数学

数论

算术基本定理

算术基本定理可表述为：任何一个大于 1 的自然数 N ,如果 N 不为质数，那么 N 可以唯一分解成有限个质数的乘积$N = P_1^{a_1} * P_2^{a_2} * P_3^{a_3} ..... P_n^{a_n}$，这里$P_1 < P_2 < P_3 ..... < P_n$均为质数，其中指数 a_i 是正整数。这样的分解称为 N 的标准分解式。

例题：$N!$ 分解质因数

思路：如果对每个数都分解质因数的话，其复杂度为$O(N\sqrt{N})$，会造成TLE，而如果我们提前将N范围内的质数给筛出来，再分解枚举每个质数，再算其指数是多少，其复杂度会大大降低。

代码：

```
const int N = 1e6+10;

int primes[N],cnt;
bool st[N];

void init(int n)
{
    rep(i,2,n){
        if(!st[i])
            primes[cnt++] = i;
        for(int j=0;primes[j]<=n/i;j++){
            st[i*primes[j]] = true;
            if(i%primes[j] == 0)
                break;
        }
    }
}

int main()
{
    int n;
    cin >> n;
    init(n);

    for(int i=0;i<cnt;i++){
        int p = primes[i];
        int s = 0;
        for(int j=n;j;j/=p)
            s += j/p;
        cout << p << " " << s << endl;
    }

    return 0;
}
```

约数相关

基本思想：

如果$N = p_1^{c_1} * p_2^{c_2} * \ldots * p_n^{c_n}$

那么约数个数为：$(c_1 + 1) * (c_2 + 1) * \ldots * (c_n + 1)$

约数之和为：$(p_1^0 + p_1^1 + \ldots + p_1^{c_1}) * (p_2^0 + p_2^1 + \ldots + p_2^{c_2}) * \ldots * (p_n^0 + p_n^1 + \ldots + p_n^{c_n})$

代码：

```
typedef long long ll;
const int mod = 1e9 + 7;

int main()
{
    int n;
    cin >> n;

    unordered_map<int,int> primes;
    while(n--){
        int x;
        cin >> x;
        for(int i=2;i<=x/i;i++){
            if(x%i==0){
                while(x%i==0){
                    x /= i;
                    primes[i]++;
                }
            }
        }
        if(x>1) primes[x]++;
    }

    ll res = 1;
    for(auto p : primes){
        ll x = p.first, a = p.second;
        ll t = 1;
        while(a--) t = (t * x + 1) % mod;
        res = res * t % mod;
    }
    cout << res << endl;
    return 0;
}
```

线性筛

```
const int N = 1e6 + 10;

int cnt;
int primes[N];
bool st[N];

void get_primes(int n)
{
    st[1] = true;
```

```
    for(int i=2;i<=n;i++){
        if(!st[i]) primes[cnt++] = i;
        for(int j=0;primes[j]<=n/i;j++){
            st[primes[j]*i] = true;
            if(i % primes[j] == 0)
                break;
        }
    }
}
```

扩展应用：区间筛

题目描述：给定l,r区间，求出l,r区间内的所有质数

数据范围: $0 <= l,r <= 2^{31}-1, 0 <= r-l <= 1e6$

思路：l,r 的数值很大，但是这个区间长度是 $1e6$ ，直接线性筛时间会 T 掉,考虑先把 $\sqrt{r}$ 内的所有质数筛出来，然后在把这些质数在 l,r 区间内的数给筛掉，原理是一个数的质因子必然在 $\sqrt{x}$ 以内。

```
const int N = 1e6+10;

int primes[N],cnt;
bool st[N];

void init(int n)
{
    memset(st,0,sizeof st);
    cnt = 0;

    rep(i,2,n){
        if(!st[i])
            primes[cnt++] = i;
        for(int j=0;primes[j]<=n/i;j++){
            st[primes[j]*i] = true;
            if(i%primes[j] == 0)
                break;
        }
    }
}

int main()
{
    int l,r;
    while(cin >> l >> r){
        init(50000);

        memset(st,0,sizeof st);
        for(int i=0;i<cnt;i++){
            ll p = primes[i];
            for(ll j=max(p*2,(l+p-1)/p*p);j<=r;j+=p)
                st[j-l] = true;
        }
        cnt = 0;
        for(int i=0;i<=r-l;i++){
            if(!st[i] && i+l>=2)
                primes[cnt++] = i+l;
        }
    }
```

```
    return 0;
}
```

gcd/lcm

优化 gcd

```
int gcd(int a,int b)
{
    int az=__builtin_ctz(a),bz=__builtin_ctz(b);
    int z=min(az,bz);
    int dif;
    b>>=bz;
    while(a)
    {
        a>>=az;
        dif=b-a;
        az=__builtin_ctz(dif);
        if(a<b) b=a;
        if(dif<0) a=-dif;
        else a=dif;
    }
    return b<<z;
}
```

时间复杂度 $O(log(n))$

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

补充：另外，对于 C++14，我们可以使用自带的 `__gcd(a,b)` 函数来求最大公约数。而对于 C++ 17，我们可以使用头中的std::gcd和std::lcm来求最大公约数和最小公倍数。

多个数的最小公倍数

可以发现，当我们求出两个数的 gcd 时，求最小公倍数是$O(1)$ 的复杂度。那么对于多个数，我们其实没有必要求一个共同的最大公约数再去处理，最直接的方法就是，当我们算出两个数的 gcd，或许在求多个数的 gcd 时候，我们将它放入序列对后面的数继续求解，那么，我们转换一下，直接将最小公倍数放入序列即可。例 2，3，4先求2和3的最小公倍数为6，再求6和4的最小公倍数为12。

快速幂

```
ll qmi(ll a,ll k,ll p)
{
    ll res = 1%p;
    a %= p;
    while(k){
        if(k&1) res = res*a%p;
        k >>= 1;
        a = a*a%p;
    }
    return res;
}
```

逆元

乘法逆元的定义

> 若整数b,m互质，并且对于任意的整数a，如果满足$b|a$，则存在一个整数x，使得$a/b \equiv a*x(\mod m)$，则称x为b的模m乘法逆元，记为$b^{-1}(\mod m)$。
>
> b 存在乘法逆元的充要条件是 b 与模数 m 互质。当模数m为质数时,b^{m-2}即为b的乘法逆元

当模数是质数时，此时 b^{m-2} 即为 b 的逆元，直接套用快速幂的板子即可。

当模数不是质数时，需要列出方程并转化为扩展欧几里得进行求解。

扩展欧几里得

主要是用来求解形如$ax+by=gcd(a,b)$类型的这种方程。

原理：略

应用：对于求解更一般的方程$ax+by=c$

设 $d=gcd(a,b)$ 则其有解当且仅当 $d|c$
求解方法如下:

用扩展欧几里得求出$ax_0+by_0=d$的解

则$a(x_0*c/d)+b(y_0*c/d)=c$

故而特解为 $x_1=x_0*c/d, y_1=y_0*c/d$

而通解 = 特解 + 齐次解

而齐次解即为方程 $ax+by=0$ 的解

故而通解为 $x=x_1+k*b/d, y=y_1-k*a/d \quad k\in z$
若令 $t=b/d$ 则对于x的最小非负整数解为 $(x_1\%t+t)\%t$

