数论—线性同余方程、乘法逆元

众所周知：

说明

如果爆 `int` 请自行开 `long long` 或边读边模，或高精度处理。

同余

定义

若 $a \bmod m = b \bmod m$，则称 a 与 b 关于模 m 同余，记为 $a \equiv b \pmod m$.

同余的性质

1. 反身性：$a \equiv a \pmod m$；
2. 对称性：若 $a \equiv b \pmod m$，则 $b \equiv a \pmod m$；
3. 传递性：若 $a \equiv b \pmod m$、$b \equiv c \pmod m$，则 $a \equiv c \pmod m$；

4. 同余式相加：若 $a \equiv b \pmod m$、$c \equiv d \pmod m$，则 $a \pm c \equiv b \pm d \pmod m$；
5. 同余式相乘：若 $a \equiv b \pmod m$、$c \equiv d \pmod m$，则 $a \times c \equiv b \times d \pmod m$；
6. 乘方：若 $a \equiv b \pmod m$，则 $a^k \equiv b^k \pmod m$；
7. 除法1：若 $ka \equiv kb \pmod{km}$，则 $a \equiv b \pmod m$；
8. 除法2：若 $ka \equiv kb \pmod m$，则 $a \equiv b \pmod{m/\gcd(k,m)}$；

线性同余方程

形式

关于 x 的方程，形如 $ax \equiv n \pmod b$，则称之为线性同余方程（Linear Congruence Equation）。

一般要求求出特解，或 $x \in [0, b-1]$ 的通解。

求解方法

方程 $ax \equiv n \pmod b$ 可以理解为 $ax + by = n$，其中 y 为一个整数。

- 证明如下：

  因为 $ax + by = n,$
  所以 $(ax + by) \bmod b = n \bmod b,$

  即 $ax \bmod b + by \bmod b = n \bmod b,$
  因为 $by \bmod b = 0,$

  所以 $ax \bmod b = n \bmod b,$
  转换为同余方程的形式就是 $ax \equiv n \pmod b.$

因此原方程转化为 $ax + by = n$，接下来就是扩展欧几里得算法的事情了；
详见：https://www.cnblogs.com/RainPPR/p/gcd-bezouts-exgcd.html

解的判断

扩展欧几里得算法只能求解 $ax + by = \gcd(a, b)$ 的情况，
所以只有当 $n = k \times \gcd(a, b)$，$k \in \mathbb{Z}^+$，才可以用扩展欧几里得算法求解。

- 证明如下：

可以求出一组 x_0、y_0，使得 $ax_0+by_0=\gcd(a,b)$，

等式两边同时乘以 k，便得到：$akx_0+bky_0=k\times\gcd(a,b)$，

因此可得到 $\begin{cases} x=kx_0 \\ y=ky_0 \end{cases}$，

此时便有 $ax+by=n$.

特解到通解

下面假设有解：

我们已经将 $ax\equiv n \pmod b$ 转化为 $ax+by=n$，并通过扩展欧几里得算法解出来一个通解 x_0.

然后请看：https://www.cnblogs.com/RainPPR/p/gcd-bezouts-exgcd.html

设 $t=\dfrac{\operatorname{lcm}(a,b)}{a}$，则有通解 $x=x_0+kt$，其中 $k\in\mathbb{Z}$。

特殊化的线性同余方程

考虑方程 $ax\equiv 1 \pmod b$，也就是上面的 $n=1$。

此时存在解的条件为 $k\times\gcd(a,b)=n=1$，也就是 $k=1$ 时的情况：$\gcd(a,b)=1$，即 a 与 b 互质，这样就可以用扩展欧几里得算法求解 $ax+by=1$ 了。

所以此时的通解公式也可以化简为 $x=x_0+kb$：

证明：$t=\dfrac{\operatorname{lcm}(a,b)}{a}$，而 $\operatorname{lcm}(a,b)=ab$，所以就有 $t=b$。

代码实现

```
// ax = 1 (mod b)
int lieu1(int a, int b)
{
    int x, y;
    int d = exgcd(a, b, x, y);
    if (d != 1)
        return -1;
    return (x % b + b) % b;
}
```

```
// ax = n (mod b)
int lieu(int a, int b, int n)
{
    int x, y;
    int d = exgcd(a, b, x, y);
    if (n % d != 0)
        return -1;
    int t = b / d;
    return (x % t + t) % t;
}
```

乘法逆元

有理数取模

加减法：$(a \pm b) \bmod p = (a \bmod p \pm b \bmod p) \bmod p$.

