```
/*****************************************************************
                                        康拓展开
 *****************************************************************/
LL fac[15];
void init(){
    fac[1]=1;
    for (int i=2;i<=11;i++) fac[i]=fac[i-1]*i;
}
void cantor(int s[], LL num, int k){//康托展开，把一个数字num展开成一个数组s，k是数组长度
    int t;
    bool h[k];//0到k-1，表示是否出现过
    memset(h, 0, sizeof(h));
    for(int i = 0; i < k; i ++){
        t = num / fac[k-i-1];
        num = num % fac[k-i-1];
        for(int j = 0, pos = 0; ; j ++, pos ++){
            if(h[pos]) j --;
            if(j == t){
                h[pos] = true;
                s[i] = pos + 1;
                break;
            }
        }
    }
}


/*****************************************************************
                                        康拓逆展开
 *****************************************************************/
void inv_cantor(int s[], LL &num, int k){//康托逆展开，把一个数组s换算成一个数字num
    int cnt;
    num = 0;
    for(int i = 0; i < k; i ++){
        cnt = 0;
        for(int j = i + 1; j < k; j ++){
            if(s[i] > s[j]) cnt ++;//判断几个数小于它
        }
        num += fac[k-i-1] * cnt;
    }
}
```

```
/*******************************************************************
                    组合数打表（乘法逆元，阶乘，逆元阶乘）
 *******************************************************************/
LL     inv[N],                     //逆元
F[N],                    //阶乘
Finv[N];                 //逆元阶乘
void build_comb(){
    inv[1]=F[0]=Finv[0]=1;
    for (int i=2;i<N;i++)
        inv[i]=(mod-mod/i)*inv[mod%i]%mod;
    for (int i=1;i<N;i++){
        F[i]=F[i-1]*i%mod;
        Finv[i]=Finv[i-1]*inv[i]%mod;
    }
}

/********************************************************************
                              组合数
 ********************************************************************/
LL comb(LL n,LL m){             //返回C n m
    if (m>n || m<0) return 0;
    return F[n]*Finv[m]%mod*Finv[n-m]%mod;
}

/*********************************************************************
                    组合数N方打表  （当无需取模且N很小时）
 *********************************************************************/
LL C[1000][1000];
void build_comb_nfang(int n){
    for (int i=0;i<n;i++) {
        C[i][0]=C[i][i]=1;
        for (int j=1;j<i;j++)
            C[i][j] =( C[i-1][j] + C[i-1][j-1] )%mod;
    }
}

/**********************************************************************
                              卢卡斯定理
         C(n, m) % p  =  C(n / p, m / p) * C(n%p, m%p) % p
 **********************************************************************/
LL Lucas_comb(LL n,LL m,LL p){
    if ((n/p)>p) return Lucas_comb(n/p,m/p,p)*comb(n%p,m%p)%p;
    else         return comb(n/p,m/p)*comb(n%p,m%p)%p;
}
```

```
typedef long long LL;
typedef pair<int,int> PLL;
/******************************************************************
                     最大公约数gcd、最小公倍数lcm
 ******************************************************************/
LL gcd(LL a,LL b){
    return b ? gcd(b,a%b) : a;
}
LL lcm(LL a,LL b){
    return a/gcd(a,b)*b;
}


/*******************************************************************
                 快速乘 (很少用,当a*a>long long时采用)
 ******************************************************************/
LL quick_mul(LL a,LL b,LL p) {                    //a*b%p
    LL ans=0;
    a%=p;
    b%=p;
    while(b) {
        if (b&1) ans=(ans+a)%p;
        a=(a+a)%p;
        b>>=1;
    }
    return ans;
}

/*******************************************************************
                         快速幂
     欧拉取模进化公式:     (a^b)%p=( a%p )^( b%phi(p) )%p;