模板:

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

int main()
{
    int n;
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        int a,b,x,y;
        cin >> a >> b;
        int d = exgcd(a,b,x,y);
        cout << x << " " << y << endl;
    }
```

```
    return 0;
}
```

欧拉函数

欧拉函数的定义

> $1 \sim N$中与N互质的数的个数被称为欧拉函数，记为$\phi(N)$。
> 若在算数基本定理中，$N = p^{a_1}p^{a_2}\dots p^{a_m}$，则：
> $\phi(N) = N \times \frac{p_1-1}{p_1} \times \frac{p_2-1}{p_2} \times \dots \times \frac{p_m-1}{p_m}$

证明：利用容斥原理进行证明

设 sum 为 1~n 中与 n 互斥的数的个数

基本思路是去掉 1~n 中所有 $p_1, p_2, \dots, p_n$ 的倍数

（1）去掉所有质数的倍数

$sum = n - \frac{n}{p_1} - \frac{n}{p_2} - \dots - \frac{n}{p_n}$

（2）由于质数之间存在公倍数的问题，导致有些数被多减了几次，因此需要加回

$n - \frac{n}{p_1} - \frac{n}{p_2} - \dots - \frac{n}{p_n} + \frac{n}{p1*p2} + \frac{n}{p1*p3} + \dots + \frac{n}{p_{k-1}*p_k}$

（3）以此类推，最后将n提出来就可以化简到原来的形式。

在求解一个数的欧拉函数时，可以试除法来进行求解。

模板代码：

```
int get_ola(int a)
{
    int res = a;
    for(int i=2;i<=a/i;i++){
        if(a%i == 0){
            res = res/i*(i-1); // 采用先除后乘防止爆int范围
            while(a%i == 0)
                a /= i;
        }
    }
    if(a>1) res = res/a*(a-1);
    return res;
}
```

在求解1 ~ n这个区间内的所有数的欧拉函数之和时，要用到筛法，其时间复杂度是$O(n)$和线性筛的复杂度是一样的；

证明：

（1）质数i的欧拉函数即为$phi[i] = i - 1$，1 ~ $i - 1$均与i互质，共$i - 1$个。

（2）$phi[primes[j] * i]$分为两种情况：
①$i \% primes[j] == 0$时：$primes[j]$是i的最小质因子，也是$primes[j] * i$ 的最小质因子，因此$1 - 1/primes[j]$这一项在$phi[i]$中计算过了，只需将基数N修正为$primes[j]$倍，最终结果为$phi[i] * primes[j]$;
② $i \% primes[j] != 0$: $primes[j]$不是i的质因子，只是$primes[j] * i$的最小质因子，因此不仅需要

将基数 N 修正为$primes[j]$倍，还需要补上$1-1/primes[j]$这一项，因此最终结果 $phi[i]*(primes[j]-1)$。

代码：

```
const int N = 1e6 + 10;
int primes[N],cnt;
int phi[N];
bool st[N];

ll get_ola(int n)
{
    phi[1] = 1;
    for(int i=2;i<=n;i++){
        if(!st[i]){
            primes[cnt++] = i;
            phi[i] = i - 1;
        }
        for(int j=0;primes[j]<=n/i;j++){
            st[primes[j] * i] = true;
            if(i%primes[j] == 0){
                phi[i*primes[j]] = primes[j] * phi[i];
                break;
            }
            phi[i*primes[j]] = phi[i] * (primes[j] - 1);
        }
    }

    ll res = 0;
    for(int i=1;i<=n;i++)
        res += phi[i];

    return res;
}

int main()
{
    int n;
    cin >> n;

    cout << get_ola(n) << endl;

    return 0;
}
```

费马定理/欧拉定理

费马定理

定义：

若p为素数，$gcd(a,p)=1$,则$a^{p-1}=1(\mathrm{mod}\ p)$.

另一个形式：对于任意整数a,有$a^p=a(\mathrm{mod}\ p)$

欧拉定理

若$gcd(a,m)=1$,则$a^{\phi(m)}=1(\mathrm{mod}\ m)$;

其中$\phi(m)$为 m 的欧拉函数

推论

若正整数 a 与 m 互质，则 $a^b \equiv a^{b \bmod \phi(m)} \pmod m$

简单证明：

$a^b=a^{\phi(m)*\lfloor b/\phi(m)\rfloor+b \bmod \phi(m)} \equiv 1*a^{b \bmod \phi(m)}(\mathrm{mod}\ m)$

扩展欧拉定理

为了解决 a 与 m 不互质时的此问题，我们引入扩展欧拉定理

这里不再引入证明，直接给出结论：

$$a^b \equiv \begin{cases} a^{b \bmod \phi(p)}, & gcd(a,p)=1 \\ a^b, & gcd(a,p)\neq 1, b<\phi(p) \pmod p \\ a^{b \bmod \phi(p)+\phi(p)}, & gcd(a,p)\neq 1, b\geq \phi(p) \end{cases}$$

简单理解:当 a 与 p 互质时，可以直接用欧拉定理求解，当 a 与 p 不互质时，并且 b 要小于 p 的欧拉函数时，可以直接利用快速幂求解，而当 b 要大于 p 的欧拉函数时，需要对 p 的欧拉函数进行取模操作。

原根/阶

类欧几里得

同余方程组

中国剩余定理

中国剩余定理 $(ChineseRemainderTheorem, CRT)$可求解如下形式的一元线性同余方程组（其中$n_1,n_2,\ldots,n_k$两两互质）：

$x \equiv a_1(\mathrm{mod}\ n_1)$

$x \equiv a_2(\mathrm{mod}\ n_2)$

$x \equiv a_k(\mathrm{mod}\ n_k)$

下面的「曹冲养猪」问题就是一元线性同余方程组的一个实例。

过程：

1. 计算所有模数的积n;
2. 对于第i个方程：
    1. 计算$m_i=n/n_i$;
    2. 计算m_i在模n_i意义下的逆元m_i^{-1};
    3. 计算$c_i=m_im_i^{-1}$。
3. 方程组在模意义下的唯一解为$x=\sum_{i=1}^{n}a_ic_i(\mathrm{mod}\ n)$

证明：略

代码：

```
const int N = 1e5+10;
```

```
ll a[N],m[N],m1[N];

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

ll crt(int n)
{
    ll M = 1;
    for(int i=1;i<=n;i++)
        M = M * m[i];
    for(int i=1;i<=n;i++){
        ll x,y;
        exgcd(M/m[i],m[i],x,y);
        m1[i] = (x%m[i]+m[i])%m[i];
        m[i] = M / m[i];
    }
    ll res = 0;
    for(int i=1;i<=n;i++)
        res = (res + (a[i] * m[i] * m1[i])) % M;
    return res;
}
```

扩展中国剩余定理

扩展中国剩余定理解决的是中国剩余定理中模数不互质的情况

思路：前两个方程 $x \equiv r_1 \pmod{m_1}, x \equiv r_2 \pmod{m_2}$转换为不定方程：
$x = m_1p + r_1 = m_2q + r_2$ 即$m_1p - m_2q = r_2 - r_1$

由裴蜀定理，当$gcd(m_1, m_2) \nmid (r_2 - r_1)$时，无解，当$gcd(m_1, m_2) | (r_2 - r_1)$时，有解

由扩欧算法，得特解$p = p * (r_2 - r_1)/gcd, q = q * (r_2 - r_1)/gcd$，其通解
$P = p + m_2/gcd * k, Q = q - m_1/gcd * k$

所以$x = m_1P + r_1 = m_1 * m_2/gcd + m_1p + r_1$，前两个方程等价合并为一个方程
$x \equiv r \pmod{m}$

其中$r = m_1p + r_1, m = lcm(m_1, m_2)$，所以$n$个同余方程只要合并$n - 1$次即可求解

代码：

