于说，b 是线性同余方程 $ax \equiv 1 \pmod m$ 的解。根据 线性同余方程 的性质可知，当且仅当 $\gcd(a, m) = 1$，即 a, m 互素时，逆元 $a^{-1} \bmod m$ 存在，且在模 m 的意义下是唯一的。

单个逆元的求法

利用扩展欧几里得算法或快速幂法，可以在 $O(\log m)$ 时间内求出单个整数的逆元。

扩展欧几里得算法

求解逆元，就相当于求解线性同余方程。因此，可以使用 扩展欧几里得算法 在 $O(\log \min\{a, m\})$ 时间内求解逆元。同时，由于逆元对应的线性方程比较特殊，可以适当地简化相应的步骤。

参考实现

C++

```cpp
// Extended Euclidean algorithm.
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;
  }
}

// Returns the modular inverse of a modulo m.
// Assumes that gcd(a, m) = 1, so the inverse exists.
int inverse(int a, int m) {
  int x, y;
  ex_gcd(a, m, x, y);
  return (x % m + m) % m;
}
```

Python

```python
# Extended Euclidean algorithm.
def ex_gcd(a, b):
    if b == 0:
        return 1, 0
    else:
        x1, y1 = ex_gcd(b, a % b)
        x = y1
        y = x1 - (a // b) * y1
        return x, y


# Returns the modular inverse of a modulo m.
# Assumes that gcd(a, m) = 1, so the inverse exists.
def inverse(a, m):
    x, y = ex_gcd(a, m)
    return (x % m + m) % m
```

这一算法适用于所有逆元存在的情形。

快速幂法

这一方法主要适用于模数是素数 p 的情形。此时，由 费马小定理 可知对于任意 $a \perp p$ 都有

$$a \cdot a^{p-2} = a^{p-1} \equiv 1 \pmod p.$$

根据逆元的唯一性可知，逆元 $a^{-1} \bmod p$ 就等于 $a^{p-2} \bmod p$，因此可以直接使用 快速幂 在 $O(\log p)$ 时间内计算：

参考实现

C++

```cpp
// Binary exponentiation.
int pow(int a, int b, int m) {
  long long res = 1, po = a;
  for (; b; b >>= 1) {
    if (b & 1) res = res * po % m;
    po = po * po % m;
  }
  return res;
}

// Returns the modular inverse of a prime modulo p.
int inverse(int a, int p) { return pow(a, p - 2, p); }
```

Python

```python
# Returns the modular inverse of a prime modulo p.
# Use built-in pow function.
def inverse(a, p):
    return pow(a, p - 2, p)
```

当然，理论上，这一方法可以利用 欧拉定理 推广到一般的模数 m 的情形，即利用 $a^{\varphi(m)-1} \bmod m$ 计算逆元。但是，单次求解 欧拉函数 $\varphi(m)$ 并不容易，因此该算法在一般情况下效率不高。

多个逆元的求法

有些场景下，需要快速处理出多个整数 $a_1, a_2, \cdots, a_n$ 在模 m 意义下的逆元。此时，逐个求解逆元，总共需要 $O(n \log m)$ 的时间。实际上，如果将它们统一处理，就可以在 $O(n + \log m)$ 的时间内求出所有整数的逆元。

考虑序列 $\{a_i\}$ 的前缀积：

$$S_0 = 1,\ S_i = a_i S_{i-1},\ i = 1, 2, \cdots, n.$$

只要每个 a_i 都与 m 互素，它们的乘积 S_n 就与 m 互素。因此，可以通过前文所述算法求出 $S_n^{-1} \bmod m$ 的值。因为乘积的逆元就是逆元的乘积，所以，从 S_n^{-1} 出发，反向遍历序列就能求出每个 S_i 的逆元：

$$S_{i-1}^{-1} = a_i S_i^{-1} \bmod m,\ i = n, n-1, \cdots, 1.$$

由此，单个 a_i 的逆元可以通过下式计算：

$$a_i^{-1} = S_{i-1}S_i^{-1} \bmod m,\ i = 1, 2, \cdots, n.$$

参考实现如下：

算法中，只求了一次单个元素的逆元，因此总的时间复杂度是 $O(n + \log m)$ 的。

线性时间预处理逆元

如果要预处理前 n 个正整数在素数模 p 下的逆元，还可以通过本节将要讨论的递推关系在 $O(n)$ 时间内计算。这一方法常用于组合数计算中前 n 个正整数的阶乘的倒数的预处理。

