[toc]

本文介绍模意义下乘法运算的逆元（Modular Multiplicative Inverse），并介绍如何使用扩展欧几里德算法（Extended Euclidean algorithm）求解乘法逆元

逆元简介

如果一个线性同余方程 $ax \equiv 1 \pmod b$，则 x 称为 $a \bmod b$ 的逆元，记作 a^{-1}。

如何求逆元

扩展欧几里得法

模板代码

```
void exgcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1, y = 0;
    return;
  }
  exgcd(b, a % b, y, x);
  y -= a / b * x;
}
```

快速幂法

因为 $ax \equiv 1 \pmod b$；

所以 $ax \equiv a^{b-1} \pmod b$（根据 费马小定理）；

所以 $x \equiv a^{b-2} \pmod b$。

然后我们就可以用快速幂来求了。

模板代码

```
inline int qpow(long long a, int b) {
  int ans = 1;
  a = (a % p + p) % p;
  for (; b; b >>= 1) {
    if (b & 1) ans = (a * ans) % p;
    a = (a * a) % p;
  }
  return ans;
}
```

线性求逆元

求出 $1,2,...,n$ 中每个数关于 p 的逆元。

如果对于每个数进行单次求解，以上两种方法就显得慢了，很有可能超时，所以下面来讲一下如何线性（$O(n)$）求逆元。

首先，很显然的 $1^{-1} \equiv 1 \pmod p$；

note: "证明" 对于 $\forall p \in \mathbf{Z}$，有 $1 \times 1 \equiv 1 \pmod p$ 恒成立，故在 p 下 1 的逆元是 1，而这是推算出其他情况的基础。

其次对于递归情况 i^{-1}，我们令 $k = \lfloor \frac{p}{i} \rfloor$，$j = p \bmod i$，有 $p = ki + j$。再放到 $\mod p$ 意义下就会得到：$ki+j \equiv 0 \pmod p$；

两边同时乘 $i^{-1} \times j^{-1}$：

$kj^{-1}+i^{-1} \equiv 0 \pmod p$

$i^{-1} \equiv -kj^{-1} \pmod p$

再带入 $j = p \bmod i$，有 $p = ki + j$，有：

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

我们注意到 $p \bmod i < i$，而在迭代中我们完全可以假设我们已经知道了所有的模 p 下的逆元 $j^{-1}, j < i$。

故我们就可以推出逆元，利用递归的形式，而使用迭代实现：

$$ i^{-1} \equiv \begin{cases} 1, & \text{if } i = 1, \\ -\lfloor\frac{p}{i}\rfloor (p \bmod i)^{-1}, & \text{otherwises}. \end{cases} \pmod p $$

note: "代码实现"

```
inv[1] = 1;
for (int i = 2; i <= n; ++i) {
  inv[i] = (long long)(p - p / i) * inv[p % i] % p;
}
```

使用 $p-\lfloor \dfrac{p}{i} \rfloor$ 来防止出现负数。

另外我们注意到我们没有对 `inv[0]` 进行定义却可能会使用它：当 $i | p$ 成立时，我们在代码中会访问 `inv[p % i]`，也就是 `inv[0]`，这是因为当 $i | p$ 时不存在 i 的逆元 i^{-1}。线性同余方程 中指出，如果 i 与 p 不互素时不存在相应的逆元（当一般而言我们会使用一个大素数，比如 $10^9 + 7$ 来确保它有着有效的逆元）。因此需要指出的是：如果没有相应的逆元的时候，`inv[i]` 的值是未定义的。

另外，根据线性求逆元方法的式子：$i^{-1} \equiv -kj^{-1} \pmod p$

递归求解 j^-1, 直到 $j=1$ 返回 1。

中间优化可以加入一个记忆化来避免多次递归导致的重复，这样求 $1,2,...,n$ 中所有数的逆元的时间复杂度仍是 $O(n)$。

注意：如果用以上给出的式子递归进行单个数的逆元求解，目前已知的时间复杂度的上界为 $O(n^{\frac 1 3})$，具体请看 知乎讨论。算法竞赛中更好地求单个数的逆元的方法有扩展欧几里得法和快速幂法。

线性求任意 n 个数的逆元

上面的方法只能求 1 到 n 的逆元，如果需要求任意给定 n 个数（$1 \le a_i < p$）的逆元，就需要下面的方法：

首先计算 n 个数的前缀积，记为 s_i，然后使用快速幂或扩展欧几里得法计算 s_n 的逆元，记为 sv_n。

因为 sv_n 是 n 个数的积的逆元，所以当我们把它乘上 a_n 时，就会和 a_n 的逆元抵消，于是就得到了 a_1 到 a_{n-1} 的积逆元，记为 sv_{n-1}。

同理我们可以依次计算出所有的 sv_i，于是 a_i^{-1} 就可以用 $s_{i-1} \times sv_i$ 求得。

所以我们就在 $O(n + \log p)$ 的时间内计算出了 n 个数的逆元。

note: "代码实现"

```
    s[0] = 1;
    for (int i = 1; i <= n; ++i) s[i] = s[i - 1] * a[i] % p;
    sv[n] = qpow(s[n], p - 2);
    // 当然这里也可以用 exgcd 来求逆元,视个人喜好而定.
    for (int i = n; i >= 1; --i) sv[i - 1] = sv[i] * a[i] % p;
    for (int i = 1; i <= n; ++i) inv[i] = sv[i] * s[i - 1] % p;
```

逆元例题

乘法逆元

乘法逆元 2

「NOIP2012」同余方程

「AHOI2005」洗牌

「SDOI2016」排列计数