```
ll n;
ll m[N],r[N];

ll exgcd(ll a,ll b,ll &x,ll &y)
{
    if(b == 0){
        x = 1;
        y = 0;
        return a;
```

```
    }

    ll d = exgcd(b,a%b,y,x);
    y -= (a/b)*x;
    return d;
}

ll mul(ll a,ll k,ll p) // 龟速乘
{
    ll res = 0;
    while(k){
        if(k&1)
            res = (res + a)%p;
        a = (a+a)%p;
        k >>= 1;
    }
    return res;
}

ll excrt()
{
    ll m1,m2,r1,r2,p,q;
    m1=m[1],r1=r[1];
    for(int i=2;i<=n;i++){
        m2=m[i],r2=r[i];
        ll temp=((r2-r1)%m[i]+m[i])%m[i];
        ll d = exgcd(m1,m2,p,q);
        if((r2-r1)%d){
            return -1;
        }
        p=p*((r2-r1)/d);
//      p=mul(p,temp/d,m[i]); // 如果有溢出的话可以使用龟速乘
        p=(p%(m2/d)+(m2/d))%(m2/d);
        r1=m1*p+r1;
        m1=m1/d*m2;
    }
    return (r1%m1+m1)%m1;
}
```

$Lucas$定理

普通$Lucas$

引入：$Lucas$ 定理用于求解大组合数取模的问题，其中模数必须为**素数**。正常的组合数运算可以通过递推公式求解，但当问题规模很大，而模数是一个不大的质数的时候，就不能简单地通过递推求解来得到答案，需要用到 Lucas 定理。

定义：$Lucas$ 定理内容如下：对于质数p，有

$$C_n^m \bmod p = C_{n/p}^{m/p} * C_{n\%p}^{m\%p} \bmod p$$

观察上述表达式，可知 $n \bmod p$ 和 $m \bmod p$ 一定是小于 p 的数，可以直接求解，$C_{n/p}^{m/p}$可以继续用 $Lucas$ 定理求解。这也就要求 p 的范围不能够太大，一般在 10^5 左右。边界条件：当 $m == 0$ 的时候，返回 1 。

时间复杂度为$O(f(n)+g(n)logn)$，其中为 $f(n)$ 预处理组合数的复杂度，$g(n)$ 为单次求组合数的复杂度。

模板：

```
int p;

int qmi(int a,int k)
{
    int res = 1 % p;
    while(k){
        if(k&1)
            res = res * 1ll * a % p;
        k >>= 1;
        a = a * 1ll * a % p;
    }
    return res;
}

int C(int a,int b)
{
    if(b>a)
        return 0;
    int res = 1;
    for(int i=1,j=a;i<=b;i++,j--){
        res = res * 1ll * j % p;
        res = res * 1ll * qmi(i,p-2) % p;
    }
    return res;
}

int lucas(ll a,ll b)
{
    if(a<p && b<p)
        return C(a,b);
    return C(a%p,b%p) * 1ll * lucas(a/p,b/p) % p;
}
```

$exLucas$

离散对数

$BSGS$

应用场景：给定正整数 a,p,b ,数据保证 a 与 p 互质，求满足 $a^x = b(\mod p)$ 的最小非负整数 x.

思路：

我们先讨论 a 与 p 互质的情况。

首先由欧拉定理可以知道

$$a^t \equiv a^{t \bmod \varphi(p)} \pmod p$$

而又知 $\phi(p) < p$，那么考虑暴力枚举 $1 \sim p$，一定可以找到最小的非负整数 t,

设 $k = \sqrt{p} + 1, t = kx - y$,其中 $1 <= x <= k, 0 <= y <= k - 1$

这样通过枚举 x, y，一定可以遍历到 $1 \sim p$ 。当然开始时要对 $x = 0$ 进行特判。

进一步考虑优化：将问题转化为求解

$$a^{kx} = b * a^y \bmod p$$

用一个 $Hash$ 表将右边所有的取值与 y 的对应存下来，然后枚举 x，在 $Hash$ 表中查找是否有与 a^{kx} 对应的 y 即可。预处理和枚举的时间复杂度都是 $O(\sqrt{p})$，因此总时间复杂度就是 $O(\sqrt{p})$ 级别的。

时间复杂度：$O(\sqrt{p})$

代码：

```
ll BSGS(ll a,ll b,ll p)
{
    if(1ll%p == b%p)
        return 0;
    ll k = sqrt(p)+1;
    unordered_map<ll,ll> hash;
    for(ll i=0,j=b%p;i<k;i++){
        hash[j] = i;
        j = j*a%p;
    }
    ll ak=1;
    for(ll i=0;i<k;i++)
        ak = ak*a%p;
    for(ll i=1,j=ak;i<=k;i++){
        if(hash.count(j))
            return i*k-hash[j];
        j = j*ak%p;
    }

    return -1;
}
```

$exBSGS$

给定正整数a,b,p，a,p 不一定互质，求满足

$$a^x \equiv b(\mathrm{mod}\ p)$$

的最小非负整数 x

分情况来看，如果 $x=0$ 时，满足 $1 \equiv b(\mathrm{mod}\ p)$,答案就是 0

如果 $gcd(a,p)=1$，那么就用朴素的 $BSGS$ 算法。

如果 $gcd(a,p)>1$，则

$$a^x+kp=b$$

设 $gcd(a,p)=d$，如果 $b \nmid d$，则输出无解。否则等式两边同除 d 得

$$\frac{a}{b}a^{x-1} \equiv \frac{b}{d}(\mathrm{mod}\ \frac{p}{d})$$

由于 $gcd(\frac{a}{d},\frac{p}{d})=1$，所以把 $\frac{a}{d}$ 移到等式右边，就是乘 $\frac{a}{d}$ 的逆元

$$a^{x-1} \equiv \frac{b}{d}(\frac{a}{d})^{-1}(\mathrm{mod}\ \frac{p}{d})$$

用新变量替换得到

$$(a_1)^x \equiv b_1(\mathrm{mod}\ p_1)$$

由于 $x>=1$ ，这样就可以递归地用扩展 $BSGS$ 求解了，新的解就为 $x+1$,逆元可以用扩欧算法进行求解。

代码模板：

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

ll BSGS(ll a,ll b,ll p)
{
    if(1ll%p == b%p)
        return 0;
    ll k = sqrt(p)+1;
    unordered_map<ll,ll> hash;
    for(ll i=0,j=b%p;i<k;i++){
        hash[j] = i;
        j = j*a%p;
    }
    ll ak=1;
    for(ll i=0;i<k;i++)
        ak = ak*a%p;
    for(ll i=1,j=ak;i<=k;i++){
        if(hash.count(j))
            return i*k-hash[j];
        j = j*ak%p;
    }

    return -inf;
}

ll exBSGS(ll a,ll b,ll p)
{
    b = (b%p+p)%p;
    if(1%p == b%p)
        return 0;
    ll x,y;
    ll d = exgcd(a,p,x,y);
    if(d>1){
        if(b%d)
            return -inf;
        exgcd(a/d,p/d,x,y);
        return exBSGS(a,b/d*x%(p/d),p/d)+1;
    }
    return BSGS(a,b,p);
}

int main()
{
    IOS;
    ll a,p,b;
    while(cin >> a >> p >> b, a || p || b){
        ll res = exBSGS(a,b,p);
        if(res<0)
            cout << "No Solution" << endl;
```

```
        else
            cout << res << endl;
    }

    return 0;
}
```

