__int128_t

```
#define int128 int128

inline int128 read() {
  int128 x = 0, f = 1;
  char ch = getchar();

  while(!isdigit(ch)) {
    if(ch == '-') f = -1;
    ch = getchar();
  }

  while(isdigit(ch)) {
    x = x * 10 + ch - '0';
    ch = getchar();
  }

  return x * f;
}

inline void print(int128 x) {
  if(x < 0) { putchar('-'); x = -x; }
  if(x > 9) print(x / 10);
  putchar(x % 10 | 48);
}

inline int128 Sqrt(int128 x) {
  int128 l = 0, r = (int128)10000000000000;
  while(l < r) {
    int128 mid = (l + r + 1) / 2;
    if(mid * mid > x) r = mid - 1;
    else l = mid;
  }
  return l;
```

```
}
```

高斯消元

列主元消去求解

```
//列主元消去求解
//序数矩阵a[n][n+1], 存放在ca[][n列+1]
//返回是否有唯一解若有解在,x中[]
int gauss(double a[][N], int n, double x[]) {
    int i, j, k, p;
    for (j = 0; j < n; ++j) {
        for (i = j + 1, p = j; i < n; ++i)
            if (fabs(a[i][j]) > fabs(a[p][j]))
                p = i;
        if (fabs(a[p][j]) < eps) return 0;
        if (p != j)
            for (k = j; k <= n; ++k)
                swap(a[j][k], a[p][k]);
        for (i = j + 1; i < n; ++i)
            for (k = n; k >= j; --k)
                a[i][k] -= a[j][k] * a[i][j] / a[j]
[j];
    }
    for (j = n - 1; j >= 0; --j) {
        x[j] = a[j][n] / a[j][j];
        for (i = j - 1; i >= 0; --i)
            a[i][n] -= a[i][j] * x[j];
    }
    return 1;
}
```

全主元消去解

精度更高，但常数略高。

```
//全主元消去解
//a[n][n]存的是左边的系数，b[n]中存的是右边系数
//返回是否有唯一解若有解在,b中[]
int gauss(double a[][N],int n, double b[]) {
    int i, j, k, row, col, index[N];
    double maxp, t;
    for (i = 0; i < n; i++) index[i] = i;
    for (k = 0; k < n; k++) {
        for (maxp = 0, i = k; i < n; i++)
            for (j = k; j < n; j++)
                if (fabs(a[i][j]) > fabs(maxp))
                    maxp = a[row = i][col = j];
        if (fabs(maxp) < eps) return 0;
        if (col != k) {
            for (i = 0; i < n; i++)
                t = a[i][col], a[i][col] = a[i][k], a[i][k] = t;
            j = index[col], index[col] = index[k], index[k] = j;
        }
        if (row != k) {
            for (j = k; j < n; j++)
                t = a[k][j], a[k][j] = a[row][j], a[row][j] = t;
            t = b[k], b[k] = b[row], b[row] = t;
        }
        for (j = k + 1; j < n; j++) {
            a[k][j] /= maxp;
            for (i = k + 1; i < n; i++)
                a[i][j] -= a[i][k] * a[k][j];
        }
        b[k] /= maxp;
        for (i = k + 1; i < n; i++) b[i] -= b[k] * a[i][k];
    }
    for (i = n - 1; i >= 0; i--)
        for (j = i + 1; j < n; j++)
            b[i] -= a[i][j] * b[j];
```

```
    for (k = 0; k < n; k++) a[0][index[k]] = b[k];
    for (k = 0; k < n; k++) b[k] = a[0][k];
    return 1;
}
```

