

目录

1	MATH	3
1.1	等差数列求和公式	3
1.2	带模的快速乘	3
1.3	带模的快速幂	3
1.4	矩阵快速幂	4
1.5	斐波那契矩阵算法:	5
1.6	矩阵递推的一般形式:	5
1.6.1	指数变量的处理	7
1.6.2	嵌套矩阵	7
1.7	因子分解	7
1.7.1	唯一分解定理	8
1.7.2	因子分解扩展	8
1.8	同余定理	8
1.9	GCD 与 LCM	9
1.9.1	更相减损法模板: (认为 $a > b$)	9
1.9.2	关于 GCD 的一些常用的性质:	9
1.9.3	最大公因数	10
1.10	欧拉函数	10
1.10.1	性质	10
1.10.2	计算方法:	10
1.10.3	线性筛得到素数与欧拉函数	11
1.11	扩展欧几里德算法与二元一次方程的整数解	11
1.11.1	拓展欧几里德算法	11
1.12	逆元	12
1.12.1	求逆元	12
1.13	拓展欧几里德	13
1.13.1	扩展欧几里得解线性方程 $ax+by=c$	13
1.13.2	拓展欧几里德求解同余方程组 $a * x \equiv b(modm)$ 的 x 解.	14
1.14	中国剩余定理	14
1.14.1	模互质的情况	14
1.14.2	模不互质的情况	15
1.15	素数	15
1.15.1	一些定理	15
1.15.2	素数分布	16
1.15.3	筛质数	16
1.15.4	Miller Rabinn 素数测试	17
1.15.5	模板	17
1.15.6	Pollard-Rho 算法大数质因子分解	19
1.16	高次同余	22
1.16.1	BSGS 算法	22
1.17	组合数学	22

1.17.1	组合数	22
1.17.2	组合数的性质	22
1.17.3	组合数一些求和公式	22
1.17.4	朱世杰恒等式	23
1.17.5	范德蒙恒等式	23
1.17.6	二阶求和公式	23
1.17.7	李善兰恒等式	23
1.17.8	求组合数	23
1.17.9	抽屉原理	26
1.17.10	容斥原理	26
1.17.11	盒子装球问题	27
1.18	康托展开	28
1.18.1	正向康托展开	28
1.18.2	逆向康托展开	29
1.18.3	错排数	29
1.19	母函数	30
1.19.1	普通型母函数求组合方案数	30
1.19.2	指数型母函数求排列数	31
1.20	特殊计数	32
1.20.1	Catalan 数	32
1.21	Stirling 数	33
1.21.1	第一类 Stirling 数	33
1.22	第二类 Stirling 数	34
1.23	概率和数学期望	35
1.23.1	期望 dp	35
1.24	高斯消元板子	35
1.24.1	整数版本	35
1.24.2	解线性同余方程组, 取模版本	39
1.24.3	求解异或方程组的版	40
1.24.4	浮点类型的版本	42
1.25	异或空间	45
1.25.1	线性基的性质:	45
1.25.2	求异或空间的基	45
1.25.3	根据线性基, 求能异或出最大的数	45
1.25.4	[HDU 3949] XOR 【线性基 _XOR 第 k 小】整数集合中能异或出的求第 k 小的数	46
1.25.5	前缀线性基	47
1.26	八数码问题有解判断	47
1.27	多项式乘法 FFT	47
1.27.1	FFT 计算 A+ 或-B 集合 C :	49
1.28	min25 筛求积性函数的前缀和	49
1.28.1	求 $1e10$ 以内的素数和模板:	54
1.29	反素数	58
1.29.1	求解代码,dfs, 在 n 的范围之内, 枚举质因子	58

1.29.2 按照这种思路, 还可以求因子个数恰好等于某个数的最小的数.	59
1.30 约瑟夫环递推公式,	59
1.30.1 递推公式	59
1.30.2 求解第 k 个出列的人	59
1.31 POJ - 2886 每次指定往后走的步数或往前走步数的约瑟夫环, 求第 k 个	60
1.32 加法与异或运算性质:	62
1.33 求满足 $x + y + z \bmod n = 0$, 且 $0 \leq x < y < z < n$ 的三元组个数:	62
1.34 数论分块	62
1.35 求和性质:	63

[TOC]

1 MATH

1.1 等差数列求和公式

$$S_n = n * a_1 + n(n-1)d/2 \text{ or } S_n = n(a_1 + a_n)/2$$

1.2 带模的快速乘

```
inline ll qmult(ll a, ll b, ll mod){
    ll ans = 0;
    while( b > 0 ){
        if( b&1 ) ans = (ans + a) % mod;
        a = ( a + a ) % mod;
        b >>= 1;
    }
    return ans;
}
```

1.3 带模的快速幂

```
ll binaryPow(ll a,ll b,ll m){
    ll ans = 1;
    while(b){
        if(b & 1){
            ans = ans * a % m;
        }
        a = a * a % m;
        b >>= 1;
    }
```

```

    }
    return ans;
}

```

1.4 矩阵快速幂

```

const int MAXN=2;
const int MOD=1e4;
class Matrix{
public:
    int m[MAXN][MAXN];
    Matrix(){
        memset(m,0,sizeof(m));
    }
};

Matrix Multi(Matrix a,Matrix b){
    Matrix res;
    for(int i=0;i<MAXN;i++){
        for(int j=0;j<MAXN;j++){
            for(int k=0;k<MAXN;k++){
                res.m[i][j]=(res.m[i][j]+a.m[i][k]*b.m[k][j]%MOD)%MOD;//取模
            }
        }
    }
    return res;
}

Matrix fastm(Matrix a,ll n){
    Matrix res;
    for(int i=0;i<MAXN;i++){
        res.m[i][i]=1;
    }
    while(n){
        if(n&1){
            res=Multi(res,a);
        }
        a=Multi(a,a);
        n>>=1;
    }
    return res;
}

```

1.5 斐波那契矩阵算法：

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}_{n \text{ times}}.$$

斐波那契通项公式

$$f(n) = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

1.6 矩阵递推的一般形式：

想办法转化为

$$A(n-1) * C = A(n)$$

结构

增加常数

在增加系数的基础上，我们可以继续的增加常数，例如下式：

$$f(n) = a f(n-1) + b f(n-2) + c$$

根据上面的推导经验，由于我们的右边要构造成 $A(n-1) \cdot C$ 的结构，为了保证递推关系，在这种情况下可以进行**扩维操作**。

$$\begin{cases} f(n) & = a f(n-1) + b f(n-2) + c \\ f(n-1) & = f(n-1) \\ c & = c \end{cases}$$

$$\begin{bmatrix} f(n) \\ f(n-1) \\ c \end{bmatrix} = \begin{bmatrix} a & b & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(n-1) \\ f(n-2) \\ c \end{bmatrix}$$

由此，对于增加常数或者增加齐次的项数这种情况，可以使用上述方法，通过扩维来扩展矩阵对多项式的表达。

图 1: /img/MatrixTrans

指数变量的处理

其实很多情况下，并不是单纯的 $f(n)$ 的单一函数，也有可能含有 $g(n)$ 的形式存在于代数项式中。如果我们想用矩阵来表示递推关系式，必须要满足 $g(n)$ 在乘积的情况下，表现出自变量 n 自增的情况。符合这种条件的就是**指数函数**。

例如下式：

$$f(n) = a c^n + b f(n-1) + d$$

我们将指数函数用 $g(n)$ 来表示，并且可以发现其中有这么一个规律：

$$g(n) = c^n$$

$$g(n) = g(n-1) \times g(1) = c^{n-1}c$$

这里 $g(n)$ 的规律，其实就是我上面所说的对函数的乘积表现为自变量的加和。

所以这里可以如此构造 $f(n)$ 的矩阵递推式：

$$\begin{bmatrix} f(n) \\ c^n \\ d \end{bmatrix} = \begin{bmatrix} b & ac & 1 \\ 0 & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f(n-1) \\ c^{n-1} \\ d \end{bmatrix}$$

如此，含有指数函数 $g(n)$ 为项式的情况我们也可以通过矩阵快速幂来求解。

图 2: ./img/cn

1.6.1 指数变量的处理

1.6.2 嵌套矩阵

嵌套矩阵

通过上面的总结，其实我们解决的核心问题就是将递推公式转化成矩阵形式即可。继续来发散性思维，如果递推公式中已经有矩阵，那么是不是也可以使用相同的关系式思路来转化问题呢？

答案肯定是可以的。这里给出一道例题 [POJ-3233 Matrix Power Series](#)。

题目描述了如果存在下列等式：

$$S_k = A + A^1 + A^2 + \dots + A^k = \sum_{i=1}^k A^i$$

其实根据这种求和式子我们可以立马推出：

$$S_k = A \cdot S_{k-1} + A$$

泛型这是一个 k 和 $k - 1$ 的递推公式，另外有一个常数项。根据前文的一些推导经验，我们来构造多项式和矩阵表示：

$$\begin{cases} S(n) = A \cdot S(n-1) + A \\ A = A \end{cases}$$

由于 $S(n)$ 和 A 都是矩阵，所以前文在构造矩阵的时候，其中的单位 1 都要改成单位矩阵 E 。

$$\begin{bmatrix} S(n) \\ A \end{bmatrix} = \begin{bmatrix} A & E \\ 0 & E \end{bmatrix} \begin{bmatrix} S(n-1) \\ A \end{bmatrix}$$

如此，矩阵的嵌套问题也就解决了。

图 3: img/M

1.7 因子分解

```

int fac[MAXN+10], cnt[MAXN+10];
int getFac(int n){
    int num=0, sum=0, m=sqrt(n+0.5);
    for(int k=2; k<=m; k++){
        sum=0;
        while(n%k==0){
            n/=k; sum++;
        }
        if(sum!=0){
            fac[num]=k; cnt[num++]=sum;
        }
    }
    if(n!=1){
        fac[num]=n;
        cnt[num++]=1;
    }
    return num; //返回的是因子个数;
}

```

优化：用素数除

1.7.1 唯一分解定理

对于任意一个正整数 n ，一定可以被唯一分解为若干个质数的乘积的形式： $n = p_1^{a_1} * p_2^{a_2} * \dots * p_k^{a_k}$.

1.7.2 因子分解扩展

1.7.2.1 因子个数 正整数 n 的所有不同因子的总个数. 计算公式： $D = (a_1 + 1) \times (a_2 + 1) \times \dots \times (a_k + 1)$.

1.7.2.2 因子求和 正整数 n 的所有不同因子的总和. 计算公式： $S = \prod_{i=1}^k [1 + p_1 + p_1^2 + \dots + p_i^{a_i}]$

等比数列求和式改写： $S = \prod [(p_i^{a_i+1} - 1) / (p_i - 1)]$

1.7.2.3 阶乘的因子分解 给定正整数 n ，求 $n!$ 的因子分解式中质因子 p 的数量，可以用以下公式求解：

$$S(p) = [n/p + n/(p^2) + n/(p^3) + \dots + n/(p^k)], \text{ 其中 } p^k \leq n$$

时间复杂度为 $O(\log(n))$.

1.8 同余定理

- 传递性： $a \equiv b \pmod{m}, b \equiv c \pmod{m}, a \equiv c \pmod{m}$

- 除法: $ac \equiv bc \pmod{m}, c! = 0, a \equiv b \pmod{m/\gcd(c, m)}$
- $a \equiv b \pmod{m}, n|m$ 则 $a \equiv b \pmod{n}$
- $a \equiv b \pmod{m_i} (i = 1, 2 \dots n), a \equiv b \pmod{[m_1, m_2, \dots m_n]}$

1.9 GCD 与 LCM

补充一点: 求解 gcd 问题可以使用两种方法: 更相减损法和辗转相除法 (欧几里得算法) 但是在遇到高精度取模的问题时, 可考虑使用更相减损来代替.