$Pohing-Hellman$

数论函数

整除分块

求$\Sigma_{i=1}^{n}\frac{n}{i}$ （全为下取整）

性质1：分块的块数$<=2\lfloor\sqrt{n}\rfloor$

当$i<=\lfloor\sqrt{n}\rfloor$时，$\lfloor\frac{n}{i}\rfloor$有$\lfloor\sqrt{n}\rfloor$种取值。

当$i>\lfloor\sqrt{n}\rfloor$时，$\lfloor\frac{n}{i}\rfloor$至多有$\lfloor\sqrt{n}\rfloor$种取值

性质2：i所在的右端点为$\lfloor\frac{n}{\lfloor\frac{n}{i}\rfloor}\rfloor$

i所在的块的值为$k=\lfloor\frac{n}{i}\rfloor$，则$k<=\frac{n}{i}$，则$\lfloor\frac{n}{k}\rfloor>=\lfloor\frac{n}{\lfloor\frac{n}{i}\rfloor}\rfloor=\lfloor i\rfloor=i$，所以，代码实现时,右断点$r=n/(n/l)$

例如： 求$\Sigma_{i=1}^{n}f(i)\frac{n}{i}$

需要预处理的前缀和，在枚举每一块，累加每块的贡献

```
for(int l=1;l<=n;l=r+1){
    r = n/(n/l);
    res += (s[r]-s[l-1])*(n/l)
}
```

狄利克雷卷积

莫比乌斯反演

基础：莫比乌斯函数

$$u(n)=\begin{cases}1 & 若n=1\\ (-1)^k & 若n无平方数因数，且n=p_1*p_2*p_3*...*p_k;\\ 0 & 若n有大于1的平方数因数\end{cases}$$

应用：题意：给定a,b,d，若$1\le x\le a$，$1\le y\le b$, 求有多少对x和y,使得$gcd(x,y)=d$

转化=>令 $x\prime=x/d$，$y\prime=y/d$, 则$1\le x\prime\le a/d$，$1\le y\prime\le b/d$,即求有多少对 x' 和 y' 互质。

如何求总共有多少对合法的呢？这里用到补集的思想：合法对数= 总对数-不合法对数。

不合法对：x'和 y'的最大公因数大于1。
令$a\prime=a/d$，$b\prime=b/d$
，利用容斥原理求得不合法对数：

$a'b'-a'/2*b'/2-a'/3*b'/3-...$(有一个质公因子)

$+a'/6*b'/6+...$ (有两个不同的质公因子)

$-a'/30*b'/30-\ldots$ (有三个不同的质公因子)

$=>\sum_{i=1}^{min(a',b')} a'/i*b'/i*mobius[i]$

很容易发现每个数前面的系数就是这个数的莫比乌斯函数。这就是莫比乌斯函数的由来。

该题由于有多组测试数据，因此需要用到数论分块，详情原理见数论分块。

代码：

```
const int N = 50010;

int primes[N],cnt;
bool st[N];
int mobius[N],sum[N];

// 莫比乌斯函数可以在O(n)的时间内求得
void init(int n)
{
    mobius[1] = 1;
    for(int i=2;i<=n;i++){
        if(!st[i]){
            primes[cnt++] = i;
            mobius[i] = -1;
        }
        for(int j=0;primes[j]<=n/i;j++){
            int t=primes[j]*i;
            st[t] = true;
            if(i % primes[j] == 0){
                mobius[t] = 0;
                break;
            }
            mobius[t] = mobius[i]*-1;
        }
    }
    for(int i=1;i<=n;i++)
        sum[i] = sum[i-1]+mobius[i];
}

int main()
{
    IOS;

    init(N-10);
    int t;
    cin >> t;
    while(t--){
        int a,b,d;
        cin >> a >> b >> d;
        a /= d,b /= d;
        int n=min(a,b);
        ll res=0;
        for(int l=1,r;l<=n;l=r+1){
            r = min(n,min(a/(a/l),b/(b/l)));
            res += 1ll*(sum[r]-sum[l-1])*(a/l)*(b/l);
        }
        cout << res << endl;
    }
```

```
    return 0;
}
```

min25筛

$Miller\_Rabin$

算法用处：快速判断一个数是不是质数

时间复杂度：最大为 $O(log^3 n)$，常数为7

注意事项：中间要用__int128进行过渡，如果编译器不支持int128, 改用龟速乘进行过度。

```
ll qpow(ll a,ll k,ll p)
{
    ll ans=1%p;
    while(k){
        if(k&1) ans = (__int128)ans*a%p;
        a = (__int128)a*a%p;
        k >>= 1;
    }
    return ans;
}

bool MRtest(ll x)
{
    if(x<3) return x==2;
    if(x%2==0) return false;
    ll A[]={2,325,9375,28178,450775,9780504,1795265022},d=x-1,r=0;
    while(d%2==0)
        d/=2,++r;
    for(auto a:A){
        ll v=qpow(a,d,x);
        if(v<=1 || v==x-1) continue;
        for(int i=0;i<r;++i){
            v = (__int128)v*v%x;
            if (v==x-1 && i!=r-1){
                v=1;
                break;
            }
            if(v==1) return false;
        }
        if(v!=1) return false;
    }
    return true;
}
```

$Pollard\_Rho$

算法用处：$Pollard\ rho$ 是一个非常玄学的方式，用于在 $O(n^{1/4})$ 的期望时间复杂度内计算合数 n 的某个非平凡因子（非1并且非自身的因子）。事实上算法导论给出的是 $O(\sqrt{p})$，p 是 n 的某个最小因子，满足 p 与 n/p 互质。但是这些都是期望，未必符合实际。但事实上 Pollard rho 算法在实际环境中运行的相当不错。