对于 $1 < i < p$ 的正整数 i，考察带余除法：

$$p = \left\lfloor \frac{p}{i} \right\rfloor i + (p \bmod i).$$

将该等式对素数 p 取模，就得到

$$0 \equiv \left\lfloor \frac{p}{i} \right\rfloor i + (p \bmod i) \pmod p.$$

将等式两边同时乘以 $i^{-1}(p \bmod i)^{-1}$ 就得到

$$i^{-1} \equiv -\left\lfloor \frac{p}{i} \right\rfloor (p \bmod i)^{-1} \pmod p.$$

这就是用于线性时间递推求逆元的公式。由于 $p \bmod i < i$，这一公式将求解 $i^{-1} \bmod p$ 的问题转化为规模更小的问题 $(p \bmod i)^{-1} \bmod p$。因此，从 $1^{-1} \bmod p = 1$ 开始，对每个 i 顺次应用该公式，就可以在 $O(n)$ 时间内获得前 n 个整数的逆元。

参考实现如下：

参考实现

C++

```cpp
// Precomputes modular inverses of all integers from 1 to n modulo prime p.
std::vector<int> precompute_inverses(int n, int p) {
  std::vector<int> res(n + 1);
  res[1] = 1;
  for (int i = 2; i <= n; ++i) {
    res[i] = (long long)(p - p / i) * res[p % i] % p;
  }
  return res;
}
```

Python

```python
# Precomputes modular inverses of all integers from 1 to n modulo prime p.
def precompute_inverses(n, p):
    res = [0] * (n + 1)
    res[1] = 1
    for i in range(2, n + 1):
        res[i] = (p - p // i) * res[p % i] % p
    return res
```

这一算法只适用于模数是素数的情形。对于模数 m 不是素数的情形，无法保证递推公式中得到的 $m \bmod i$ 仍然与 m 互素，因而递推所需要的 $(m \bmod i)^{-1}$ 可能并不存在。一个这样的例子是 $m=8, i=3$。此时，$m \bmod i = 2$，不存在模 m 的逆元。

另外，得到该递推公式后，一种自然的想法是直接递归求解任意一个数 a 的逆元。每次递归时，都利用递推公式将它转化为更小的余数 $p \bmod a$ 的逆元，直到余数变为 1 时停止。目前尚不[redacted]这样做的复杂度[1]，因此，推荐使用前文所述的常规方法求解。

习题

- LOJ 110 乘法逆元
- LOJ 161 乘法逆元 2
- LOJ 2605「NOIP2012」同余方程
- Luogu P2054「AHOI2005」洗牌
- LOJ 2034「SDOI2016」排列计数

参考资料与注释

- Modular multiplicative inverse - Wikipedia

---

1. riteme 在知乎上的回答 中指出，这样做理论上已知的复杂度的上界是 $O(p^{1/3+\varepsilon})$，而在实际随机数据中的表现接近于 $O(\log p)$。 ↩

本页面最近更新：2025/8/23 00:58:04，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：Ir1d, Xeonacid, Enter-tainer, sshwy, StudyingFather, MegaOwIer, PeterlitsZo, hsfzLZH1, iamtwz, jifbt, Marcythm, ouuan, stevebraveman, Tiphereth-A, abc1763613206, buggg-hfc, c-forrest, Chrogeek, Early0v0, Great-designer, Henry-ZHR, hqztrue, ImpleLee, JellyGoat, ksyx, lhhxxxxx, Menci, MioChyan, n-WN, Phemon, shawlleyw, Siyuan, skr2005, thredreams, Tiooo111, WAAutoMaton, Zhaoyangzhen