1.9.1 更相减损法模板: (认为 $a > b$)

原理: 欧几里得算法的一个特例:

$$\gcd(a - nb, b) = \gcd(a, b) \quad \gcd(a, b) = \gcd(a, b - a)$$

推广到 n 个数:

$$(a, b, c) = (a, b - a, c - b)$$

根据这个式子, 可以把原数组转化成差分数组

```
int gcd(int a, int b)
{
    if(b == 0) return a;
    else return gcd(b, a - b);
}
```

```
int gcd(int a, int b){
    return b==0?a:gcd(b, a%b);
}
```

```
int lcm(int a, int b){
    return a/gcd(a, b)*b;
}
```

1.9.2 关于 GCD 的一些常用的性质:

- (1. 结合律) $\gcd(a, b, c) = \gcd(\gcd(a, b), c)$.
- (2. 区间) $\gcd(a_1, \dots, a_r) = \gcd(\gcd(a_1, \dots, a_{m-1}), \gcd(a_m, \dots, a_r))$.
- (3. 分配律) $\gcd(ka, kb) = k * \gcd(a, b)$.
- (4. 互质) 若 $\gcd(a, b) = p$, 则 a/p 与 b/p 互质.
- (5. 线性变换) $\gcd(a + k*b, b) = \gcd(a, b)$.

(6. 因子分解) $\text{GCD}(a,b)=\prod[\pi^{\min(a_i,b_i)}]$.

7. $\text{gcd}(a,b)=\text{gcd}(a,-b)$

8. 求:

$$\begin{aligned}\text{gcd}(\text{lcm}(a,b), \text{lcm}(a,c)) &= \text{gcd}\left(\frac{ab}{\text{gcd}(a,b)}, \frac{ac}{\text{gcd}(a,c)}\right) \\ &= a \times \text{gcd}\left(\frac{b}{\text{gcd}(a,b)}, \frac{c}{\text{gcd}(a,c)}\right) = a \times \frac{\text{gcd}(b,c)}{\text{gcd}(a,b,c)} = \text{lcm}(a, \text{gcd}(b,c))\end{aligned}$$

1.9.3 最大公因数

(1. 结合律) $\text{LCM}(a,b,c)=\text{LCM}(\text{LCM}(a,b),c)$.

(2. 分配律) $\text{LCM}(ka,kb)=k*\text{LCM}(a,b)$.

(3. 因子分解) $\text{LCM}(a,b)=\prod[\pi^{\max(a_i,b_i)}]$.

1.10 欧拉函数

1.10.1 性质

- 积性函数 $\varphi(mn) = \varphi(m) * \varphi(n)$ 当 m,n 互质时.
- 除了 $N=2, \varphi(N)$ 都是偶数;
- 当 N 为奇数时, $\varphi(2 * N) = \varphi(N)$; $\varphi(1) = 1$
- 当 N 是质数时, $\varphi(N) = N - 1$;
- 若 N 是质数 p 的 k 次幂 $\varphi(N) = p^k - p^{(k-1)} = (p-1)p^{(k-1)}$, 因为除了 p 的倍数之外, 其他数都与 N 互质;
- 如果 $i \bmod p = 0, p$ 是素数那么 $\varphi(i * p) = p * \varphi(i)$
- 如果 $i \bmod p \neq 0, p$ 为素数. 那么 $\varphi(i * p) = \varphi(i) * (p - 1)$

1.10.2 计算方法:

$$\varphi(n) = \prod (p_i^{a_i-1}) * (p_i - 1).$$

$$n = p_1^{a_1} p_2^{a_2} p_3^{a_3} \dots p_k^{a_k}$$

$$\varphi(x) = x \prod_{i=1}^n \left(1 - \frac{1}{p_i}\right)$$

通式:

//求一个数的欧拉函数

```
ll euler(ll x){
    ll ans=x; //最终答案
    for(ll i=2; i*i<=x; i++){
```

```

if(x%i==0)////找到 a 的质因数
{
    ans=ans/i*(i-1);//先进行除法是为了防止中间数据的溢出
    while(x%i==0) x/=i;//x 通过质因子分解 x/=i 质因数
}
}
if(x>1) ans=ans/x*(x-1);
return ans;
}

```

1.10.3 线性筛得到素数与欧拉函数

```

//check[i] 0 表示是质数,1 表示是和数
int prime[Maxn+10],check[Maxn+10],phi[Maxn+10],num=0;
void GetPhi(){
    phi[1]=1;
    memset(check,0,sizeof(check));
    for(int i=2;i<Maxn;i++){
        if(!check[i]){
            prime[num++]=i;phi[i]=i-1;
        }
        for(int j=0;j<num;j++){
            if(i*prime[j]>Maxn)break;
            check[i*prime[j]]=1;//不是素数
            if(i%prime[j]==0){
                phi[i*prime[j]]=phi[i]*prime[j];break;
            }
            else{
                phi[i*prime[j]]=phi[i]*(prime[j]-1);
            }
        }
    }
    return;
}
}

```

1.11 扩展欧几里德算法与二元一次方程的整数解

1.11.1 拓展欧几里德算法

```

11 exgcd(11 a,11 b,11 &x,11 &y){
    if(a==0&&b==0)return -1;
    if(b==0){
        x=1;y=0;
        return a;
    }
    11 ans=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return ans;
}

```

变量 x 和 y 中存储了方程 $ax+by=\gcd(a,b)$ 的一组整数解;

函数的返回值是 $\gcd(a,b)$, 若返回-1, 则无解;

1.12 逆元

1.12.1 求逆元

1.12.1.1 费马小定理求逆元 费马小定理: p 为质数,a 为任意自然数, 且 a,p 互质, 则

$$a^p \equiv a \pmod{p}$$

图 4: fermat

所以

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

图 5: little

将 a^{p-1} 拆成 $a^{p-2} * a$ 就是 a 的逆元快速幂求逆元;

要求 mod 是质数且与 a 互质

时间复杂度 $O(\log n)$

1.12.1.2 欧拉定理

- 欧拉定理: 若 a 和 p 互质, 则 $a^{\varphi(p)} \bmod p = 1$.
- 所以逆元: $\text{inv}(a) = a^{\varphi(p)-1} \bmod p$.
- 计算过程首先要求出欧拉函数, 然后使用快速幂优化

(只要求 a 与 mod 互质, 需要欧拉函数与快速幂)

1.12.1.3 线性打表求逆元 条件: 互质

```
const int MAXN=1e5;
const ll mod=1e9+7;
ll inv[MAXN+10];
void getInv(){
    inv[1]=1;
    for(ll i=2;i<=MAXN;i++){
        inv[i]=(mod-mod/i)*inv[mod%i]%mod;
    }
    return ;
}
```

复杂度 $O(n)$

1.12.1.4 扩展欧几里德求逆元 即求解同余方程 $a * x \equiv 1(mod\ m)$ 求 x, 要求 a,m 互质

1.12.1.5 通用方法处理除法 条件 $b|a$

$$\frac{a}{b} mod\ p = \frac{a\ mod\ (b * p)}{b}$$

1.13 拓展欧几里德

只要求 a 与 mod 互质, 时间复杂度 $O(\log(n))$

1.13.1 扩展欧几里得解线性方程 $ax+by=c$

线性方程有解的充分必要条件是 $\gcd(a,b)$ 可以整除 c.

```
bool LinearEqu(int a,int b,int c,int &x,int &y){
    int d=exgcd(a,b,x,y);
    if(c%d==0){
        int k=c/d;x*=k;y*=k;
        return true;//有解
    }
    return false;//无解
    //返回一组解，可能为负
}
```

通解的求法：若 (x_0, y_0) 是线性方程 $ax+by=c$ 的一组特解，那么对于任意的整数 t ：

$x = x_0 + (b/\gcd(a, b)) * t, y = y_0 - (a/\gcd(a, b)) * t$ 都是线性方程的解。

1.13.2 拓展欧几里德求解同余方程组 $a * x \equiv b(\text{mod } m)$ 的 x 解.

```
bool ModularEqu(int a,int b,int m,int &x0){
    int x,y,k;
    int d=exgcd(a,m,x,y);
    if(b%d==0){
        x0=x*(b/d)%m;k=m/d;x0=(x0%k+k)%k;
        return true;
    }
    return false;
}
```

解法：首先将方程改写为 $ax-my=b$ 的形式，然后使用拓展欧几里德求出一组特解 (x_0, y_0) 。

如果题目要求找到最小的正整数解，可以令 $k=n/\gcd(a,n)$ ，这样 x 的最小正整数解可以通过表达式 $x=(x_0\%k+k)\%k$ 求出。

1.14 中国剩余定理

- 中国剩余定理：设正整数 N 满足线性同余方程组 $N \equiv a_i(\text{mod } p_i)$ ，其中 $1 \leq i \leq n, p_i$ 两两互质，则 $N = \sum[a_i * W_i * \text{inv}(W_i, p_i)] \% M$ 。

其中, $M = \prod p_i, W_i = M/p_i, \text{inv}(W_i, p_i)$ 表示 W_i 在模 p_i 下的逆元。

- 特别注意：这里的模 p_i 必须两两互质。

1.14.1 模互质的情况

```
//N 方程个数
int N,pp=1;
int exgcd(int a,int b,int &x,int &y);
int inv(int n,int m){
    int x,y,d=exgcd(n,m,x,y);
    return (x%m+m)%m;
}
//解 N \equiv a_i(\text{mod } p_i)
int China(int p[],int a[]){
    int ans=0;
    for(int i=0;i<N;i++)pp*=p[i];
    for(int i=0;i<N;i++){
```

```

    int W=pp/p[i];
    ans=(ans+a[i]*W*inv(W,p[i]))%pp;
}
return (pp+ans%pp)%pp;//返回满足条件的最小整数解
}

```

1.14.2 模不互质的情况

```

/*
下标从 1 开始
求解  $x \equiv a_i \pmod{m_i}$  的同余方程组
*/
ll excrt(ll ai[], ll mi[], ll n){
    ll x, y, k;
    ll M=mi[1], ans=ai[1];
    for(ll i=2; i<=n; i++){
        ll a=M, b=mi[i], c=(ai[i]-ans%b+b)%b;
        ll gcd=exgcd(a, b, x, y), bg=b/gcd;//要用到扩展欧几里德
        if(c%gcd!=0) {
            return -1; //无解 因为答案可以等于-1, 所以可用 flag 判断有无解
        }
        //if(bg==0)while(1);
        x=qmult(x, c/gcd, bg);//有溢出风险要用到快速乘
        ans+=x*M;
        M*=bg;//M 为前 k 个模数的 lcm
        ans=(ans%M+M)%M;
    }
    return (ans%M+M)%M;
}

```

1.15 素数

1.15.1 一些定理

素数定理：不超过 x 的质数的总数近似于 $x/\ln(x)$.

推论：第 n 个素数的大小： $O(n\log(n))$

素数的间隔：相邻两个质数的差值非常小，估算在 $\ln^2(x)$ 以内.

- 在 10^7 的范围以内，质数的个数为 664579 个.

- $1e9$ 范围内相邻两个质数的最大间隔只有 282.
- 所有除 2 以外的质数个位数字都是 1、3、7、9.
- 任何一个大于 2 的偶数都可以表示为两个质数的和.