     超欧拉取模进化公式: (a^b)%p=( a%p )^( phi(p)+b%phi(p) )%p
 *******************************************************************/
LL quick_pow(LL a,LL b,LL p) {                           //a的b次方 , %p
    LL ans=1;
    a%=p;
    while(b) {
        if (b&1) ans=ans*a%p;
        a=a*a%p;
        b>>=1;
    }
    return ans;
}

/********************************************************************
                             扩展欧几里得
 ********************************************************************/
void ex_gcd(LL a,LL b,LL &x,LL &y,LL &d) {
    if (!b) {
        d=a;
        x=1;
        y=0;
    } else {
        ex_gcd(b,a%b,y,x,d);
        y-=x*(a/b);
    }
```

```
}
/************************************************************************
                                单个逆元
 对于(a/b)%p=a*inv(b)
 1. (b与p互质)  欧拉定理 b关于p的逆元              inv(b)=b^( phi(p)-1 )
 2. (b与p互质并且p为素数) 费马小定理 b关于p的逆元        inv(b)=b^(p-2)
 3. (a与p不互质) (a/b)%p=
 ***********************************************************************/
LL get_inv(LL a,LL p) {         //    a关于p的逆元
    //如果p是素数，直接返回  a的p-2次方%p        quick_pow(a,p-2,p);
    LL d,x,y;
    ex_gcd(a,p,x,y,d);
    return (d==1) ? (x%p+p)%p : -1;
}

/************************************************************************
                            关于p乘法逆元打表
 ***********************************************************************/
LL inv[N];
void build_inv(int p) {                         //O(n)关于p线性打逆元表
    inv[1]=1;
    for (int i=2; i<N; i++)
        inv[i]=(p-p/i)*inv[p%i]%p;
}


/***********************************************************************
                            中国剩余定理
         n个方程: x=a[i](mod m[i]) (0<=i<n)    m之间 两两互质

                        W= m1*m2*m3  *mn
                            Mi=W/mi;
                        ti=inv(Mi,mi);
                        x=sigma(ai * ti * Mi );
 **********************************************************************/

LL china(int n, LL *a, LL *m) {
    LL M = 1, ret = 0;
    for(int i = 0; i < n; i ++) M *= m[i];
    for(int i = 0; i < n; i ++) {
        LL w = M / m[i];
        ret = (ret + w * get_inv(w, m[i]) * a[i]) % M;
    }
    return (ret + M) % M;
}
/***********************************************************************
                            中国剩余定理
     n个方程: A[i]x=B[i](mod M[i]) (0<=i<n)    m之间不互质
 **********************************************************************/
PLL linear(LL A[], LL B[], LL M[], int n) {
    //求解A[i]x = B[i] (mod M[i]),总共n个线性方程组
    LL x = 0, m = 1;
    for(int i = 0; i < n; i ++) {
        LL a = A[i] * m, b = B[i] - A[i]*x, d = gcd(M[i], a);
        if(b % d != 0)  return PLL(0, -1);//答案不存在，返回-1
```

```
        LL t = b/d * inv(a/d, M[i]/d)%(M[i]/d);
        x = x + m*t;
        m *= M[i]/d;
    }
    x = (x % m + m ) % m;
    return PLL(x, m);//返回的x就是答案，m是最后的lcm值
}
```

```
/*********************************************************************
                   欧拉函数O(n) +素数打表 + 莫比乌斯函数
 *********************************************************************/
void build_phi(){
    memset(phi,0,sizeof(phi));
    phi[i]=1;
    for (int i=2;i<=N;i++) {
        if (!phi[i]) {
            primes.push_back(i);
            phi[i]=i-1;
            mu[i]=-1;
        }
        for (int j=0;j<primes.size() && i*primes[j]<N;j++){
            if (i%primes[j]){
                phi[i*primes[j]]=phi[i]*(primes[j]-1);
                mu[i*primes[j]]=-mu[i];
            }else{
                phi[i*primes[j]]=phi[i]*primes[j];
                mu[i*primes[j]]=0;
                break;
            }
        }
    }
}
```