时间复杂度：$O(n^{1/4})$

注意事项：需要和 $Miller\_Rabin$ 配合一起使用，同时如果需要求最大质因数的话可以直接递归去求最大质因数，加一个简单的记忆化比不带记忆化的要快一点。

```
#include <ctime>

ll qpow(ll a,ll k,ll p)
{
    ll ans=1%p;
    while(k){
        if(k&1) ans = (__int128)ans*a%p;
        a = (__int128)a*a%p;
        k >>= 1;
    }
    return ans;
}

bool MRtest(ll x)
{
    if(x<3) return x==2;
    if(x%2==0) return false;
    ll A[]={2,325,9375,28178,450775,9780504,1795265022},d=x-1,r=0;
    while(d%2==0)
        d/=2,++r;
    for(auto a:A){
        ll v=qpow(a,d,x);
        if(v<=1 || v==x-1) continue;
        for(int i=0;i<r;++i){
            v = (__int128)v*v%x;
            if (v==x-1 && i!=r-1){
                v=1;
                break;
            }
            if(v==1) return false;
        }
        if(v!=1) return false;
    }
    return true;
}

template <class T>
T randint(T l,T r=0)
{
    static mt19937 eng(time(0));
    if(l>r) swap(l,r);
    uniform_int_distribution<T> dis(l,r);
    return dis(eng);
}

ll Pollard_Rho(ll N)
{
    if(N == 4) return 2;
    if(MRtest(N)) return N;
    while(1){
        ll c = randint(1ll,N - 1);
        auto f = [=](ll x){
            return ((__int128)x * x + c) % N;
        };
```

```
        ll t=0,r=0,p=1,q;
        do{
            for(int i=0;i<128;++i){
                t=f(t),r=f(f(r));
                if (t==r || (q=(__int128)p*abs(t-r)%N)==0) break;
                p = q;
            }
            ll d=gcd(p,N);
            if(d>1) return d;
        }while (t!=r);
    }
}

unordered_map<ll, ll> um;
ll max_prime_factor(ll x)
{
    if(um.count(x)) return um[x];
    ll fac = Pollard_Rho(x);
    if (fac == x) um[x] = x;
    else um[x] = max(max_prime_factor(fac), max_prime_factor(x / fac));
    return um[x];
}
```

偶尔方程

单位根反演

二次剩余

勒让德符号

$Cipolla$算法

线性代数

矩阵快速幂

应用：加速递推，通过分析递推方程来构造一个矩阵，使得递推方程能通过矩阵来计算，而后通过矩阵的快速幂来快速计算相关结果。

板子：

```
const int N=110;

int n,m;

struct Matrix{
    int n,m;
    ll a[N][N];
    Matrix(){
        n = m = 0;
    }
    Matrix(int n,int m):n(n),m(m){}
    void init(){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=m;j++)
```

```
                a[i][j] = 0;
        }
    };
    Matrix operator* (const Matrix B) const{
        Matrix C(n,B.m);
        C.init();
        for(int i=1;i<=n;i++){
            for(int j=1;j<=B.m;j++){
                for(int k=1;k<=m;k++)
                    C.a[i][j]+=a[i][k]*B.a[k][j];
            }
        }
        return C;
    }
    void print(){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=m;j++)
                cout<<a[i][j]<<" ";
            cout<<endl;
        }
    }
}f,c;

void qmi(ll k)
{
    f.n=f.m=3;
    c.n=c.m=3;
    f.a[1][1] = f.a[1][2] = f.a[1][3] = 1;
    c.a[1][2] = c.a[2][1] = c.a[2][2] = c.a[2][3] = c.a[3][3] = 1;
    while(k){
        if(k&1)
            f = f * c;
        c = c * c;
        k >>= 1;
    }
}
```

高斯消元

时间复杂度：$O(n^3)$

用途：

（1）通过初等行变换把增广矩阵化为阶梯型矩阵 并回代 得到方程的解。

（2）适用于求解包含n个方程组，n个未知数的多元线性方程组。

例如该方程组：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \qquad\qquad\qquad\qquad\qquad\quad \ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

其增广矩阵为：

$$\begin{pmatrix} a_{11} & a_{22} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ & & & \dots\dots & \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{pmatrix}$$

接下来所有操作都用该增广矩阵来代替原方程组

算法思想：

（1）将上述增广矩阵通过初等行变换变成阶梯型矩阵。

（2）再把阶梯型矩阵从下往上回代到第一层即可得到方程的解

算法步骤：

枚举每一列c

（1）找到当前列绝对值最大的一行
（2）用初等行变换 (2) 把这一行换到最上面（未确定阶梯型的行，并不是第一行），（只交换两行不改变矩阵的相关性质）
（3）用初等行变换 (1) 将该行的第一个数变成1（其余所有的数字依次跟着变化），（将改行乘上一个系统，矩阵的性质不发生变换）
（4）用初等行变换 (3) 将下面所有行的当且列的值变成 0（一行加上或者减去另一行乘上K倍，矩阵的性质不发生变换）。

代码：

```
const int N = 110;
const double eps = 1e-6;

double a[N][N];
int n;

int gauss()
{
    int c,r;
    for(c=0,r=0;c<n;c++){
        int t = r;
        for(int i=r;i<n;i++){
            if(fabs(a[i][c])>fabs(a[t][c]))
                t = i;
        }
        if(fabs(a[t][c])<eps)//如果当前列最大值为0 不必进行操作，直接跳出循环
            continue;

        for(int i=c;i<=n;i++)
            swap(a[t][i],a[r][i]);//将绝对值最大的列所在行与待处理的行交换
        for(int i=n;i>=c;i--)
            a[r][i] /= a[r][c];//此时r即为当前最大列所在的行 将首位变为1
        for(int i=r+1;i<n;i++){
            if(fabs(a[i][c])>eps){
                for(int j=n;j>=c;j--){
                    a[i][j] = a[i][j] - a[i][c] * a[r][j];
                }
            }
        } // 将剩下的所有行的当前列清0 第二层循环要反着写 最后一个处理 a[i][c]

        r++;//每次处理完了，将r++，r保留的是处理了多少行
```

```
    }

    if(r<n){//如果操作小于n行说明不是完美的阶梯形 只有完美阶梯型有唯一解
        for(int i=r;i<n;i++)
            if(fabs(a[i][n])>eps)//存在0 == ! 0的情况 无解
                return 2;
        return 1;//不存在则有无数组解
    }

    // 如果存在唯一解，逆序求每一个x的值 a[i][n] 即为xi的值
    //此时xi的系数已经为1 ，只需将xi以后的x系数全部清0即可 拿a[i][n] - 每个系数乘上x的值
    for(int i=n-1;i>=0;i--){
        for(int j=n-1;j>i;j--)
            a[i][n] = a[i][n] - a[i][j] * a[j][n];
    }

    return 0;
}

int main()
{
    cin >> n;
    for(int i=0;i<n;i++){
        for(int j=0;j<n+1;j++)
            cin >> a[i][j];
    }

    int t = gauss();
    if(t==0){
        for(int i=0;i<n;i++){
            if(fabs(a[i][n]) < eps)//可能会输出-0.00的情况 加以判断
                a[i][n] = 0;
            printf("%.2lf\n",a[i][n]);
        }
    }
    else if(t==1)
        puts("Infinite group solutions");
    else
        puts("No solution");

    return 0;
}
```