1.15.2 素数分布

Prime number theorem (illustrated by selected values n from 10^2 to 10^{14})			
n	$\pi(n)$ = number of primes less than or equal to n	$\frac{\pi(n)}{n}$ = proportion of primes among the first n numbers	$\frac{1}{\log n}$ = predicted proportion of primes among the first n numbers
10^2	25	0.2500	0.2172
10^4	1,229	0.1229	0.1086
10^6	78,498	0.0785	0.0724
10^8	5,761,455	0.0570	0.0543
10^{10}	455,052,511	0.0455	0.0434
10^{12}	37,607,912,018	0.0377	0.0362
10^{14}	3,204,941,750,802	0.0320	0.0310

图 6: primenum

1.15.3 筛质数

```

int primes[1005]; //素数数组
bool is_prime[Maxn]; //1 是素数 0 是合数
void sieve(int n){
    for(int i=0; i<=n; i++) is_prime[i]=true; //可能会慢
    is_prime[0]=is_prime[1]=false;
    for(int i=2; i*i<=n; i++){
        if(is_prime[i]){
            for(int j=i*i; j<=n; j+=i){
                is_prime[j]=0;
            }
        }
    }
    //标记

    return ;
}

```


1.15.3.1 埃氏筛法筛素数

```
const int MAXN=1e7;
int primes[MAXN+10],NUM=0;
bool isnot_prime[MAXN+10]; //0 表示是素数,1 表示不是素数
int sumPrime[MAXN];
void GetPrime2(){
    isnot_prime[1] = 0; // 不用初始化, 会快
    for (int i = 2; i <= MAXN; i++) {
        if (!isnot_prime[i]){
            primes[NUM++] = i;
        }
        for (int j = 0; j < NUM && i * primes[j] <= MAXN; j++) {
            isnot_prime[i * primes[j]] = 1;
            if (i % primes[j] == 0)
                break;
        }
    }
}
```

1.15.3.2 线性筛

1.15.4 Miller Rabinn 素数测试

- 用 Miller Rabin 快速判断一个 $< 2^{63}$ 的数是不是素数. 的数是不是素数
- 时间复杂度: $O(k * \log_2 n)$

1.15.4.1 依据

- 费马小定理: 若 p 是质数, a 为整数, 且 $(a,p)=1$, 则有 $a^{p-1} \equiv 1(mod p)$
- 二次探测定理: 若 p 是质数, 且 $0 < x < p$ 则方程 $x^2 \equiv 1(mod p)$ 的解为 $x = 1$, 或者 $x = p - 1$

1.15.5 模板

```
typedef unsigned long long ll;
//typedef long long ll;
//ll*ll 可能会溢出, 所以乘法化加法
/* *****
```

```

* Miller_Rabin 算法进行素数测试
* 速度快可以判断一个  $< 2^{63}$  的数是不是素数
*
*****/
#include<time.h>
#include<stdlib.h>
const int S = 8; //随机算法判定次数一般 8~10 就够了
// 计算  $ret = (a*b)\%c$   $a, b, c < 2^{63}$ 
ll mult_mod(ll a, ll b, ll c){
    a%=c;
    b%=c;
    ll ret=0;
    ll tmp=a;
    while(b){
        if(b&1){
            ret+=tmp;
            if(ret>c)ret-=c; //直接取模慢得多
        }
        tmp<<=1;
        if(tmp>c)tmp-=c;
        b>>=1;
    }
    return ret;
}

// 计算  $ret = (a^n)\%mod$ 
ll pow_mod(ll a, ll n, ll mod){
    ll ret=1;
    ll tmp=a%mod;
    while(n){
        if(n&1)ret=mult_mod(ret, tmp, mod);
        tmp=mult_mod(tmp, tmp, mod);
        n>>=1;
    }
    return ret;
}

// 通过  $a^{(n-1)}=1(mod n)$  来判断  $n$  是不是素数
//  $n - 1 = x * (2^t)$ 
// 中间使用二次判断
// 是合数返回 true, 不一定是合数返回 false
bool check(ll a, ll n, ll x, ll t){
    ll ret = pow_mod(a, x, n);
    ll last = ret;

```

```

for(int i = 1; i <= t; i++){
    ret = mult_mod(ret, ret, n);
    if(ret == 1 && last != 1 && last != n-1) return true; //合数
    last = ret;
}
if(ret != 1) return true; // 费马小定理
else return false;
}

//*****
// Miller_Rabin 算法
// 是素数返回 true, (可能是伪素数)
// 不是素数返回 false
//*****
bool Miller_Rabin(ll n){
    if( n < 2) return false;
    if( n == 2) return true;
    if( (n&1) == 0) return false; //偶数
    ll x = n - 1;
    ll t = 0;
    while( (x&1) == 0 ){x >>= 1; t++;}

    srand(time(NULL)); /* ***** */

    for(int i = 0; i < S; i++){
        ll a = rand()%(n-1) + 1;
        if( check(a, n, x, t) )
            return false;
    }
    return true;
}

```

1.15.6 Pollard-Rho 算法大数质因子分解

1e18

$$n^{1/4} * \log(n)$$

```

#include<iostream>
#include<cstdio>
#include<iomanip>
#include<algorithm>
#include<cstring>
#include<cstdlib>

```

```

#include<ctime>
#include<queue>
#include<vector>
#include<stack>
#include<map>
#include<set>
#define ull unsigned long long
#define lb long double
#define ll long long
#define debug(x) cout<<"###"<<x<<"###"<<endl;
using namespace std;
inline ll Abs(ll x){return x<0?-x:x;}//取绝对值
inline ll gcd(ll x,ll y){//非递归求 gcd
    ll z;
    while(y){z=x;x=y;y=z%y;}
    return x;
}
inline ll qMult(ull x,ull y,ll p){//O(1) 快速乘 (防爆 long long)
    return (x*y-(ull)((lb)x/p*y)*p+p)%p;
}
inline ll qPow(ll x,ll y,ll p){//快速幂
    ll res=1;
    while(y){
        if(y&1)res=qMult(res,x,p);
        x=qMult(x,x,p); y>>=1;
    }return res;
}
inline bool Miller_Rabin(ll x,ll p){//mille rabin 判质数
    if(qPow(x,p-1,p)!=1)return 0;//费马小定理
    ll y=p-1,z;
    while(!(y&1)){//二次探测
        y>>=1; z=qPow(x,y,p);
        if(z!=1&&z!=p-1)return 0;
        if(z==p-1)return 1;
    }return 1;
}
inline bool prime(ll x){ if(x<2)return 0;//mille rabin 判质数
    if(x==2||x==3||x==5||x==7||x==43) return 1;
    return
        ⇨ Miller_Rabin(2,x)&&Miller_Rabin(3,x)&&Miller_Rabin(5,x)&&Miller_Rabin(7,x)&&Miller_Rabin(43,x);
}
inline ll Miller_Rabin(ll p){//求出 p 的非平凡因子

```

```

ll x,y,z,c,g; int i,j;//先摆出来 (z 用来存 (y-x) 的乘积)
while(1){//保证一定求出一个因子来
    y=x=rand()%p;//随机初始化
    z=1; c=rand()%p;//初始化
    i=0,j=1;//倍增初始化
    while(++i){//开始玄学生成
        x=(qMult(x,x,p)+c)%p;//可能要用快速乘
        z=qMult(z,Abs(y-x),p);//我们将每一次的 (y-x) 都累乘起来
        if(x==y||!z)break;//如果跑完了环就再换一组试试 (注意当 z=0 时, 继续下去是没意义的)
        if(!(i%127)||i==j){//我们不仅在等 127 次之后 gcd 我们还会倍增的来 gcd
            g=gcd(z,p);
            if(g>1)return g;
            if(i==j)y=x,j<=1;//维护倍增正确性, 并判环 (一箭双雕)
        }
    }
}

//fac 存因子,tot 0~tot-1
ll fac[100000],tot;
inline void findfac(ll p){//不断的找他的质因子
    if(p==1)return;
    if(prime(p)){fac[tot++]=p;return;}
    ll pi=Miller_Rabin(p);//我们一次必定能求的出一个因子, 所以不用 while
    while(p%pi==0)p/=pi;//记得要除尽
    return findfac(pi),findfac(p);//分开继续分解质因数
}

int main(){
    ll t,n;
    scanf("%lld",&t); srand(time(0));//随机数生成必备!!!
    while(t--){
        scanf("%lld",&n);
        tot=0;
        if(prime(n)){
            printf("Prime\n");continue;
        }
        findfac(n);
        sort(fac,fac+tot);
        printf("%lld\n",fac[tot-1]);
    }
    return 0;
}

```

1.16 高次同余

1.16.1 BSGS 算法

用于求 $a^x \equiv b \pmod{p}$ 高次方程的最小正整数解 x , 其中 p 为素数.

```
//求解  $A^x \equiv B \pmod{p}$   $A$  是底数,  $B$  是余数,  $p$  是质数,  $x$  是未知数
unordered_map<ll,int>mp;
ll BSGS(ll A,ll B,ll p){
    ll ans;
    mp.clear();
    ll m=ceil(sqrt(p));
    for(ll i=0,t=B;i<=m;i++,t=t*A%p)mp[t]=i;
    for(ll i=1,tt=binaryPow(A,m,p),t=tt;i<=m;i++,t=t*tt%p){
        if(mp.count(t)){
            ans=(i*m-mp[t]);
            return ans;
        }
    }
    return (ll)-1; //没有找到返回-1
}
```

1.17 组合数学

1.17.1 组合数

组合数计算公式:

$$\binom{n}{m} = \frac{A_n^m}{m!} = \frac{n!}{m!(n-m)!}$$

1.17.2 组合数的性质

- $\binom{n}{m} = \binom{n}{n-m}$ (对称性)
- $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$ (定义)
- $\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$ (递推式)
- $\binom{n}{k} = \frac{n-k+1}{k} \binom{n}{k-1}$
- $\sum_{l=0}^n \binom{l}{k} = \binom{n+1}{k+1}$ (斜向求和)
- $\sum_{i=0}^m \binom{n}{i} \binom{m}{m-i} = \binom{m+n}{m} (n \geq m)$
- $\binom{n}{i} \binom{i}{m} = \binom{n}{m} \binom{n-m}{i-m}$

1.17.3 组合数一些求和公式

- $C_n^0 + C_n^1 + C_n^2 + \dots + C_n^n = 2^n$

- $C_n^0 + C_n^2 + C_n^4 + \cdots = C_n^1 + C_n^3 + C_n^5 + \cdots = 2^{n-1}$
- $\sum_{i=m}^n \binom{a+i}{i} = \binom{a+n+1}{n} - \binom{a+m}{m-1}$

证明：

$$\binom{a+m}{m-1} + \binom{a+m}{m} + \binom{a+m+1}{m+1} + \cdots + \binom{a+n}{n} = \binom{a+n+1}{n}$$

1.17.4 朱世杰恒等式

$$\sum_{i=m}^n \binom{i}{a} = \binom{n+1}{a+1} - \binom{m}{a+1}$$

1.17.5 范德蒙恒等式

$$\sum_{i=0}^k \binom{n}{i} \binom{m}{k-i} = \binom{n+m}{k}$$

1.17.6 二阶求和公式

$$\sum_{r=0}^n \binom{n}{r}^2 = \binom{2n}{n}$$

1.17.7 李善兰恒等式

$$\binom{n+k}{k}^2 = \sum_{j=0}^k \binom{k}{j}^2 \binom{n+2k-j}{2k}$$

1.17.8 求组合数

公式 [\[编辑\]](#)

对于非负整数m和n和素数p，同余式：

$$\binom{m}{n} \equiv \prod_{i=0}^k \binom{m_i}{n_i} \pmod{p},$$

成立。其中：

$$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0,$$

并且

$$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$$

是m和n的p进制展开。当m < n时，二项式系数 $\binom{m}{n} = 0$ 。

结论 [\[编辑\]](#)

- 二项式系数 $\binom{m}{n}$ 可被素数 p 整除当且仅当在p进制表达下n的某一位的数值大于m

1.17.8.1 原理：卢卡斯定理 定义式子：

递推式: $\binom{sp+q}{tp+r} = \binom{s}{t} \binom{q}{r} (\text{mod } p)$, p 为素数

则有 $\binom{n}{m} \text{mod } p = \binom{n/p}{m/p} \binom{n \text{ mod } p}{m \text{ mod } p} \text{mod } p$

复杂度 $O(\log_p^n * p)$ 打表可降至 $O(\log_p n + p)$

1.17.8.2 计算组合数

1.17.8.2.1 计算单个组合数 大组合数求模, p 是小素数 ($1e5$) 时使用

```
//快速幂
ll binaryPow(ll a,ll b,ll m);
ll C(ll n,ll m){
    if(n<m) return 0;
    if(m>n-m) m=n-m;
    ll a=1,b=1;
    for(int i=0;i<m;i++){
        a=(a*(n-i))%p;
        b=(b*(i+1))%p;
    }
    return a*binaryPow(b,p-2,p)%p; //费马小定理求逆元
}
//算的时候的入口 计算  $C_n^m$ 
ll Lucas(ll n,ll m){
    if(m==0) return 1;
    return Lucas(n/p,m/p)*C(n%p,m%p)%p;
}
```

1.17.8.3 预处理阶乘逆元表. 使用定义式 $C(n, m) = \frac{n!}{m!*(n-m)!}$

如果不能接受多个 \log 乘积的复杂度时, 这个办法还可以, 缺点是需要 $O(n)$, 如果 n 不大时还可以。

```
//用  $O(n)$  的时间预处理逆元表  $inv[n]$ .
ll inv[Maxn+10];
void setInv(int n){
    inv[0]=inv[1]=1;
    for(int i=2;i<=n;i++){
        inv[i]=1LL*(mod-mod/i)*inv[mod%i]%mod;
    }
}
//预处理阶乘表  $fac[n]=(fac[n-1]*n)\%p=(n!)\%p$ 
//预处理阶乘的逆元表  $facInv[n]=(facInv[n-1]*inv[n])\%p$  .
ll facInv[Maxn+10], fac[Maxn+10];
void setFac(int n){
```



```

    fac[0]=facInv[0]=1;
    for(int i=1;i<=n;i++){
        fac[i]=1LL*fac[i-1]*i%mod;
        facInv[i]=1LL*facInv[i-1]*inv[i]%mod;
    }
}

//计算组合数  $C_n^m$ 
ll C(int n,int m){
    if(n<m) return 0;
    if(n<0||m<0) return 0;
    int ans=fac[n];
    ans=1LL*ans*facInv[m]%mod;
    ans=1LL*ans*facInv[n-m]%mod;
    return ans;
}

```

1.17.8.4 阶乘 + 快速幂逆元求组合数 通常用这种就够了，还是算好复杂度。按需取用。

```

//预处理阶乘表 fac[n]=(fac[n-1]*n)%p=(n!)%p
ll fac[Maxn+10];
void setFac(int n){
    fac[0]=1;
    for(int i=1;i<=n;i++){
        fac[i]=1LL*fac[i-1]*i%mod;
    }
}

ll binaryPow(ll a,ll b,ll m){
    ll ans = 1;
    while(b){
        if(b & 1){
            ans = ans * a % m;
        }
        a = a * a % m;
        b >>= 1;
    }
    return ans;
}

//计算组合数  $C_n^m$ 
ll C(int n,int m){
    if(n<m) return 0;
    if(n<0||m<0) return 0;
    ll t=fac[n-m]*fac[m]%mod;

```

```

    ll inv=binaryPow(t,mod-2,mod);
    return fac[n]*inv%mod;
}

```

1.17.9 抽屉原理

把 $n+1$ 个物体放进 n 个盒子, 至少有一个盒子包含两个或更多盒子.