高斯消元求行列式

```
int guess(int p[][N],int n)
{ // 高斯消元 求 行列式
    int ans = 1;
    for(int i = 1; i <= n; i++) {
        for(int j = i; j <= n; j++)
            if (p[j][i]) {              //对角线的部分不能为0
                for(int k = i; k <= n; k++) swap(p[i][k], p[j][k]);
                if (i != j) ans = -ans;     //交换两行,行列式正负改变
                break;
            }

        // 用第 i 行去修改第 j 行
        // p[j][k] = p[j][k] - p[i][k] * p[j][i] / p[i][i];
        for (int j = i + 1, invf = inv(p[i][i]); j <= n; ++j) {
            int t = p[j][i] * invf % mod;
            for(int k = n; k >= i; k--) p[j][k] = ((p[j][k] - p[i][k] * t % mod) % mod + mod) % mod;
        }

        // 行列式的值就是化成上三角后主对角线的积乘上已经提取出来的数字
        ans = (ans * p[i][i] % mod + mod) % mod;
    }
    return ans;
}
```

复数

```
struct Complex{
    double x, y;
    Complex operator+(const Complex &b) const {
        return Complex({x+b.x, y+b.y});
    }
    Complex operator-(const Complex &b) const {
        return Complex({x-b.x, y-b.y});
    }
    Complex operator*(const Complex &b) const {
        return Complex({x*b.x - y*b.y, x*b.y + 
y*b.x});
    }
    Complex operator/(const Complex &b) const {
        return Complex({(x*b.x + y*b.y)/(b.x*b.x + 
b.y*b.y), (y*b.x - x*b.y)/(b.x*b.x + b.y*b.y)});
    }
};
```

MOD

```
template<int T>
struct ModInt {
    const static int mod = T;
    int x;
    ModInt(int x = 0) : x(x % mod) {}
    int val() { return x; }
    ModInt operator + (const ModInt &a) const { int x0 
= x + a.x; if (x0 >= mod) x0 -= mod; if (x0 < 0) x0 += 
mod; return ModInt(x0); }
```

```
    ModInt operator - (const ModInt &a) const { int x0
= x - a.x; if (x0 >= mod) x0 -= mod; if (x0 < 0) x0 +=
mod; return ModInt(x0); }
    ModInt operator * (const ModInt &a) const { return
ModInt(1LL * x * a.x % mod); }
    ModInt operator / (const ModInt &a) const { return
*this * a.inv(); }
    void operator += (const ModInt &a) { x += a.x; if
(x >= mod) x -= mod; if (x < 0) x += mod;}
    void operator -= (const ModInt &a) { x -= a.x; if
(x < 0) x += mod; if (x >= mod) x -= mod;}
    void operator *= (const ModInt &a) { x = 1LL * x *
a.x % mod; }
    void operator /= (const ModInt &a) { *this = *this
/ a; }
    friend ostream &operator<<(ostream &os, const
ModInt &a) { return os << a.x;}

    ModInt pow(int n) const {
        ModInt res(1), mul(x);
        while(n){
            if (n & 1) res *= mul;
            mul *= mul;
            n >>= 1;
        }
        return res;
    }

    ModInt inv() const {
        int a = x, b = mod, u = 1, v = 0;
        while (b) {
            int t = a / b;
            a -= t * b; swap(a, b);
            u -= t * v; swap(u, v);
        }
        if (u < 0) u += mod;
        return u;
    }
```

```
};
typedef ModInt<mod> mint;
```

积性函数

积性函数满足$f(n)=\prod f(p_i^{e_i})$

积性函数`f`被所有的p^e处的值所决定的

常见的积性函数

$$id(x)=1$$
$$I(x)=x$$
$$e(x)=\begin{cases}1 & x=1\\ 0 & x!=1\end{cases}$$
$$\phi(x)=n\prod_{p|n}(1-\frac{1}{p})$$
$$d(n)因子个数$$
$$d(p^e)=e+1$$
$$\sigma(n)因子和$$
$$\sigma(p^e)=\frac{p_{e+1}-1}{p-1}$$
$$\mu(p^e)=\begin{cases}1 & e=0\\ -1 & e=1\\ 0 & e\geq 2\end{cases}$$