不带注释版本：

```
const int N = 110;
const double eps = 1e-6;

double a[N][N];
int n;

// 0代表唯一解 1代码无数解 2代表无解
int gauss()
{
    int c,r;
    for(c=0,r=0;c<n;c++){
        int t = r;
```

```
        for(int i=r;i<n;i++){
            if(fabs(a[i][c])>fabs(a[t][c]))
                t = i;
        }
        if(fabs(a[t][c])<eps)
            continue;

        for(int i=c;i<=n;i++)
            swap(a[t][i],a[r][i]);
        for(int i=n;i>=c;i--)
            a[r][i] /= a[r][c];
        for(int i=r+1;i<n;i++){
            if(fabs(a[i][c])>eps){
                for(int j=n;j>=c;j--){
                    a[i][j] = a[i][j] - a[i][c] * a[r][j];
                }
            }
        }
        r++;
    }

    if(r<n){
        for(int i=r;i<n;i++)
            if(fabs(a[i][n])>eps)
                return 2;
        return 1;
    }

    for(int i=n-1;i>=0;i--){
        for(int j=n-1;j>i;j--)
            a[i][n] = a[i][n] - a[i][j] * a[j][n];
    }

    return 0;
}

int main()
{
    cin >> n;
    for(int i=0;i<n;i++){
        for(int j=0;j<n+1;j++)
            cin >> a[i][j];
    }

    int t = gauss();
    if(t==0){
        for(int i=0;i<n;i++){
            if(fabs(a[i][n]) < eps)
                a[i][n] = 0;
            printf("%.2lf\n",a[i][n]);
        }
    }
    else if(t==1)
        puts("Infinite group solutions");
    else
        puts("No solution");

    return 0;
```

```
}
```

扩展应用：求解异或线性方程组

核心思路：

```
核心思想: 异或-不进位的加法
那么等式与等式间的异或要一起进行才能保证等式左右两边依然是相等关系!
 a^b^c = x
   d^f = y
   则
 a^b^d^c^f = x^y
1 左下角消0
  1.1 枚举列
  1.2 找第一个非零行
  1.3 交换
  1.4 把同列下面行消零(异或)
2 判断3种情况
  2.1 唯一解
  2.2 秩<n
      2.2.1 有矛盾 无解
      2.2.2 无矛盾 无穷多解
```

相关代码：

```
const int N = 110;

int a[N][N];
int n;

int gauss()
{
    int r,c;
    for(r=c=0;c<n;c++){
        int t = r;
        for(int i=r;i<n;i++){
            if(a[i][c]){
                t = i;
                break;
            }
        }
        if(a[t][c] == 0)
            continue;
        for(int i=c;i<n+1;i++)
            swap(a[t][i],a[r][i]);

        for(int i=r+1;i<n;i++){
            if(a[i][c]){
                for(int j=c;j<n+1;j++)
                    a[i][j] = a[i][j] ^ a[r][j];
            }
        }
        r++;
    }
    if(r<c){
        for(int i=r;i<n;i++){
```

```
            if(a[i][n])
                return 2;
        }
        return 1;
    }

    for(int i=n-1;i>=0;i--){
        for(int j=i+1;j<n;j++)
            a[i][n] = a[i][n] ^ (a[i][j] & a[j][n]);
    }

    return 0;
}

int main()
{
    cin >> n;
    for(int i=0;i<n;i++){
        for(int j=0;j<n+1;j++)
            cin >> a[i][j];
    }

    int res = gauss();

    if(res == 0){
        for(int i=0;i<n;i++)
            cout << a[i][n] << endl;
    }
    else if(res == 1)
        puts("Multiple sets of solutions");
    else
        puts("No solution");

    return 0;
}
```

线性幂

线性基

概念：在线性代数中，对于向量组 $\{a_1, a_2, \ldots, a_n\}$，我们把其张成空间的一组**线性无关**的基成为该向量组的线性基。(本质就是向量组可以表示出很多向量出来，而这些向量可以用数量更少的向量进行表示，因此如果直接求线性基的话可以简化操作)。

二进制集合 $S = \{x_1, x_2, \ldots, x_{n-1}, x_n\}$，得到另一个二进制集合 $S\prime = \{x_1, x_2, \ldots, x_n\}$，保证在 S 中任取子集 A，都能在 S' 中找到对应的子集 A'，使得 A 与 A' 的异或和相等；同时 A' 中任意一个元素都不能被 A' 中其他元素的组合异或出来。我们把 A' 称为 A 的**线性基**，利用它可以方便的求出原集合的**k大异或和**

构造方法有贪心法和高斯消元法两种方法

贪心法

对原集合的每个数 p 转为二进制，从高位向低位扫，对于第 x 位是 1 的，如果 a_x 不存在，那么令 $a_x < -p$ 并结束扫描，如果存在，令 $p < -p\ xor\ a_x$。

查询原集合内任意几个元素 xor 的最大值，只需将线性基从高位向低位扫，若 xor 上当前扫到的 a_x 答案变大，就把答案异或上a_x。

为什么能行呢？因为从高往低位扫，若当前扫到第 i 位，意味着可以保证答案的第 i 位为 1，且后面没有机会改变第 i 位。

查询原集合内任意几个元素 xor 的最小值，就是线性基集合所有元素中最小的那个。

查询某个数是否能被异或出来，类似于插入，如果最后插入的数 p 被异或成了 0，则能被异或出来。

代码：

```
template<int bit> struct LinearBasis
{
    vector<ll> v,tmp;
    int cnt;
    bool flag;
    LinearBasis(){
        flag = false;
        cnt = 0;
        tmp.resize(bit);
        v.resize(bit);
    }
    void insert(ll x){
        for (int i=bit-1;i>=0;--i){
            if (x>>i&1ll){
                if(v[i])
                    x ^= v[i];
                else{
                    v[i] = x;
                    return ;
                }
            }
        }
        flag = true;
        return;
    }
    ll query_max(){
        ll res = 0;
        for(int i=bit-1;i>=0;--i)
            res = max(res,res^v[i]);
        return res;
    }
    ll query_min(){
        if(flag)
            return 0;
        for(int i=0;i<=bit-1;++i)
            if(v[i])
                return v[i];
        return 0;
    }
    bool check(ll x){
        for(int i=bit-1;i>=0;--i){
            if((x>>i)&1ll)
                if(!v[i])
                    return false;
                else
                    x^=v[i];
        }
        return true;
    }
```

```
    void rebuild(){
        cnt = 0;
        for(int i=0;i<=bit-1;++i){
            for(int j=i-1;j>=0;--j)
                if(v[i]&(1ll<<j))
                    v[i]^=v[j];
            if(v[i])
                tmp[cnt++]=v[i];
        }
    }
    ll query(ll k){ // 必须先调用rebuild
        ll res = 0;
        k -= flag;
        if(!k)
            return 0;
        if(k>=(1ll<<cnt))
            return -1;
        for(int i=0;i<cnt;++i)
            if(k&(1ll<<i))
                res^=tmp[i];
        return res;
    }
    vector<ll> base()
    {
        vector<ll> res;
        for(int i = bit - 1; i >= 0; --i)
        {
            if(v[i])
                res.push_back(v[i]);
        }
        return res;
    }
};
```