1.17.10 容斥原理

要计算几个集合并集的大小, 我们要先将所有单个集合的大小计算出来, 然后减去所有两个集合相交的部分, 再加回所有三个集合相交的部分, 再减去所有四个集合相交的部分, 依此类推, 一直计算到所有集合相交的部分.

1.17.10.1 莫比乌斯函数

求每个数的莫比乌斯函数, 具体《算法竞赛进阶指南》

```

for(int i=1;i<=n;i++) miu[i]=1,v[i]=0;
for(int i=2;i<=n;i++){
    if(v[i])continue;
    miu[i]=-1;
    for(int j=2*i;j<=n;j+=i){
        v[j]=1;
        if((j/i)%i==0)miu[j]=0;
        else miu[j]*=-1;
    }
}

```

```

const int MAXN=5e4+10;//质数表
int primes[MAXN+10],NUM=0;
bool is_prime[MAXN+10];
void GetPrime2(){
    memset(is_prime, true, sizeof(is_prime));
    is_prime[1] = 0;
    for (int i = 2; i <= MAXN; i++) {
        if (is_prime[i])
            primes[NUM++] = i;
        for (int j = 0; j < NUM && i * primes[j] <= MAXN; j++) {
            is_prime[i * primes[j]] = 0;
            if (i % primes[j] == 0)
                break;
        }
    }
}

```

```

    }
}
int fac[Maxn]; //分解
int getfac(int x){
    int num=0;
    for(int i=0; i<NUM&&primes[i]<=x; i++){
        if(x%primes[i]==0){
            fac[num++]=primes[i];
            while(x%primes[i]==0){
                x/=primes[i];
            }
        }
    }
    if(x>1) fac[num++]=x;
    return num;
}
ll solve(int x, int n){ //求 [1, n] 中与 x 不互素的数的个数
    int num=getfac(x); //分解 x
    ll sum=0;
    for(ll i=1; i<(1<<num); i++){
        int mult=1, bits=0;
        for(int j=0; j<num; j++){
            if(i&(1<<j)){
                bits++; mult*=fac[j]; //二进制枚举
            }
        }
        ll cur=n/mult; //除得到个数
        //容斥定理 奇数个+, 偶数个-
        if(bits&1) sum+=cur;
        else sum-=cur;
    }
    return sum;
}
}

```

1.17.10.2 二进制枚举求 [1, n] 中与 x 不互素的数的个数, 反过来可求互质的数的个数

1.17.11 盒子装球问题

1.17.11.1 1. 球相同, 盒子不同, 不能有空盒 就是把 n 个球分成 m 份, 每一份不能为空, 插 m-1 个板即可.

$$ans = C_{n-1}^{m-1}$$

1.17.11.2 2. 球相同, 盒子不同, 可以有空盒 一份再拿走一个球即可.

把 n 个球分成 m 份, 每一份可以为空, 再增加 m 个球, 插 $m-1$ 个板, 每

$$ans = C_{n+m-1}^{m-1}$$

1.17.11.3 3. 球不同, 盒子不同, 可以有空盒 球与球之间是独立的.

对于每一个球, 你都可以放到 $[1, m]$ 的任意一个位置, 由于球不同, 所以

$$ans = m^n$$

1.17.11.4 4. 球不同, 盒子相同, 不能有空盒

相当于把 n 个元素的集合划分成 m 份, 也就是第二类斯特林数.

$$ans = S_n^m$$

时间复杂度 $O(n^2)$

1.17.11.5 5. 球不同, 盒子不同, 不能有空盒

公式表示是

$$ans = m! * S_n^m$$

1.17.11.6 6. 球不同, 盒子相同, 可以有空盒 第二类斯特林数加起来就可以了.

因为可以有空盒, 我们可以枚举每次一共用了几个盒子, 然后把相应的

$$ans = \sum_{i=0}^m S[n][i]$$

也叫 Bell 数: 第 n 个 Bell 数表示集合 $[1, 2, 3, \dots, n]$ 的划分方案数

$$B_n = \sum_{m=1}^n S[n][m]$$

1.17.11.7 7. 球相同, 盒子相同, 可以有空盒

设 $f[n][m]$ 表示 n 个球放到 m 个盒子里的方案数

$$if(n == 0 || m == 1) f[n][m] = 1 \quad if(n < m) f[n][m] = f[n][n]$$

$$if(n >= m) f[n][m] = f[n-m][m] + f[n][m-1]$$

如果球比盒子多, 分为放满和不放满两种情况讨论等价于自然数拆分问题

1.17.11.8 8. 球相同, 盒子相同, 不能有空盒

我们首先在所有的盒子中放一个球, 就转化成了问题 7

$$ans = f[n-m][m]$$

1.18 康托展开

1.18.1 正向康托展开

```

//返回是这个排列从小到大第几大的数，从 0 开始
ll factor[100]; //要预处理 n 的阶乘
ll Cantor(int a[], int n){
    ll ans=0, count;
    for(int i=0; i<n; i++){
        count=0;
        for(int j=i+1; j<n; j++){
            if(a[i]>a[j]){
                count++; //当前未出现的元素排在第几个
            }
        }
        ans+=count*factor[n-i-1];
    }
    return ans;
}

```

1.18.2 逆向康托展开

```

ll factor[100]={1,1,2,6,24,120,720,5040,40320,392880}; //预处理阶乘
//返回从 0 开始的第几大的序列在 a 中 , num: 第几大, n 长度, a 返回结果
void DeCantor(int num, int *a, int n=9){
    int v[10]; memset(v, 0, sizeof(v));
    int cnt, j, less;
    for(int i=0; i<n; i++){
        less=num/factor[n-i-1];
        for(j=0; j<n; j++){
            if(!v[j]){
                if(!less) break;
                less--;
            }
        }
        a[i]=j; num%=factor[n-i-1]; v[j]=1;
    }
}

```

1.18.3 错排数

错位排列 $D[i]$ 表示 i 个数都不在原来的位置上的排列的个数。

```

D[0] = 1;
D[1] = 0;
for(int i = 2; i <= 1000000; i++) {
    if(i & 1) {
        D[i] = ((11)i * D[i - 1] - 111) % MOD;
        if(D[i] < 0)
            D[i] += MOD;
    } else D[i] = ((11)i * D[i - 1] + 111) % MOD;
}

```

1.19 母函数

1.19.1 普通型母函数求组合方案数

```

/* 求多项式展开系数 (1+x+x^2+...)(1+x^2+x^4+...)(1+x^3+x^6+...)...
* 相当于手动改展开过程
*c1[n] 项用于记录每次展开后 x^n 项的系数，计算结束后 c1[n] 就是整数 n 的划分数
*c2[] 用于记录临时结果
*/
const int MAXN=200;
int c1[MAXN+1],c2[MAXN+1];
void part(){
    int i,j,k;
    for(i=0;i<=MAXN;i++){//初始化，即第一部分 (1+x+x^2+...)，都是 1
        c1[i]=1;c2[i]=0;
    }
    for(k=2;k<=MAXN;k++){//从第二部分 (1+x^2+x^4+...) 开始展开
        for(i=0;i<=MAXN;i++){
            //k=2 时,i 循环第 1 部分 (1+x+x^2+...),j 循环第二部分 (1+x^2+x^4+...)
            for(j=0;j+i<=MAXN;j+=k){
                c2[i+j]+=c1[i];
            }
        }
        for(i=0;i<=MAXN;i++){//更新本次展开结果
            c1[i]=c2[i];c2[i]=0;
        }
    }
}

```

1.19.2 指数型母函数求排列数

例 2.5 有 1,2,3,4 四个数字组成的五位数中,要求数 1 出现次数不超过 2 次,但不能不出现;2 出现次数不超过 1 次;3 出现次数最多 3 次,可以不出现;4 出现次数为偶数,求满足上述条件的数的个数。

解 c_i 对应的指数型母函数为

$$G(x) = \left(\frac{x}{1!} + \frac{x^2}{2!} \right) (1+x) \left(1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} \right) \left(1 + \frac{x^2}{2!} + \frac{x^4}{4!} \right) =$$

$$x + \frac{5}{2}x^2 + 3x^3 + \frac{8}{3}x^4 + \frac{43}{24}x^5 + \frac{43}{48}x^6 + \frac{17}{48}x^7 + \frac{1}{288}x^8 + \frac{1}{48}x^9 + \frac{1}{288}x^{10} =$$

$$\frac{x}{1!} + 5 \frac{x^2}{2!} + 18 \frac{x^3}{3!} + 64 \frac{x^4}{4!} + 215 \frac{x^5}{5!} + 645 \frac{x^6}{6!} + 1785 \frac{x^7}{7!} + 140 \frac{x^8}{8!} +$$

$$7650 \frac{x^9}{9!} + 12600 \frac{x^{10}}{10!}$$

由此可见,满足条件的 5 位数共 215 个。

例子:

上面那个只是例子不代表下面那个代码, 其他类似构造求解板子:

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <iostream>

using namespace std;

typedef long long ll;

const int maxn = 1e5 + 10;
double a[maxn], b[maxn]; // 注意为浮点型

int siz[maxn];

double f[11];
void init() {
    f[0] = 1;
    for (int i = 1; i <= 10; i++) {
        f[i] = f[i - 1] * i;
```