完全积性函数$f(n)=\prod f(p_i)^{e_i}$

常见的完全积性函数

$$id(x)=1$$
$$1(x)=x$$
$$e(x)=\begin{cases}1 & x=1\\ 0 & x!=1\end{cases}$$

积性函数，最重要的就是最小质因子和最小质因子的幂次和，以此我们可推出从p^{e-1}到p^e的式子

有些比较特殊的就不用求`p`与`pe`了

线性筛

```
int primes[N],pe[N],p[N],cnt;
//pe[n]:n最小质因子的幂次和, p[n]:n对应的最小质因子
bool st[N];
int n;

void init()
{
    st[1] = 1;
    for(int i=2;i<=n;i++)
    {
        if(!st[i])
        {
            primes[++cnt] = i;
            p[i] = pe[i] = i;
        }
        for(int j=1;j<=cnt&&primes[j]*i<=n;j++)
        {
            st[primes[j]*i] = 1;
            p[primes[j]*i] = primes[j];
            if(i%primes[j]==0)
            {
                pe[i*primes[j]] = pe[i]*primes[j];
                break;
            }
            pe[i*primes[j]] = primes[j];
        }
    }
}
```

线性求约数之和

```
void get_sig()
{
    sig[1] = 1;
    for(int i=2;i<=n;i++)
        if(i==pe[i]) sig[i] = sig[i/p[i]] + i;
        else sig[i] = sig[i/pe[i]]*sig[pe[i]];
}
```

线性求约数个数

```
void get_d()
{
    d[1] = 1;
    for(int i=2;i<=n;i++)
        if(i==pe[i]) d[i] = d[i/p[i]] + 1;
        else d[i] = d[pe[i]]*d[i/pe[i]];
}
```

线性求欧拉函数

```
void get_phi()
{
    phi[1] = 1;
    for(int i=2;i<=n;i++)
        if(i==pe[i]) phi[i] = i/p[i]*(p[i]-1);
        else phi[i] = phi[i/pe[i]]*phi[pe[i]];
}
```

线性求i^n(n给定)

```
mint qpow[N];
int primes[N],cnt;
bool st[N];

void init()
```

```
{
    qpow[1] = 1;
    for(int i=2;i<N;i++)
    {
        if(!st[i])
        {
            primes[cnt++] = i;
            qpow[i] = mint(i).pow(n);
        }
        for(int j=0;primes[j]*i<N;j++)
        {
            st[primes[j]*i] = 1;
            qpow[primes[j]*i] =
qpow[primes[j]]*qpow[i];
            if(i%primes[j]==0) break;
        }
    }
}
```

莫比乌斯函数及反演

整数分块函数

```
int g(int x,int l)//边界为x，该块边界为l
{
    return x/(x/l);
}
```

线性求莫比乌斯函数

```
int primes[N],cnt,mu[N],sum[N];
bool st[N];

void init(int x)
{
    mu[1] = 1;
    for(int i=2;i<=x;i++)
```

```
    {
        if(!st[i]) primes[cnt++] = i,mu[i] = -1;
        for(int j=0;primes[j]*i<=x;j++)
        {
            st[primes[j]*i] = 1;
            if(i%primes[j]==0) break;
            mu[primes[j]*i] = -mu[i];
        }
    }
    for(int i=1;i<=x;i++) sum[i] = sum[i-1] + mu[i];
}
```

莫比乌斯反演

满足式子。

$$f = g * 1 \ \Leftrightarrow g = f * \mu$$

经典应用

主要展示一下推导过程。

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

这种类型的，我们的统一推导过程为。

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

首先，找到一个这样的式子。

$$f = 1 * g$$

等价转化为。

$$g = f * \mu$$

即为。

$$f(n)=\sum_{d|n} g(d)$$

带入整理。

$$\sum_{1\le d\le n} g(d)\left\lfloor \frac{n}{d}\right\rfloor \left\lfloor \frac{m}{d}\right\rfloor$$