高斯消元法：

高斯消元法相当于从线性方程组的角度去构造线性基，正确性显然。

矩阵求逆

常系数线性递推

矩阵树定理

BM

BEST 定理

特征值、特征向量

组合数学

组合数

```
using T = long long;
namespace binom {
    T fact[N],infact[N];
    int __ = []{
        fact[0] = 1;
        for(int i=1;i<=N-5;i++)
            fact[i] = fact[i - 1]*i%mod;
        infact[N-5]=qmi(fact[N-5],mod-2,mod);
        for(int i=N-5;i;i--)
            infact[i-1] = infact[i]*i% mod;
        return 0;
    }();

    inline T C(int n, int m)
    {
        if (n < m || m < 0)
            return 0;
        return fact[n]*infact[m]%mod*infact[n-m]%mod;
    }

    inline T A(int n, int m)
    {
        if (n < m || m < 0)
            return 0;
        return fact[n]*infact[n-m]%mod;
    }
}
using namespace binom;
```

牢记：熟练使用隔板法！！！！

杨辉三角

集组合数

二项式定理

概率论

期望的线性性

条件概率

容斥原理

基础

定义：设 U 中元素有 n 种不同的属性，而第 i 种属性称为 P_i，拥有属性 P_i 的元素构成集合 S_i，那么

$$\bigcup_{i=1}^{n}|S_i|=\sum_{i}|S_i|-\sum_{i<j}|S_i\cap S_j\cap S_k|-\ldots.+(-1)^{m-1}\sum_{a_i<a_{i+1}}|\bigcap_{i=1}^{m}S_i|-\ldots.$$

简单理解：奇正偶负，或者奇正偶负，需要注意的是用容斥原理的复杂度是 $O(2^n)$，当 n 的量级大约是 20 时可以考虑使用容斥原理。

例题：给定一个整数 n 和 m 个不同的质数 $p_1, p_2, \ldots, p_m$。

请你求出 $1 \sim n$ 中能被 $p_1, p_2, \ldots, p_m$ 中的至少一个数整除的整数有多少个。

代码：

```
const int N = 20;

int n,m;
int p[N];

int main()
{
    cin >> n >> m;
    for(int i=0;i<m;i++)
        cin >> p[i];

    int res = 0;

    for(int i=1;i<(1<<m);i++){
        int t = 1,cnt = 0;
        for(int j=0;j<m;j++){
            if((i>>j)&1){
                if((ll)t*p[j]>n){
                    t = -1;
                    break;
                }
                cnt++;
                t = t * p[j];
            }
        }
        if(t!=-1){
            if(cnt % 2 == 1)
                res += n / t;
            else
                res -= n / t;
        }
    }

    cout << res << endl;

    return 0;
}
```

min-max容斥

二项式反演

常见数列

斐波那契数列

例题：

代码：

```
const int N = 50;
```

```
ll f[N];

string check(int n,int x,int y)
{
    if(x==1 && y==1)
        return "YES";
    if(y<=f[n] && y>f[n-1])
        return "NO";
    if(y>f[n])
        y=y-f[n];
    return check(n-1,y,x);
}

void solve()
{
    int n,x,y;
    cin >> n >> x >> y;
    cout << check(n,x,y) << endl;
}

int main()
{
    IOS;
    f[0] = f[1] = 1;
    for(int i=2;i<=45;i++)
        f[i] = f[i-1]+f[i-2];
    int t;
    cin >> t;
    while(t--)
        solve();

    return 0;
}
```

错排问题

卡特兰数

解决的相关问题：

1. 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，问有多少种方法使得只要有 10 元的人买票，售票处就有 5 元的钞票找零?
2. 一位大城市的律师在她住所以北n个街区和以东n个街区处工作。每天她走$2n$个街区去上班。如果他从不穿越（但可以碰到)从家到办公室的对角线，那么有多少条可能的道路?
3. 在圆上选择$2n$个点，将这些点成对连接起来使得所得到的n条线段不相交的方法数?
4. 对角线不相交的情况下，将一个凸多边形区域分成三角形区域的方法数?
5. 一个栈（无穷大）的进栈序列为$1, 2, 3, \ldots, n$有多少个不同的出栈序列?
6. n个结点可构造多少个不同的二叉树?
7. n个$+1$和n个-1构成$2n$项$a_1, a_2, \ldots, a_{2n}$,其部分和满足
   $a_1 + a_2 + \ldots + a_k >= 0 (k = 0, 1, 2, 3, \ldots, 2n)$,有多少个这种数列?

   该递推关系的解为：第n项$C_n = C_{2n}^n/(n+1) \quad (n >= 2$且$n$为整数$)$

关于卡特兰数的常见公式：其中H_n表示第n项卡特兰数

$$H_n = \Sigma_{i=1}^{n} H_{i-1} H_{n-i}$$

$$H_n = \frac{H_{n-1}(4n-2)}{n+1}$$

$$H_n = C_{2n}^{n} - C_{2n}^{n-1}$$

递推代码：

```
const int N = 6e4+10;

ll f[N];

int main()
{
    int n;
    cin >> n;
    f[1] = 1;
    rep(i,2,n){
        f[i] = f[i-1]*(i*4-2)/(i+1);
    }
    cout << f[n] << endl;

    return 0;
}

/*
50的时候就已经爆ll了，如果数据范围很大需要写高精度。
*/
```

组合代数代码：

```
const int mod = 1e9+7;

ll qmi(ll a,ll k,ll p)
{
    ll ans = 1;
    while(k){
        if(k&1)
            ans = ans*a%mod;
        k >>= 1;
        a = a*a%mod;
    }
    return ans;
}

int main()
{
    int n;
    cin >> n;

    ll x = 1,y = 1;
    rep(i,2,n*2){
        x = x*i%mod;
        if(i<=n)
            y = y*i%mod;
```

```
    }

    ll ans = (((x*qmi(y,mod-2,mod)%mod)*qmi(y,mod-2,mod)%mod)*qmi(n+1,mod-
2,mod)%mod);
    cout << ans << endl;

    return 0;
}
```

拆分数

斯特林数

贝尔数

伯努利数

prufer序列

LGV引理

杨表

博弈论

Nim游戏

若一个游戏满足：

1. 由两名玩家交替行动
2. 在游戏进行的任意时刻，可以执行的合法行动与轮到哪位玩家无关
3. 不能行动的玩家判负

则称该游戏为一个公平组合游戏。

尼姆游戏（NIM）属于公平组合游戏，但常见的棋类游戏，比如围棋就不是公平组合游戏，因为围棋交战双方分别只能落黑子和白子，胜负判定也比较负责，不满足条件2和3。

例题：

题目描述

给定 n 堆石子，两位玩家轮流操作，每次操作可以从任意一堆石子中拿走任意数量的石子（可以拿完，但不能不拿），最后无法进行操作的人视为失败。
问如果两人都采用最优策略，先手是否必胜。

例如：有两堆石子，第一堆有 2 个，第二堆有 3 个，先手必胜。

操作步骤：

1. 先手从第二堆拿走 1 个，此时第一堆和第二堆数目相同
2. 无论后手怎么拿，先手都在另外一堆石子中取走相同数量的石子即可

必胜状态和必败状态
在解决这个问题之前，先来了解两个名词：

1. 必胜状态，先手**进行某一个操作**，留给后手是一个必败状态时，对于先手来说是一个必胜状态。即**先手可以走到某一个必败状态。**

2. 必败状态，先手**无论如何操作**，留给后手都是一个必胜状态时，对于先手来说是一个必败状态。即**先手走不到任何一个必败状态。**

结论

假设 n 堆石子，石子数目分别是 $a_1, a_2, \ldots, a_n$，如果$a_1 \oplus a_2 \oplus \ldots \oplus a_n \neq 0$, 先手必胜；否则先手必败。