```

    }
}
int main() {
    int n,m;
    init();
    while (~scanf("%d%d", &n, &m)) { //从 n 个物品中选 m 个进行排列
        memset(a, 0, sizeof a);
        memset(b, 0, sizeof(b));
        memset(siz, 0, sizeof siz);
        for (int i = 0; i < n; i++) {
            scanf("%d", &siz[i]); //记录每种物品的多少
        }
        for (int i = 0; i <= siz[0]; i++) a[i] = 1.0 / f[i]; //初始化第一项
        for (int i = 1; i < n; i++) {
            memset(b, 0, sizeof(b)); //初始化 b 临时数组
            for (int j = 0; j <= m; j++) {
                for (int k = 0; k <= siz[i] && k + j <= m; k++) { //合并到 min(siz, m) 就行了
                    b[j + k] += a[j] * 1.0 / f[k]; //注意这里
                }
            }
            memcpy(a, b, sizeof b); //合并完
        }
        printf("%.0f\n", a[m] * f[m]); //a[m] 数组记录的  $x^m$  系数; 相当于通分整理
    }
    return 0;
}

```

1.20 特殊计数

1.20.1 Catalan 数

Catalan 数数列定义如下:

$$C_n = \frac{1}{n+1} \binom{2n}{n}, n = 0, 1, 2, \dots$$

前 20 项的 Catalan 数为: 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, ... 增长极快.

1.20.1.1 模型 1 $C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \binom{2n}{n} - \binom{2n}{n-1}$

可用模型 1 解释: 1. 把 n 个 1 和 n 个 0 排成一行, 使这一行的任意 k 个数中 1 的数量总是大于或等于 0 的数量 (或者相反 [等价]), 这样的排列有多少个?

2. 棋盘问题: 在 $n * n$ 的方格地图中, 从一个角到另外一个角, 求不跨越对角线的路径数有多少种.

3. 括号问题: 用 n 个左括号和 n 个右括号组成一串字符串有多少种合法的组合? 不匹配是非法的.
4. 出栈序列问题: 一个栈的进栈序列为 $1, 2, 3, \dots, n$, 求不同的出栈序列有多少种.
5. 买票找零问题: 有 $2n$ 个人排成一行进入剧场. 入场费 5 元. 其中只有 n 个人有一张 5 元钞票, 另外 n 人只有 10 元钞票, 剧院无其它钞票, 问有多少中方法使得只要有 10 元的人买票, 售票处就有 5 元的钞票找零?

1.20.1.2 模型 2

$$C_n = C_0 C_{n-1} + C_1 C_{n-2} + \dots + C_{n-2} C_1 + C_{n-1} C_0 = \sum_{k=0}^{n-1} C_k C_{n-k-1}, C_0 = 1$$

1. 二叉树问题 n 个结点构成的二叉树共有多少种情况 (或者有 $2n+1$ 个节点的满二叉树, 本质一样)
2. 三角剖分问题: 把一个有 $n+2$ 条边的凸多边形划分成多个三角形有多少种方法?

以上答案都是 C_n

1.20.1.3 Catalan 数的计算 $1. C_n = C_0 C_{n-1} + C_1 C_{n-2} + \dots + C_{n-2} C_1 + C_{n-1} C_0 = \sum_{k=0}^{n-1} C_k C_{n-k-1}, C_0 = 1$ (适用于 n 较小, $n \leq 100$, 优点是不用算逆元处理)

$$2. C_n = \frac{4n-2}{n+1} C_{n-1}, C_0 = 1$$

$$3. C_n = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!(n!)}$$

$$4. C_n = \binom{2n}{n} - \binom{2n}{n-1} \text{ (还是比较好的)}$$

根据不同的范围, 选择方便的计算, 更大的可能需要高精计算

1.21 Stirling 数

1.21.1 第一类 Stirling 数

定义: $s(n, k)$: 把 n 个不同的元素分配到 k 个圆排列里, 圆不能为空, 问有多少种分法?

1.21.1.1 递推公式: $s(n, k) = s(n-1, k-1) + (n-1)s(n-1, k), 1 \leq k \leq n$

$$s(0, 0) = 1, s(k, 0) = 0, s(n, n) = 1, 1 \leq k \leq n$$

部分 Stirling

$s(n, k)$	0	1	2	3	4	5	6	...
0	1							
1	0	1						
2	0	1	1					
3	0	2	3	1				
4	0	6	11	6	1			
5	0	24	50	35	10	1		

s(n,k)	0	1	2	3	4	5	6	...
6	0	120	274	225	85	15	1	
...								

```

11 s[Maxn+10][Maxn+10];
11 C[Maxn+10][Maxn+10];
void init(){
    for(int i=0;i<=Maxn;i++){
        C[i][0]=C[i][i]=s[i][i]=1;
        for(int j=1;j<i;j++){
            C[i][j]=(C[i-1][j]+C[i-1][j-1])%mod;
            s[i][j]=(s[i-1][j-1]+(i-1)*s[i-1][j]%mod)%mod;
        }
    }
}

```

1.21.1.2 组合数 +striling 数打表

1.22 第二类 Stirling 数

定义 $S(n, k)$: 把 n 个不同的球分配到 k 个相同的盒子里, 不能有空盒. 问有多少种分法?

$S(n, k)$ 的递推公式如下:

$$S(n, k) = kS(n-1, k) + S(n-1, k-1), 1 \leq k \leq n$$

$$S(0, 0) = 1, S(i, 0) = 0, S(i, i) = 1, 1 \leq i \leq n$$

第二类 Stirling 数 $S(n, k)$ 的值

s(n,k)	0	1	2	3	4	5	6	...
0	1							
1	0	1						
2	0	1	1					
3	0	1	3	1				
4	0	1	7	6	1			
5	0	1	15	25	10	1		
6	0	1	31	90	65	15	1	
...								

1.23 概率和数学期望

设有随机变量 X , 出现取值 x_i 的概率是 p_i , 把它们的乘积之和称为数学期望.

$$E(X) = \sum_{i=1}^n x_i p_i \text{ 线性性质 (DP): } E(X + Y) = E(X) + E(Y)$$

1.23.1 期望 dp

总的来说, 求概率的需要从前往后推, 求期望的需要从后往前推

$$E = \sum p_1 * (E_1 + X_1) + \sum p_2 * (E + X_2)$$

其中 E 为当前状态的期望, E_1 为下一个状态的期望, p_1 和 X_1 分别为将当前状态转移到下一个状态的概率和花费, p_2 和 X_2 分别为保持当前状态的概率和花费.

1.24 高斯消元板子

1.24.1 整数版本

```
#include<stdio.h>
#include<algorithm>
#include<iostream>
#include<string.h>
#include<math.h>
using namespace std;
const int MAXN=50;
int a[MAXN][MAXN]; //增广矩阵
int x[MAXN]; //解集
bool free_x[MAXN]; //标记是否是不确定的变元
inline int gcd(int a,int b){
    int t;
    while(b!=0){
        t=b;
        b=a%b;
        a=t;
    }
    return a;
}
inline int lcm(int a,int b){
    return a/gcd(a,b)*b; //先除后乘防溢出
}
```

// 高斯消元法解方程组 (Gauss-Jordan elimination). (-2 表示有浮点数解, 但无整数解,

```

// -1 表示无解, 0 表示唯一解, 大于 0 表示无穷解, 并返回自由变元的个数)
// 有 equ 个方程, var 个变元。增广矩阵行数为 equ, 分别为 0 到 equ-1, 列数为 var+1, 分别为 0 到 var.
int Gauss(int equ,int var)
{
    int k;
    int max_r;// 当前这列绝对值最大的行.
    int col;//当前处理的列
    int free_x_num;
    int free_index;

    for(int i=0; i<=var; i++){
        x[i]=0;
        free_x[i]=true;
    }

    //转换为阶梯阵.
    col=0; // 当前处理的列
    for(k = 0; k < equ && col < var; k++,col++){
        // 枚举当前处理的行.
        // 找到该 col 列元素绝对值最大的那行与第 k 行交换.(为了在除法时减小误差)
        max_r=k;
        for(int i=k+1; i<equ; i++)
        {
            if(abs(a[i][col])>abs(a[max_r][col])) max_r=i;
        }
        if(max_r!=k)
        {
            // 与第 k 行交换.
            for(int j=k; j<var+1; j++) swap(a[k][j],a[max_r][j]);
        }
        if(a[k][col]==0)
        {
            // 说明该 col 列第 k 行以下全是 0 了, 则处理当前行的下一列.
            k--;
            continue;
        }
        for(int i=k+1; i<equ; i++)
        {
            // 枚举要删去的行.
            if(a[i][col]!=0)
            {
                int LCM = lcm(abs(a[i][col]),abs(a[k][col]));

```

```

    int ta = LCM/abs(a[i][col]);
    int tb = LCM/abs(a[k][col]);
    if(a[i][col]*a[k][col]<0)tb=-tb;//异号的情况是相加
    for(int j=col; j<var+1; j++)
    {
        a[i][j] = a[i][j]*ta-a[k][j]*tb;
    }
}
}
}

```

// 1. 无解的情况：化简的增广阵中存在 $(0, 0, \dots, a)$ 这样的行 ($a \neq 0$) .

```
for (int i = k; i < equ; i++)
```

```
{
```

// 对于无穷解来说，如果要判断哪些是自由变元，那么初等行变换中的交换就会影响，则要记录交换 .

```
if (a[i][col] != 0) return -1;
```

```
}
```

// 2. 无穷解的情况：在 $var * (var + 1)$ 的增广阵中出现 $(0, 0, \dots, 0)$ 这样的行，即说明没有形成严格的上三角阵 .

// 且出现的行数即为自由变元的个数 .

```
if (k < var)
```

```
{
```

// 首先，自由变元有 $var - k$ 个，即不确定的变元至少有 $var - k$ 个 .

```
for (int i = k - 1; i >= 0; i--){
```

// 第 i 行一定不会是 $(0, 0, \dots, 0)$ 的情况，因为这样的行是在第 k 行到第 equ 行 .

// 同样，第 i 行一定不会是 $(0, 0, \dots, a)$ ， $a \neq 0$ 的情况，这样的无解的 .

$free_x_num = 0$; // 用于判断该行中的不确定的变元的个数，如果超过 1 个，则无法求解，它们仍然为不确定的变元 .

```
for (int j = 0; j < var; j++){
```

```
    if (a[i][j] != 0 && free_x[j]) free_x_num++, free_index = j;
```

```
}
```

if ($free_x_num > 1$) continue; // 无法求解出确定的变元 .

// 说明就只有一个不确定的变元 $free_index$ ，那么可以求解出该变元，且该变元是确定的 .

```
int temp = a[i][var];
```

```
for (int j = 0; j < var; j++){
```

```
    if (a[i][j] != 0 && j != free_index) temp -= a[i][j] * x[j];
```

```
}
```

```
x[free_index] = temp / a[i][free_index]; // 出该变元 .
```

```
free_x[free_index] = 0; // 该变元是确定的 .
```

```
}
```

```
return var - k; // 自由变元有  $var - k$  个 .
```

```
}
```

```

// 3. 唯一解的情况：在  $var * (var + 1)$  的增广阵中形成严格的上三角阵。
// 计算出  $x_{n-1}, x_{n-2} \dots x_0$ .
for (int i = var - 1; i >= 0; i--){
    int temp = a[i][var];
    for (int j = i + 1; j < var; j++){
        if (a[i][j] != 0) temp -= a[i][j] * x[j];
    }
    if (temp % a[i][i] != 0) return -2; // 说明有浮点数解，但无整数解。
    x[i] = temp / a[i][i];
}
return 0;
}

int main(void){
    int i, j;
    int equ, var;
    while (scanf("%d %d", &equ, &var) != EOF){
        memset(a, 0, sizeof(a));
        for (i = 0; i < equ; i++){
            for (j = 0; j < var + 1; j++){
                scanf("%d", &a[i][j]);
            }
        }
        int free_num = Gauss(equ, var);
        if (free_num == -1) printf(" 无解!\n");
        else if (free_num == -2) printf(" 有浮点数解，无整数解!\n");
        else if (free_num > 0){
            printf(" 无穷多解！自由变元个数为%d\n", free_num);
            for (i = 0; i < var; i++){
                if (free_x[i]) printf("x%d 是不确定的\n", i + 1);
                else printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        else{
            for (i = 0; i < var; i++){
                printf("x%d: %d\n", i + 1, x[i]);
            }
        }
        printf("\n");
    }
    return 0;
}

```

```
}
```

1.24.2 解线性同余方程组，取模版本

```
int MOD = 7;
const int MAXN = 75;
int a[MAXN][MAXN]; // 增广矩阵
int x[MAXN]; // 解集
bool free_x[MAXN]; // 标记是否是不确定的变元
long long inv(long long a, long long m){
    if(a == 1) return 1;
    return inv(m%a, m)*(m-m/a)%m;
}
// 高斯消元法解方程组。(-2 表示有浮点数解，但无整数解，-1 表示无解，
// 0 表示唯一解，大于 0 表示无穷解，并返回自由变元的个数)
// 有 equ 个方程，var 个变元。增广矩阵行数为 equ，分别为 0 到 equ-1，列数为 var+1，分别为 0 到 var
int Gauss(int equ, int var)
{
    for(int i = 0; i <= var; i++){
        x[i] = 0;
        free_x[i] = true;
    }
    int k;
    int max_r; // 当前这列绝对值最大的行
    int free_x_num;
    int free_index;

    // 转换为阶梯阵
    int col = 0; // 处理当前的列
    for(k = 0; k < equ && col < var; k++, col++){
        { // 枚举当前处理的行，找到该 col 列元素绝对值最大的那行与第 k 行交换。(为了在除法时减小误差)
            int max_r = k;
            for(int i = k+1; i < equ; i++){
                {
                    if((a[i][col]) > (a[max_r][col])) max_r = i;
                }
            }
            if(max_r != k)
                { // 与第 k 行交换
                    for(int j = k; j < var+1; j++) swap(a[k][j], a[max_r][j]);
                }
            if(a[k][col] == 0)

```