乘法：$(a \times b) \bmod p = (a \bmod p \times b \bmod p) \bmod p$.

那除法呢？举例可知 $\frac{a}{b} \bmod p$ 不一定等于 $\frac{a \bmod p}{b \bmod p}$.

如 $\frac{10}{2} \bmod 3 = 5 \bmod 3 = 2$，而 $\frac{10 \bmod 3}{2 \bmod 3} = \frac{1}{2}$.

乘法逆元的定义

若 $\frac{a}{b} \bmod p = (a \times x) \bmod p$,

则称 x 为 b 的模 p 意义下的乘法逆元（或 x 为 $b \bmod p$ 的逆元），记作 $x = b^{-1}$

思路

根据 $\frac{a}{b} \bmod p = (a \times x) \bmod p$ 可以写出同余方程：$\frac{a}{b} \equiv a \times x \pmod p$

两边同时乘以 $\frac{b}{a}$ 可以得到：$bx \equiv 1 \pmod p$；或者可以理解为 x 在模 p 意义下等价于 $\frac{1}{b}$。

转化一下就是 $xb + kp = 1$，而 $xb + kp = \gcd(b, p)$,

因此逆元并不是普遍存在的，条件是 $\gcd(b, p) = 1$，也就是 b 与 p 互质。

扩展欧几里得算法求逆元

上面已经得到了 $bx \equiv 1 \pmod p$ 及 $xb + kp = 1$，而这就是上面讲到的特殊化的线性同余方程，可以使用扩展欧几里得算法求逆元。

详见上面：线性同余方程。

快速幂求逆元

> 前置知识：快速幂、费马小定理
>
> 若 p 为素数，$\gcd(a, p) = 1$，则 $a^{p-1} \equiv 1 \pmod p$。
> 证明见：https://oi-wiki.org/math/number-theory/fermat/

仅当 p 是质数时，即 $\gcd(b, p) = 1$ 时，也可以用快速幂求逆元：

上面已得 $bx \equiv 1 \pmod p$，
根据费马小定理，$b^{p-1} \equiv 1 \pmod p$，
可以转化为 $b \times b^{p-2} \equiv 1 \pmod p$，
而我们要求的是 $bx \equiv 1 \pmod p$。

因此可得 $x = b^{p-2}$。

代码实现

```cpp
// s1: exgcd
int inv1(int a, const int p) {
    int x, y;
    exgcd(a, p, x, y);
    return (x % p + p) % p;
}
```

```cpp
// s2: pow
int inv2(int a, const int p) {
    return quick_pow(a, p - 2, p);
}
```

线性求逆元

线性求任意 n 个数的逆元

给定长度为 n 的序列 a（$1 \leq a_i < p$），求序列每个数的逆元。

> - a_i，表示原序列，即给定的序列；
> - $s_i = \prod_{i=1}^{n} a_i$，表示原序列的前缀积。

> - $inv_i = {a_i}^{-1}$，表示原序列的乘法逆元，即待求的序列；
> - $sv_i = {s_i}^{-1} = \prod_{i=1}^{n} sv_i$，表示原序列前缀积的乘法逆元，根据逆元性质也等于原序列乘法逆元的前缀积。

1. 计算给定序列 a_i 的前缀积，记为 s_i；
2. 使用快速幂或扩展欧几里得法计算 s_n 的逆元，记为 sv_n；
3. 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；这样就得到了 a_1 到 a_{n-1} 的积逆元，记为 sv_{n-1}；
4. 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。

```
// 计算前缀积
s[0] = 1;
for (int i = 1; i <= n; ++i) s[i] = s[i - 1] * a[i] % p;
// 计算全部乘法逆元的前缀积
sv[n] = quick_pow(s[n], p - 2, p);
// 递推前缀积、求序列的乘法逆元
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;
```