证明

1. 操作到最后时，每堆石子数都是 0，$0 \oplus 0 \oplus \ldots 0 = 0$
2. 在操作过程中，如果 $a_1 \oplus a_2 \oplus \ldots \oplus a_n = x \neq 0$。那么玩家必然可以通过拿走某一堆若干个石子将异或结果变为 0。
   证明：不妨设x的二进制表示中最高一位 1 在第k位，那么在 $a_1, a_2, \ldots, a_n$中，必然有一个数 a_i，它的第 k 位是1，且 $a_i \oplus x < a_i$
   ，那么从第 i 堆石子中拿走 $(ai - ai \oplus x)$ 个石子，第 i堆石子还剩
   $a_i - (a_i - a_i \oplus x) = a_i \oplus x$，此时 $a_1 \oplus a_2 \oplus \ldots \oplus a_i \oplus x \oplus \ldots \oplus a_n = x \oplus x = 0$
3. 在操作过程中，如果 $a_1 \oplus a_2 \oplus \ldots \oplus a_n = 0$，那么无论玩家怎么拿，必然会导致最终异或结果不为 0
   反证法：假设玩家从第 i 堆石子拿走若干个，结果仍是0。不妨设还剩下 $a\prime$ 个，因为不能不拿，所以 $0 \leq a\prime < a_i$，且 $a_1 \oplus a_2 \oplus \ldots \oplus a\prime \oplus \ldots \oplus a_n = 0$
   。那么 $(a_1 \oplus a_2 \oplus \ldots \oplus a_i \oplus \ldots a_n) \oplus (a_1 \oplus a_2 \oplus \ldots \oplus a\prime \oplus \ldots \oplus a_n) = a_i \oplus a\prime = 0$
   ，则 $a_i = a\prime$，与假设 $0 \leq a\prime < a_i$矛盾。
   基于上述三个证明：
4. 如果先手面对的局面是 $a_1 \oplus a_2 \oplus \ldots \oplus a_n \neq 0$，那么先手总可以通过拿走某一堆若干个石子，将局面变成 $a_1 \oplus a_2 \oplus \ldots \oplus a_n = 0$
   。如此重复，最后一定是后手面临最终没有石子可拿的状态。先手必胜。
5. 如果先手面对的局面是 $a_1 \oplus a_2 \oplus \ldots \oplus a_n = 0$ 那么无论先手怎么拿，都会将局面变成 $a_1 \oplus a_2 \oplus \ldots \oplus a_n \neq 0$，那么后手总可以通过拿走某一堆若干个石子，将局面变成 $a_1 \oplus a_2 \oplus \ldots \oplus a_n = 0$。如此重复，最后一定是先手面临最终没有石子可拿的状态。先手必败。

SG函数

有向图游戏

给定一个有向无环图，图中有一个唯一的起点，在起点上放有一枚棋子。两名玩家交替地把这枚棋子沿有向边进行移动，每次可以移动一步，无法移动者判负。该游戏被称为**有向图游戏**。

任何一个公平组合游戏都可以转化为有向图游戏。具体方法是，把每个局面看成图中的一个节点，并且从每个局面向沿着合法行动能够到达的下一个局面连有向边。

Mex 运算

设 S 表示一个非负整数集合。定义 $\operatorname{mex}(S)$ 为求出不属于集合 S 的最小非负整数的运算，即：

$$\operatorname{mex}(S) = \min_{x \in \mathbb{N}, x \notin S} \{x\}$$

SG 函数

在有向图游戏中，对于每个节点 x，设从 x 出发共有 k 条有向边，分别到达节点 $y_1, y_2, \cdots, y_k$，定义 $\operatorname{SG}(x)$ 为 x 的后继节点 $y_1, y_2, \cdots, y_k$ 的 SG 函数值构成的集合再执行 mex 运算的结果，即：

$$\operatorname{SG}(x) = \operatorname{mex}(\{\operatorname{SG}(y_1), \operatorname{SG}(y_2), \cdots, \operatorname{SG}(y_k)\})$$

特别地，整个有向图游戏 G 的 SG 函数值被定义为有向图游戏起点 s 的 SG 函数值，即 $\operatorname{SG}(G) = \operatorname{SG}(s)$。

有向图游戏的和

设 $G_1, G_2, \cdots, G_m$ 是 m 个有向图游戏。定义有向图游戏 G，它的行动规则是任选某个有向图游戏 G_i，并在 G_i 上行动一步。G 被称为有向图游戏 $G_1, G_2, \cdots, G_m$ 的和。

有向图游戏的和的 SG 函数值等于它包含的各个子游戏 SG 函数值的异或和，即：

$$\operatorname{SG}(G) = \operatorname{SG}(G_1) \operatorname{xor} \operatorname{SG}(G_2) \operatorname{xor} \cdots \operatorname{xor} \operatorname{SG}(G_m)$$

定理

有向图游戏的某个局面必胜，当且仅当该局面对应节点的 SG 函数值大于 0。

有向图游戏的某个局面必败，当且仅当该局面对应节点的 SG 函数值等于 0。

我们不再详细证明该定理。读者可以这样理解：

在一个没有出边的节点上，棋子不能移动，它的 SG 值为 0，对应必败局面。

若一个节点的某个后继节点 SG 值为 0，在 mex 运算后，该节点的 SG 值大于 0。这等价于，若一个局面的后继局面中存在必败局面，则当前局面为必胜局面。

若一个节点的后继节点 SG 值均不为 0，在 mex 运算后，该节点的 SG 值为 0。这等价于，若一个局面的后继局面全部为必胜局面，则当前局面为必败局面。

对于若干个有向图游戏的和，其证明方法与 NIM 博弈类似。

相关代码：

```
const int N = 110,M = 10010;

int n,m;
int s[N],f[M];

int sg(int x)
{
    if(f[x]!=-1)
        return f[x];
```

```
    unordered_set<int> S;
    for(int i=0;i<m;i++){
        int sum = s[i];
        if(x>=sum)
            S.insert(sg(x-sum));
    }

    for(int i=0;;i++){
        if(!S.count(i))
            return f[x] = i;
    }
}

int main()
{
    cin >> m;
    for(int i=0;i<m;i++)
        cin >> s[i];

    memset(f,-1,sizeof f);

    cin >> n;
    int res = 0;
    for(int i=0;i<n;i++){
        int x;
        cin >> x;
        res ^= sg(x);
    }
    if(res)
        puts("Yes");
    else
        puts("No");

    return 0;
}
```

常见结论

不平等博弈

多项式

FFT/NTT

拉格朗日插值

生成函数

多项式全家桶

集合幂级数

FWT/FMT

群论

置换

Burnside引理

Polya定理

线性规划

定理相关

伯特兰—切比雪夫定理

伯特兰—切比雪夫定理说明：若整数$n > 3$，则至少存在一个质数p，符合$n < p < 2n - 2$。另一个稍弱说法是：对于所有大于1的整数n，至少存在一个质数p，符合$n < p < 2n$。