```

    { //说明该 col 列第 k 行一下全是 0 了, 则处理当前行的下一列
        k--;
        continue;
    }
    for(int i = k+1; i < equ; i++)
    { //枚举要删去的行
        ll now = inv(a[k][col], MOD) * a[i][col] % MOD;
        if(a[i][col] != 0)
        {
            for(int j = col; j < var+1; j++)
            {
                a[i][j] = ((a[i][j] - now * a[k][j]) % MOD + MOD) % MOD;
            }
        }
    }
}

//3. 唯一解的情况: 在 var*(var+1) 的增广阵中形成严格的上三角阵
//计算出  $x_{n-1}, x_{n-2}, \dots, x_0$ 
for(int i = var-1; i >= 0; i--)
{
    ll temp = a[i][var];
    for(int j = i+1; j < var; j++)
    {
        if(a[i][j] != 0) temp -= a[i][j] * x[j];
        temp = (temp % MOD + MOD) % MOD;
    }
    //if(temp % a[i][i] != 0) return -2;
    x[i] = (temp * inv(a[i][i], MOD)) % MOD;
}
return 0;
}

```

1.24.3 求解异或方程组的版

```

int a[Maxn][Maxn]; //增广矩阵
int x[Maxn]; //答案求的解集
int free_x[Maxn] //标记是否为不顶元
int equ, var; //eqy 方程个数, var 变量个数
/*
    equ 从 0 开始, var 也是从 0
    返回自由元个数, -1 表示无解, 0 表示唯一解

```



```

*/
int Gauss(){
    int max_r;
    int col=0,num = 0;
    int k;
    for(int i = 0;i<=var;++i) x[i] = free_x[i] = 0;
    for(k = 0;k < equ && col < var;k++,col++){
        max_r=k;
        for(int i=k+1;i<equ;i++){
            if(abs(a[i][col])>abs(a[max_r][col])) max_r=i;
        }
        if(max_r!=k){
            for(int j=k ;j<var+1;j++) swap(a[k][j],a[max_r][j]);
        }
        if(a[k][col]==0){
            free_x[num++] = col;
            k--; continue;
        }
        for(int i=k+1;i<equ;i++){
            if(a[i][col]!=0){
                for(int j=col;j<var+1;j++){
                    a[i][j]^=a[k][j];
                }
            }
        }
    }
    for(int i = k;i<equ;++i){
        if(a[i][col] != 0) return -1;
    }
    if(k < var) return var - k;
    for(int i = var - 1; i >= 0; i--){
        x[i]=a[i][var];
        for(int j = i + 1; j < var; j++){
            x[i] ^= ( a[i][j] && x[j]);
        }
    }
    return 0;
}

//枚举自由元. n 自由元个数,ans 最小的 1 的个数的解
void enum_freex(int n,int & ans){
    int num = (1<<(n));

```

```

ans = 1e9+7;
for(int i = 0; i < num; ++i){
    int cnt = 0;
    for(int j = 0; j < n; ++j){
        if(i & (1 < j)){
            cnt++;
            x[free_x[j]] = 1;
        } else x[free_x[j]] = 0;
    }
    for(int k = var - n - 1; k >= 0; --k){ // 没有自由元的最下面一行
        int index = 0;
        for(index = k; index < var; index++){ // 在当前行找到第一个非 0 自由元 (如果存在的话)
            if(a[k][index]) break;
        }
        x[index] = a[k][var];
        for(int j = index + 1; j < var; ++j){ // 向后依次计算出结果
            if(a[k][j]) x[index] ^= x[j];
        }
        cnt += x[index]; // 如果结果为 1, 则统计
    }
    // 枚举所有自由元的情况, 找到含 1 最小的解
    ans = min(ans, cnt);
}
}

```

1.24.4 浮点类型的版本

```

const double eps = 1e-7;
int equ, v;
double a[Maxn][Maxn], x[Maxn]; // a 增广矩阵, x 解集
bool free_x[Maxn]; // 标记是否是不确定的, 1 为是不确定的, 0 为确定
/*
无解返回 -1 唯一解返回 0, 多解返回自由元的个数
equ 从 0 开始, var 从 0 开始
*/
int Gauss(){
    int col = 0, k = 0; // col 为列号, k 为行号
    int free_x_num, free_x_index;
    double temp;
    for(int i = 0; i < var; i++){
        x[i] = 0; free_x[i] = true;
    }
}

```

```

}
for (;k<equ&&col<var;++k,++col){
    int r = k;
    for (int i=k+1;i<equ;++i)
        if(fabs(a[i][col])>fabs(a[r][col]))r=i;
    if (fabs(a[r][col])<eps){k--;continue;}//列全为 0
    if (r!=k)for(int i=col;i<=var;++i)
        swap(a[k][i],a[r][i]);
    for (int i=k+1;i<equ;++i){
        //消元
        if(fabs(a[i][col])>eps){
            double t = a[i][col]/a[k][col];
            for (int j=col;j<=var;j++)a[i][j]-=a[k][j]*t;
            a[i][col] = 0;
        }
    }
}
}
for(int i=k ;i<equ ;++i)//无解
    if (fabs(a[i][var])>eps) return -1;
if (k < var){
    for(int i=k-1;i>=0;i--){
        free_x_numm=0;
        for(int j=0;j<var;j++){
            if(fabs(a[i][j])>eps&&free_x[j]){
                free_x_numm++,free_x_index=j;
            }
        }
        if(free_x_numm>1)continue;//多余一个无法求出确定变元
        temp=a[i][var];
        for(int j=0;j<var;j++){
            if(fabs(a[i][j])>eps&&j!=free_x_index){
                temp-=a[i][j]*x[j];
            }
        }
        x[free_x_index]=temp/a[i][free_x_index];

        free_x[free_x_index]=0;
    }
    return var - k; //返回自由元个数
}
for (int i =var-1; i>=0; i--){//回带求解
    double temp = a[i][var];

```

```

        for (int j=i+1; j<var; ++j)
            temp -= x[j] * a[i][j];
        x[i] = (temp / a[i][i]);
    }
    return 0;
}

int main()
{
    while(scanf("%d %d",&equ,&var)==2)
    {
        memset(a,0,sizeof(a));
        for(int i = 0;i < equ;i++)
        {
            for(int j = 0;j < var+1;j++)
            {
                scanf("%lf",&a[i][j]);
            }
        }

        int free_num = Gauss();
        if(free_num == -1) printf("No solution\n");
        else if(free_num == -2) printf("Float but no int solution\n");
        else if(free_num > 0)
        {
            printf("Infinite solution, 自由变元个数为%d\n",free_num);
            for(int i = 0;i < var;i++)
            {
                if(free_x[i]) printf("x%d 是不确定的\n",i+1);
                else printf("x%d: %.2lf\n",i+1,x[i]);
            }
        }
        else
        {
            for(int i = 0;i < var;i++)
            {
                printf("x%d: %.2lf\n",i+1,x[i]);
            }
        }
        printf("\n");
    }
    return 0;
}

```

1.25 异或空间

1.25.1 线性基的性质：

1. 原集合中的任何一个数都可以由线性基中的一些数异或得到
2. 线性基中任意一些数异或都不能得到 0
3. 也就是基向量的个数唯一.
4. 一个集合可以有多个大小相同但基向量不完全相同的线性基.

1.25.2 求异或空间的基

虽然高斯消元也可以做, 但是高斯消元是 $n*n$ 的复杂度. 一种很快的求线性基的方法:

```
// p 相当于行阶梯矩阵, 行号是每一位的情况, 有可能某个 p[i] 为 0
// 插入操作
void add(ll x)
{
    for(int i = maxBit - 1; i >= 0; -- i )// maxBit 为有多少位数
    {
        if(x >> i & 1)
        {
            if(!p[i]) { p[i] = x; break; }//p[i] 为行阶梯矩阵
            x ^= p[i];
        }
    }
}

// 可以继续得到行最简矩阵, p[i] 大小不一定是递增顺序, 可能超时
for(int i = maxBit - 1; i >= 0; -- i ){
    for(int j = i - 1; j >= 0; -- j){
        if(p[i] >> j & 1)
            p[i] ^= p[j];
    }
}
```

1.25.3 根据线性基, 求能异或出最大的数

贪心, 阶梯矩阵如果异或那一位发现变小, 就没有必要异或了。

```
ll work(){
    ll ans=0;
    for(int i=60;i>=0;i--)//记得从线性基的最高位开始
```

```

    if((ans^p[i])>ans)ans^=p[i];
    return ans;
}

```

1.25.4 [HDU 3949] XOR 【线性基 __XOR 第 k 小】整数集中能异或出的求第 k 小的数

```

#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;
typedef long long ll;

const int maxN = 55;
const int maxBit = 63;

ll n, a, p[maxBit];
ll cnt, Linear[maxBit];

void add(ll x)
{
    for(int i = maxBit - 1; i >= 0; -- i )
    {
        if(x >> i & 1)
        {
            if(!p[i]) { p[i] = x; break; }
            x ^= p[i];
        }
    }
}

void op()
{
    for(int i = maxBit - 1; i >= 0; -- i )
        for(int j = i - 1; j >= 0; -- j )
            if(p[i] >> j & 1)
                p[i] ^= p[j];
    cnt = 0;
    for(int i = 0; i < maxBit; ++ i )
        if(p[i]) Linear[cnt++] = p[i]; // 相当于紧凑，排序
}

```

```

int main()
{
    int cas = 0;
    int TAT; cin >> TAT;
    while(TAT -- )
    {
        memset(p, 0, sizeof(p));
        scanf("%lld",&n);
        for(int i = 0; i < n; ++ i )
            scanf("%lld",&a), add(a);
        op();
        printf("Case #%d:\n", ++cas);
        int QAA; cin >> QAA;
        while(QAA -- )
        {
            ll k;scanf("%lld",&k);
            if(cnt != n) -- k;
            if(k >= 1ll << cnt)
            {
                printf("-1\n");
                continue;
            }
            ll ans = 0;
            for(int i = 0; i < cnt; ++ i )
                if(k >> i & 1) ans ^= Linear[i]; // 二进制选取
            printf("%lld\n", ans);
        }
    }
    return 0;
}

```

1.25.5 前缀线性基

1.26 八数码问题有解判断

给定 nm 的矩阵, 判断两个局面是否有解: 当 m 为奇数时, 逆序数个数的奇偶性相同.

1.27 多项式乘法 FFT

最后得到两个多项式相乘系数

