中国剩余定理

Chinese Remainder Theorem

中国剩余定理

Theorem：设正整数 $m_1,m_2,\cdots,m_k$ 两两互质，则一元线性同余方程组

$$\begin{cases}x\equiv a_1\pmod{m_1}\\x\equiv a_2\pmod{m_2}\\\cdots\\x\equiv a_k\pmod{m_k}\end{cases}$$

有整数解，并且解在模 $M=\prod\limits_{i=1}^km_i$ 下唯一，为

$$x=\left(\sum_{i=1}^ka_iM_i{M_i}^{-1}\right)\bmod M$$

其中 $M_i=\frac{M}{m_i}$，${M_i}^{-1}$ 是 M_i 在模 m_i 意义下的逆元。

Proof: 首先证明上述解的正确性：将 x 代入原方程可知，和式的第 i 项正好满足方程，而其余项均为 0；其次证明上述解的唯一性：假设 x,y 均是其解，则 $\forall i\in\{1,2,\cdots,k\}$, $m_i\mid x-y$，又 m_i 两两互质，故 $M\mid x-y$，在模 M 意义下只能 $x=y$。证毕。

Understanding：设 n_i 满足 $n_i\equiv a_i\pmod{m_i}$，即 n_i 满足了第 i 个方程。那么欲使 $\sum\limits_{i=1}^kn_i$ 成为满足这个方程的解，需要保证除 n_i 以外的其他 n_j 都是 m_i 的倍数。换一个角度看，也就是要保证每个 n_i 都满足：(1) $n_i\equiv a_i\pmod{m_i}$；(2) $M_i\mid n_i$. 于是构造出 $n_i=a_iM_i{M_i}^{-1}$ 便可以保证上述两点都成立，而 $x=\sum\limits_{i=1}^kn_i$ 就是原方程组的解。

Application：解一元线性同余方程组；模数不保证是质数，但是没有平方因子，可以拆开求解后用 **CRT** 合并。

Code:

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

inline LL inv(LL x, LL MOD){
    LL res, y;
    exgcd(x, MOD, res, y);
    ((res %= MOD) += MOD) %= MOD;
    return res;
}

LL CRT(){
    LL x = 0;
    for(int i = 1; i <= n; i++){
        (x += a[i] * M[i] % M[0] * inv(M[i], m[i]) % M[0]) %= M[0];
    }
    return x;
}
```

扩展中国剩余定理

Theorem：当 $m_1, m_2, \cdots, m_k$ 不是相互互质时，同余方程组

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \cdots \\ x \equiv a_k \pmod{m_k} \end{cases}$$

的解由以下递推的步骤给出：设前 r 个方程中，$M = \mathrm{lcm}\{m_i\}$，x 是满足前 r 个方程的一个解，则 $x + k \cdot M$ 是前 r 个方程的通解。现在加入第 $r+1$ 个方程，则只需找到一个 k 使 $x + k \cdot M \equiv a_{r+1} \pmod{m_{r+1}}$，那么 $x + k \cdot M$ 就是新的 x。容易用扩展欧几里得算法解出 k，更新 x 与 M 即可。

Application：解同余方程组。

Code：

```
LL fmul(LL x, LL y, LL MOD){
    x %= MOD;
    y %= MOD;
    LL res = 0;
    while(y){
        if(y & 1)   (res += x) %= MOD;
        y >>= 1;
        (x <<= 1) %= MOD;
    }
    return res;
}

LL exgcd(LL a, LL b, LL &x, LL &y){
    if(b == 0){
        x = 1;
        y = 0;
        return a;
    }
    LL gcd = exgcd(b, a % b, x, y);
    LL t = x;
    x = y;
    y = t - (a / b) * y;
    return gcd;
}

LL exCRT(int n, LL a[], LL m[]){
    LL M = m[1], x = a[1];
    LL k, y;
    for(int i = 2; i <= n; i++){
        LL c = ((a[i] - x) % m[i] + m[i]) % m[i];
        LL g = exgcd(M, m[i], k, y);
        ((k %= m[i]) += m[i]) %= m[i];
        k = fmul(k, c / g, m[i] / g);
        x += k * M;
        M = M / g * m[i];
        ((x %= M) += M) %= M;
    }
    ((x %= M) += M) %= M;
    return x;
}
```