▼ 来自 OI-Wiki 的代码

```
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;
```

特化：线性求 1 ~ n 的逆元

即原序列 $a_i = i$，此时有更加快速的方法，但是这里不讲（见 OI-Wiki 内）。

我们在此就简化原程序。

```
s[0] = 1;
for (int i = 1; i <= n; ++i) s[i] = s[i - 1] * i % p;
sv[n] = quick_pow(s[n], p - 2, p);
for (int i = n; i >= 1; --i) sv[i - 1] = sv[i] * i % p;
for (int i = 1; i <= n; ++i) inv[i] = sv[i] * s[i - 1] % p;
```

例题

线性同余方程

▼ 点击查看代码

题目：P1082 同余方程

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

int main()
{
    ll a = rr, b = rr;
    ll x = 0, y = 0;

    exgcd(a, b, x, y);
    printf("%lld", (x % b + b) % b);
    return 0;
}
```

快速幂求逆元

▼ 点击查看代码

题目：P2613 有理数取余

```
const ll MOD = 19260817;

ll qpow(ll a, ll b, const ll p, ll res = 1)
{
    for (; b; b >>= 1)
        b & 1 ? res = res * a % p, a = a *a % p : a = a * a % p;
    return res;
}

int main()
{
    ll a = read(), b = read();
    if (b == 0)
        printf("Angry!\n"), exit(0);
    ll res = a * qpow(b, MOD - 2, MOD) % MOD;
    printf("%lld\n", res);
    return 0;
}
```

线性求 1 ~ n 的逆元

▼ 点击查看代码

题目：P3811 模意义下的乘法逆元

```
typedef long long ll;

const int N = 3e6 + 10;

ll s[N], sv[N];

ll qpow(ll a, ll b, const ll p, ll r = 1)
{
    for (; b; b >>= 1)
        b & 1 ? r = r * a % p, a = a * a % p : a = a * a % p;
    return r;
}

int main()
{
    const int n = rr;
    const ll p = rr;

```

```
    s[0] = 1;
    for (int i = 1; i <= n; ++i)
        s[i] = s[i - 1] * i % p;

    sv[n] = qpow(s[n], p - 2, p);
    for (int i = n; i; --i)
        sv[i - 1] = sv[i] * i % p;

    for (int i = 1; i <= n; ++i)
        printf("%lld\n", sv[i] * s[i - 1] % p);
    return 0;
}
```

线性求 n 数的逆元

▼ 点击查看代码

题目：P5431 模意义下的乘法逆元 2

求：$\sum_{i=1}^{n}\frac{k^i}{a_i}$.

```
typedef long long ll;

const int N = 5e6 + 10;

ll a[N];
ll s[N], sv[N];

ll qpow(ll a, ll b, const ll p, ll r = 1)
{
    for (; b; b >>= 1)
        b & 1 ? r = r * a % p, a = a *a % p : a = a * a % p;
    return r;
}

int main()
{
    const int n = rr;
    const ll p = rr, k = rr;

    s[0] = 1;
    for (int i = 1; i <= n; ++i)
        a[i] = rr, s[i] = s[i - 1] * a[i] % p;
```

```
    sv[n] = qpow(s[n], p - 2, p);
    for (int i = n; i; --i)
        sv[i - 1] = sv[i] * a[i] % p;

    ll res = 0, kt = k;
    for (int i = 1; i <= n; ++i)
        res = (res + kt * (sv[i] * s[i - 1] % p) % p) % p, kt = kt * k % p;

    printf("%lld\n", res);
    return 0;
}
```

Reference

[1] https://oi-wiki.org/math/number-theory/fermat/

[2] https://oi-wiki.org/math/number-theory/inverse/

[3] https://oi-wiki.org/math/number-theory/linear-equation/

合集：学习笔记

标签：算法 ， 学习笔记