```

#include<algorithm>
#include<cstdio>
#include<cmath>
#define Maxn 1350000
using namespace std;
const double Pi=acos(-1);
struct CP
{
    CP (double xx=0,double yy=0){x=xx,y=yy;}
    double x,y;
    CP operator + (CP const &B) const
    {return CP(x+B.x,y+B.y);}
    CP operator - (CP const &B) const
    {return CP(x-B.x,y-B.y);}
    CP operator * (CP const &B) const
    {return CP(x*B.x-y*B.y,x*B.y+y*B.x);}
}f[Maxn<<1]; //只用了个复数数组
int tr[Maxn<<1];
void fft(CP *f,bool flag)
{
    for (int i=0;i<n;i++)
        if (i<tr[i])swap(f[i],f[tr[i]]);
    for(int p=2;p<=n;p<=1){
        int len=p>>1; //待合并的长度
        CP tG(cos(2*Pi/p),sin(2*Pi/p));
        if(!flag)tG.y*=-1; // p 次单位根
        for(int k=0;k<n;k+=p){
            CP buf(1,0); // 遍历一个子部分并合并
            for(int l=k;l<k+len;l++){
                CP tt=buf*f[l+len];
                f[l+len]=f[l]-tt;
                f[l]=f[l]+tt;
                buf=buf*tG;
            }
            /* 依据:
                
$$F(x)=FL(x^2)+x*FR(x^2)$$

                
$$F(W^k)=FL(W^k)+W^k*FR(W^k)$$

                
$$F(W^{k+n/2})=FL(W^k)-W^k*FR(W^k)$$

            */
        }
    }
}

```



```

int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    for (int i=0;i<=n;i++)f[i].x=read(); // 输入多项式 第  $x^i$  项的系数
    for (int i=0;i<=m;i++)f[i].y=read(); // 另一个多项式, 第  $x^i$  项的系数
    for(m+=n,n=1;n<=m;n<=<=1);
    for(int i=0;i<n;i++)
        tr[i]=(tr[i]>>1)>>1|((i&1)?n>>1:0); // 算出每个数的二进制翻转后的数.
/* " 三次变两次 " 优化
    根据  $(a+bi)*(c+di)=ac-bd+adi+bci$ 
    假设我们需要求  $F(x)*G(x)$ 
    设复多项式  $P(x)=F(x)+G(x)i$  也就是实部为  $F(x)$ , 虚部为  $G(x)$ .
    则  $P(x)^2=(F(x)+G(x)i)^2=F(x)^2-G(x)^2+2F(x)G(x)i$ 
    发现  $P(x)^2$  的虚部为  $2F(x)G(x)$ 
    也就是说求出  $P(x)^2$  之后, 把它的虚部除以 2 即可.
*/
/*
原本 3 次 dft 写法:
    for (int i=0;i<=n;i++)scanf("%lf",&f[i].x); //f 多项式系数
    for (int i=0;i<=m;i++)scanf("%lf",&p[i].x); // p 多项式系数
    for(m+=n,n=1;n<=m;n<=<=1); // m 一共有项数
    for(int i=0;i<n;i++)
        tr[i]=(tr[i]>>1)>>1|((i&1)?n>>1:0);
    fft(f,1);fft(p,1); //两个分别做 DFT
    for(int i=0;i<n;++i)f[i]=f[i]*p[i]; // 做乘法
    fft(f,0); // idft
    for(int i=0;i<=m;++i)printf("%d ",(int)(f[i].x/n+0.49)); //得到答案,f[i].y=0
*/
    fft(f,1); // 正向: dft
    for(int i=0;i<n;++i)f[i]=f[i]*f[i]; // f 乘
    fft(f,0); // 反向: idft
    for(int i=0;i<=m;++i)
        printf("%d ",(int)(f[i].y/n/2+0.49)); // 虚部除 2
    return 0;
}

```

1.27.1 FFT 计算 A+ 或-B 集合 C :

1.28 min25 筛求积性函数的前缀和

有时取模过多会超时

n 个数两个相减（相加）先序知识

Describe

给定 $n + m$ 个数：

$$A = \{a_1, a_2, a_3, a_4, a_5, \dots, a_n\}$$

$$B = \{b_1, b_2, b_3, b_4, b_5, \dots, b_m\}$$

求 $A \pm B$ 的集合 C

注意： $A + B$ 指的是 A 中的每一个元素与 B 中的每一个元素相加，即

$$C = \{c_{ij} \mid \forall i \in [1, n], \forall j \in [1, m]\}$$

相关题型：[Hash Function B-小圆前辈的素数](#)

图 7: fftex

Solve

朴素双重循环复杂度为 $O(N^2)$ ，对于数据量大于 10^5 的问题显然无法解决。

考虑 $\forall a_n, |a_n| \leq 10^5; \forall b_n, |b_n| \leq 10^5$ 我们将上述集合换一种表达式：

$$A(x) = x^{a_1} + x^{a_2} + x^{a_3} \dots x^{a_n}$$

$$B(x) = x^{b_1} + x^{b_2} + x^{b_3} \dots x^{b_m}$$

则对于 $C = A + B$ ，转化为 $C(x) = A(x) * B(x)$

$$C(x) = A(x) * B(x) = \sum_{\forall(i,j) \in [1,1] \times [n,m]} x^{a_i+b_j}$$

因此，我们只要将一个多项式中的 x^{a_n} 和 x^{b_m} 的系数赋值为 1，其他赋值为 0。进行一遍FFT后，检验 x^k 的系数，若 x^k 的系数大于等于 1，则 $k \in C$ 。

提示：对于减法，实际上是加上一个负数，因为负下标的问题，可以先统一加上一个偏移量，最后减去即可得到原值。

图 8: sovle

```
//Min-25 筛使用条件是：
//1.f(x) 在 x 属于质数的时候能够用多项式表示，如 f(p)=p^2+p. 为了能把其写成多个完全积性函数的若干倍的和/差
//2.f(x^k) 在 x 属于质数时要能够快速算出，否则，复杂度会变大
//代码计算的是函数 f(i) 的前 n 项和
#include<bits/stdc++.h>
#define LL long long
using namespace std;

const int N = 2e5 + 10; //2*sqrt(n) 的范围
const LL INV2 = 5e8 + 4; //2 的逆元（可能用到）
const LL INV6 = 166666668; //6 的逆元（可能用到）
const LL MOD = 1e9 + 7;

bool isnotp[N]; //memset 置 1 会拖慢速度
LL n,m,sz,sqrt_n,c0,c1,p[N],w[N],id0[N],id1[N],g0[N],g1[N],sum0[N],sum1[N];
//n 是输入的数，sqrt_n 保存 sqrt(n)，即预处理的数量
//sz 是质数个数，isp[i] 表示 i 是否质数，p[i] 存储第 i 个质数
//sum0[i] 存储的是前 i 个质数的 f0 值之和，sum1[i] 存储的是前 i 个质数的 f1 值之和
//m 是 n/k 的个数，w[i] 存储 n/k 第 i 种值（倒序），id0 和 id1[i] 存储 i 这个值在 w[i] 中的下标
//g0[i] 和 g1[i] 等分别存储 f 在取质数时的多项式中的不同次方项（此处只有两个数组，即假设题目中的 f 取质数时只
    有两项），g0[i] 存的是 g(w[i],0~sz)，g1[i] 存的是 g1(w[i],0~sz)
```

// $g_0(n, i) = \sum_{j=1}^n [j \text{ 是质数 or } j \text{ 的最小质因子 } > p[i]] * f_0(j)$ 其中 f_0 为 f 取质数时的第一个次方项
 // $g_1(n, i) = \sum_{j=1}^n [j \text{ 是质数 or } j \text{ 的最小质因子 } > p[i]] * f_1(j)$ 其中 f_1 为 f 取质数时的第二个次方项
 // c_0 和 c_1 等保存的是不同次方项的系数（此处只有两个系数，即假设题目中的 f 取质数时只有两项）

//计算 $f(p^t)$ ，若要降低常数也可把这个函数用增量法在调用处实现，比如可快速幂算出

```
inline LL f(LL p, LL t)
{
    //...
}
```

//线性筛，求函数 f_0 、 f_1 在前 i 个质数处的前缀和

```
void init(LL num)
{
    sz=0;
    memset(isnotp, 0, sizeof(isnotp));
    isnotp[1]=1;
    sum0[0]=0;
    sum1[0]=0;
    for (LL i=2; i<=num; i++)
    {
        if (!isnotp[i])
        {
            p[++sz]=i;
            //计算 sum0, 即  $sum_0(i) = \sum_{j=1}^i f_0(p[j])$  一个前缀和计算即可，注意取模
            //...
            //计算 sum1, 即  $sum_1(i) = \sum_{j=1}^i f_1(p[j])$ 
            //...
        }
        for (int j=1; j<=sz && p[j]*i<=num; j++)
        {
            isnotp[i*p[j]]=0;
            if (i%p[j]==0) break;
        }
    }
}

inline int get_id(LL x) {
    if(x<=sqrt_n) return id0[x];
    else return id1[n/x];
}
```

//计算原理中的多项式的项，只会计算 $g_0(n/i)$ ， $g_1(n/i)$

```

void sieve_g(LL num)
{
    m=0;
    for (LL i=1,j;i<=num;i=j+1)
    {
        LL k=num/i; j=num/k;
        w[++m]=k;
        if(k<=sqrt_n) id0[k]=m;
        else id1[num/k]=m;

        k%=MOD;
        //计算原理中的  $g_0(w[m], 0)$ , 即  $\sum_{j=2}^{w[m]} f_0(j)$ , 存在  $g_0[m]$  中, 注意可能要减去第一项, 下标
        ↪ 从 2 开始
        //...
        //计算原理中的  $g_1(w[m], 0)$ , 即  $\sum_{j=2}^{w[m]} f_1(j)$ , 存在  $g_1[m]$  中, 注意可能要减去第一项, 下标
        ↪ 从 2 开始
        //...
    }
    for (int i=1;i<=sz;i++)
    for (int j=1;j<=m&& p[i]*p[i]<=w[j];j++)
    {
        int op=get_id(w[j]/p[i]);
        // 计算出  $g(j, m)$ 
        //根据  $g_0[j] = (g_0[j] - f_0(p[i]) * ((g_0[op] - sum0[i-1] + MOD) \% MOD) \% MOD + MOD) \% MOD$  计算
        //...
        //根据  $g_1[j] = (g_1[j] - f_1(p[i]) * ((g_1[op] - sum1[i-1] + MOD) \% MOD) \% MOD + MOD) \% MOD$  计算
        //...
    }
}

/*  $S(x, y)$  小于等于  $x$ , 最小质因子大于等于第  $y$  个质数, 的所有数
* 的和.
*
*/
LL S(LL x, LL y)
{
    if (x<=1 || p[y]>x) return 0; //base case
    LL k=get_id(x), res=0;

    ↪ res=((c0*g0[k]%MOD+c1*g1[k]%MOD+MOD)%MOD-(c0*sum0[y-1]%MOD+c1*sum1[y-1]%MOD+MOD)%MOD+MOD)%MOD; //质
    ↪ 数部分的贡献
    //下面的二重循环统计的是合数部分的贡献

```

```

for(int i=y;i<=sz&& p[i]*p[i]<=x;i++)//枚举合数的最小质因子
{
    LL t0=p[i], t1=p[i]*p[i];
    for(LL e=1; t1<=x; t0=t1,t1*=p[i],e++)//枚举最小质因子的次数
    {
        LL fp0=f(p[i],e), fp1=f(p[i],e+1);
        (res+=(fp0*S(x/t0,i+1)%MOD+fp1)%MOD)%=MOD;
    }
}
return res;
}

int main()
{
    //freopen("test.in", "r", stdin);
    scanf("%lld",&n);
    sqrt_n=sqrt(n);
    init(sqrt_n); sieve_g(n);
    //此处对不同次项的系数 c0,c1 进行直接赋值
    //...
    //此处计算的是原函数 f 在取值为 1 时的函数值, 即 f(1), 存在 f_1 中; 若是积性函数的话一般有 f(1)=1
    long long f_1=1;
    //...
    printf("%lld\n",((S(n,1)+f_1)%MOD));
    return 0;
}

```

1.28.1 求 $1e10$ 以内的素数和模板:

ccpc 2020 网络赛 Graph Theory Class

构造 $f(x)=x$, 函数即可

```

//Min-25 筛使用条件是:
//1.  $f(x)$  在  $x$  属于质数的时候能够用多项式表示, 如  $f(p)=p^2+p$ . 为了能把其写成多个完全积性函数的若干倍的和/差
//2.  $f(x^k)$  在  $x$  属于质数时要能够快速算出, 否则, 复杂度会变大
//代码计算的是函数  $f(i)$  的前  $n$  项和
#include<bits/stdc++.h>

using namespace std;
typedef long long LL;
const int N = 2e5 + 10; //2*sqrt(n) 的范围
LL INV2 = 5e8 + 4; //2 的逆元 (可能用到)

```

```

LL INV6 = 166666668; //6 的逆元 (可能用到)
LL MOD;

bool isnotp[N];
LL n,m,sz,sqrt_n,c0,p[N],w[N],id0[N],id1[N],g0[N],sum0[N];
//n 是输入的数, sqrt_n 保存 sqrt(n), 即预处理的数量
//sz 是质数个数, isp[i] 表示 i 是否质数, p[i] 存储第 i 个质数
//sum0[i] 存储的是前 i 个质数的 f0 值之和, sum1[i] 存储的是前 i 个质数的 f1 值之和
//m 是 n/k 的个数, w[i] 存储 n/k 第 i 种值 (倒序), id0 和 id1[i] 存储 i 这个值在 w[i] 中的下标
//g0[i] 和 g1[i] 等分别存储 f 在取质数时的多项式中的不同次方项 (此处只有两个数组, 即假设题目中的 f 取质数时只
    ↪ 有两项), g0[i] 存的是 g(w[i], 0~sz), g1[i] 存的是 g1(w[i], 0~sz)
//g0(n, i) = \Sigma_{j=1}^n [j 是质数 or j 的最小质因子 > p[i]] * f0(j) 其中 f0 为 f 取质数时的第一个次方项
//g1(n, i) = \Sigma_{j=1}^n [j 是质数 or j 的最小质因子 > p[i]] * f1(j) 其中 f1 为 f 取质数时的第二个次方项
//c0 和 c1 等保存的是不同次方项的系数 (此处只有两个系数, 即假设题目中的 f 取质数时只有两项)

//计算 f(p~t), 若要降低常数也可把这个函数用增量法在调用处实现, 比如可快速幂算出
inline LL f(LL p, LL t)
{
    p%=MOD;
    LL res=1;
    while(t){
        if(t&1){
            res=res*p%MOD;
        }
        p=p*p%MOD;
        t>>=1;
    }
    return res;
}

inline void init(LL num)
{
    sz=0;
    isnotp[1]=1;
    sum0[0]=0;
    for (LL i=2; i<=num; i++)
    {
        if (!isnotp[i])
        {
            p[++sz]=i;
            //计算 sum0, 即 sum0(i) = \Sigma_{j=1}^i f0(p[j]) 一个前缀和计算即可
            sum0[sz]=sum0[sz-1]+i;
            //计算 sum1, 即 sum1(i) = \Sigma_{j=1}^i f1(p[j])

```

```

        //...
    }
    for (int j=1; j<=sz&& p[j]*i<=num; j++)
    {
        isnotp[i*p[j]]=1;
        if (i%p[j]==0) break;
    }
}

}

inline int get_id(LL x) {
    if(x<=sqrt_n) return id0[x];
    else return id1[n/x];
}

//计算原理中的多项式的项，只会计算  $g_0(n/i)$ ,  $g_1(n/i)$ 
inline void sieve_g(LL num)
{
    m=0;
    //    int cnt=0;
    for (LL i=1, j; i<=num; i=j+1)
    {
        //cnt++;
        LL k=num/i; j=num/k;
        w[++m]=k;
        if(k<=sqrt_n) id0[k]=m;
        else id1[num/k]=m;

        //k%=MOD;
        //计算原理中的  $g_0(w[m], 0)$ , 即  $\sum_{j=2}^{\infty} \{w[m]\} f_0(j)$ , 存在  $g_0[m]$  中, 注意可能要减去第一项, 下标
        ↪ 从 2 开始
        g0[m]=(1+k)*k/2-1;
        //计算原理中的  $g_1(w[m], 0)$ , 即  $\sum_{j=2}^{\infty} \{w[m]\} f_1(j)$ , 存在  $g_1[m]$  中, 注意可能要减去第一项, 下标
        ↪ 从 2 开始
        //...
    }
    //cout<<"### " << cnt << endl;
    for (int i=1; i<=sz; i++)
    for (int j=1; j<=m&& p[i]*p[i]<=w[j]; j++)
    {
        int op=get_id(w[j]/p[i]);
        // 计算出  $g(j, m)$ , 最小质因子  $\geq$  第  $j$  个质数, 或者质数为质数的和.
    }
}

```



```

        //根据  $g0[j]=(g0[j]-f0(p[i])*((g0[op]-sum0[i-1]+MOD)\%MOD)\%MOD+MOD)\%MOD$  计算
         $g0[j]=g0[j]-(p[i])*(g0[op]-sum0[i-1]);$ 
        //根据  $g1[j]=(g1[j]-f1(p[i])*((g1[op]-sum1[i-1]+MOD)\%MOD)\%MOD+MOD)\%MOD$  计算
        //...
    }
}

/*
 * 求素数和
 */
inline LL S(LL x,LL y)
{
    if (x<=1||p[y]>x) return 0;//base case
    LL k=get_id(x),res=0;
    res=c0*g0[k]%MOD;//质数部分的贡献
    return res;
}

int main()
{
    int t;
    scanf("%d",&t);
    //memset(isnotp,0,sizeof isnotp);
    while(t--){
        //cout<<"!"<<endl;
        scanf("%lld%lld",&n,&MOD);
        /*
         * 数据范围:  $n\leq 1e10, MOD\leq 1e9$ ;
         * 取模过多会 T
         */
        c0=1;
        INV2=f(2,MOD-2);
        LL ans=(n%MOD+3)*n%MOD*INV2%MOD;
        ans=(ans-4+MOD)%MOD;
        n=n+1;
        sqrt_n=sqrt(n);
        init(sqrt_n);sieve_g(n);
        ans=ans+S(n,1);//ans 加上  $(S(n,1):$  为所有素数的贡献)
        ans%=MOD;
        printf("%lld\n",ans);
        //此处对不同次项的系数 c0,c1 进行直接赋值
        //...
        //此处计算的是原函数  $f$  在取值为 1 时的函数值, 即  $f(1)$ , 存在  $f\_1$  中; 若是积性函数的话一般有  $f(1)=1$ 

```

```
//      long long  f_1=1;
//      //...
//      printf("%lld\n",((S(n,1)+f_1)%MOD));
//  }
//cout<<"!"<<endl;
return 0;
}
```

1.29 反素数

要求 1-n 中因子个数最多的数, 并且最小的数. 只能求 1-n 连续区间. 根据因子个数定理, 反素数的特点:

1. 反素数肯定是从 2 开始的连续素数的幂次形式的乘积.
2. 数值小的素数的幂次大于等于数值大的素数

1.29.1 求解代码,dfs, 在 n 的范围之内, 枚举质因子

```
#include<cstdio>
#include<iostream>
#define ULL unsigned long long

int p[16] = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53};
ULL n;
ULL ans,ans_num;//ans 为 n 以内的最大反素数 (会持续更新),ans_sum 为 ans 的因子数.

void dfs(int depth,ULL temp,ULL num,int up){ //depth 深度,temp 当前值,num 因子个数 up 上一个因子高度
    if(depth >= 16||temp > n)return;
    if(num > ans_num){ // 挑因子个数最多的数
        ans = temp;
        ans_num = num;
    }
    if(num == ans_num&&ans > temp){
        ans = temp;// 挑最小的数
    }
    for(int i = 1;i <= up;i++){
        if(temp*p[depth] > n)break;
        dfs(depth+1,temp *= p[depth],num*(i+1),i);
    }
    return;
}

int main(){
```

```

while(scanf("%lld",&n) != EOF){
    ans_num = 0;
    dfs(0,1,1,60); // 枚举到  $2^{60}$  次幂
    printf("%lld\n",ans);
}
return 0;
}

```

1.29.2 按照这种思路, 还可以求因子个数恰好等于某个数的最小的数.

1.30 约瑟夫环递推公式,

下标从 0 开始

$$f(N, M) = (f(N - 1, M) + M)$$

1.30.1 递推公式

```

int cir(int n,int m){
    int p=0;
    for(int i=2;i<=n;i++){
        p=(p+m)%i;
    }
    return p; // 下标从 0 开始
}

```

1.30.2 求解第 k 个出列的人

```

/*
 * 下标从 1 开始
 */
#include <iostream>
using namespace std;
int main(){
    int n,c,k;
    // n 个人
    cin>>n;
    // 报数报到 c
    cin>>c;
}

```

```

// 求第 k 个出去
cin>>k;
k=n+1-k;
int pos=(c-1)%k+1;
for (int i=k+1;i<=n;i++)
    pos=(pos+c-1)%i+1;
cout<<pos<<endl;
return 0;
}

```

1.31 POJ - 2886 每次指定往后走的步数或往前走步数的约瑟夫环, 求第 k 个

题目求的是反素数, 也就是第 k 个

```

#include<cstdio>
#include<vector>
#include<stack>
#include<queue>
#include<cstring>
#include<string>
#include<cmath>
#include<cstdlib>
#include<algorithm>
#define ll long long
const int maxn = 500000+5;
const int MOD = 1e7;
const int INF = 0x3f3f3f3f;
using namespace std;
struct node{
    char name[12];
    int num;
}e[maxn];
int get[maxn],sum[maxn << 2];
void pushup(int rt){
    sum[rt] = sum[rt << 1] + sum[rt << 1 | 1];
}
void build(int l,int r,int rt){
    if(l == r){
        sum[rt] = 1;
        return;
    }
    int m = (l + r) >> 1;

```

```

    build(l,m,rt << 1);
    build(m + 1,r,rt << 1 | 1);
    pushup(rt);
}

int update(int pos,int l,int r,int rt){
    if(l == r){
        sum[rt] = 0;
        return l;
    }
    int ans;
    int m = (l + r) >> 1;
    if(pos <= sum[rt << 1]) //pos 代表在剩余人数中的第几个，所以写法略有不同
        ans = update(pos,l,m,rt << 1);
    else
        ans = update(pos - sum[rt << 1],m + 1,r,rt << 1 | 1);
    pushup(rt);
    return ans;
}

void init(){
    int top = 500000;
    memset(get,0,sizeof(get));
    for(int i = 1;i <= top;i++){
        get[i]++;
        for(int j = 2*i;j <= top;j += i){
            get[j]++;
        }
    }
}

int main(){
    init();
    int n,k;
    while(~scanf("%d%d",&n,&k)){ // 输入一共人数，从第 k 个开始
        build(1,n,1);
        for(int i = 1;i <= n;i++){
            scanf("%s%d",e[i].name,&e[i].num);
            // 输入每个人的名字，那个人被杀后，
            // 下一轮往前或者往后的需要数多少个。
        }
        int aim = 0,MAX = -1;
        for(int i = 1;i <= n;i++){ // 这里得到反素数
            if(get[i] > MAX){
                aim = i;
                MAX = get[i];
            }
        }
    }
}

```

```

    }
}
int pos;    //开始计算的位置
int mov,all,now = 0;
while(true){
    pos = update(k,1,n,1);    //剩下的问题相当于动态求解前缀和等于 k 的从左往右的第一个下标是哪一个
    // 每次查询后把那个区间-1
    now++;
    if(now == aim) break; // 找到第 k 个
    mov = e[pos].num;
    if(mov > 0){    //在左边
        k = (k - 1 + mov - 1)%sum[1] + 1;    // 因为取模，下标从 0 开始，但线段树下标从 1 开始，
        ↪ 需要转换。
        //第二个 +1 是防止% 之后归 0
    }
    else{    //在右边
        k = ((k - 1 + mov)%sum[1] + sum[1])%sum[1] + 1;
    }
}
printf("%s %d\n",e[pos].name,MAX);
}
return 0;
}

```

1.32 加法与异或运算性质：

$$a + b = a|b + a \& b$$

1.33 求满足 $x + y + z \bmod n = 0$, 且 $0 \leq x < y < z < n$ 的三元组个数：

设 $f(n)$ 为满足 $x + y + z = 0 \pmod n$ 的三元组个数。

$$f(n) = \begin{cases} \frac{n^2-3n+6}{6} > \frac{n^2-3n}{6} & n \bmod 3 = 0 \\ \frac{n^2-3n+2}{6} > \frac{n^2-3n}{6} & n \bmod 3 \neq 0 \end{cases}$$

考虑 $f(n+3), f(n)$ 之间的转移, 将漏掉的加上。 $f(n+3) = f(n) + n$ 证明推导

1.34 数论分块

整数分块

以 $\sqrt{(n)}$ 的复杂度计算类似 $\sum_{i=1}^n \lfloor \frac{n}{i} \rfloor$

因为中间很多值是一样或可以用公式计算出来其连续运算的值。

我们记得

$$\begin{cases} k = \left\lfloor \frac{n}{l} \right\rfloor \\ r = \max(i), ik \leq n \end{cases}$$

$$r = \left\lfloor \frac{n}{k} \right\rfloor = \left\lfloor \frac{n}{\left\lfloor \frac{n}{l} \right\rfloor} \right\rfloor$$

推导方式：

```
ans = 0;
for(int l = 1, r; l <= n; l = r + 1){
    r = n / (n / l);
    ans += n / l * (r - l + 1);
}
```

1.35 求和性质：

- 提取：

$$\sum_{i=1}^n g(k, l) f(i, j) = g(k, l) \sum_{i=1}^n f(i, j)$$

$g(k, l)$ 是与下标 i 无关的函数

- 交换

$$\sum_{i=1}^4 \sum_{j=1}^i f(i, j) = \sum_{j=1}^4 \sum_{i=j}^4 f(i, j)$$

因为： $[1 \leq i \leq 4][1 \leq j \leq i] = [1 \leq j \leq i \leq 4] = [1 \leq j \leq 4][j \leq i \leq 4]$