所以，我们关键就是找，给的那个式子和莫比乌斯函数卷出来是个什么。

接下来就是看看，我们利用的卷积式子。

互质数对

利用$\mu = e*\mu$

求$\sum_{1\le i\le n}\sum_{1\le j\le m}[(i,j)=1]$

$$\sum_{1\le i\le n}\sum_{1\le j\le m} e((i,j))$$

又由

$$e=1*\mu$$

则，我们可以得到

$$\sum_{1\le i\le n}\sum_{1\le j\le m}\sum_{d|(i,j)}\mu(d)$$

同时因为`d|(i,j)`等价于`d|i且d|j`，则式子再次变化。我们将`i,j`独立开来了。

$$\sum_{1\le i\le n}\sum_{1\le j\le m}\sum_{d|i,d|j}\mu(d)$$

我们变换一下前后顺序。

$$\sum_{1\le d\le n}\mu(d)\sum_{1\le i\le n,d|i}\ \sum_{1\le j\le m,d|j} 1$$

后半部分，即为求`[1,n]`中`d`的倍数个数与`[1,m]`中`d`的倍数个数的乘积。则式子变为。

$$\sum_{1\le d\le n}\mu(d)\left\lfloor\frac{n}{d}\right\rfloor\left\lfloor\frac{m}{d}\right\rfloor$$

gcd之和

利用$\phi=id*\mu$

求$\sum_{1\le i\le n}\sum_{1\le j\le m}id((i,j))$

带入通式。

$$\sum_{1\le d\le n}\phi(d)\left\lfloor\frac{n}{d}\right\rfloor\left\lfloor\frac{m}{d}\right\rfloor$$

互质集合

问**{1,2,...,n}**中有多少个非空子集，满足这些数的最大公约数是1。输出答案对10^9+7取模的结果。

式子即为。

$\sum_{S\in\{1,2,\dots,n\}}[gcd(S)=1]$

我们按套路改变一下式子。

$$\sum_{S\in\{1,2,\dots,n\}}\sum_{d|S}\mu(d)$$

即为，我们改变一下顺序，则对于一个`d`，我们要找的是对于其所有倍数构成的非空子集的个数。

$$\sum_{d=1}^{n}\mu(d)(2^{\left\lfloor\frac{n}{d}\right\rfloor}-1)$$

LCMSUM2

此题，为手推出积性函数，然后用线性筛求出。

求$(\sum_{i=1}^{n}\sum_{j=1}^{m}[i,j])\ mod\ 2^{60}$

第一步，我们想看到`gcd`

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{ij}{gcd(i,j)}$$

我们设$f(x)=\frac{1}{x}$

则式子变为。

$$\sum_{i=1}^{n}\sum_{j=1}^{m}ijf(i,j)$$

我们假设`g(x)`为`f(x)`的莫反函数。

则式子可以变为。

$$\sum_{d=1}^{n}g(d)\sum_{1\le i\le n,d|i}\sum_{1\le j\le m,d|j}ij$$

此时`i,j`独立，因此我们考虑后半部分求得就是，所有是`d`的倍数的所有i的和与所有`j`的和。

式子则变化为。

$$\sum_{d=1}^{n}g(d)d^2\frac{(\left\lfloor\frac{n}{d}\right\rfloor+1)\left\lfloor\frac{n}{d}\right\rfloor(\left\lfloor\frac{m}{d}\right\rfloor+1)\left\lfloor\frac{m}{d}\right\rfloor}{4}$$

则此时我们设一个新的函数$h(x)=g(x)x^2$

我们知道$g(x),x^2$是积性函数，因此$h(x)$也是积性函数。

我们来推一下式子。

$$g(p^e)=\frac{1}{p^e}-\frac{1}{p^{e-1}}$$

则

$$h(p^e)=p^e-p^{e+1}=p^e(1-p)$$

我们可以直接类推得到。

$$h(n)=n\prod_{p|n}(1-p)$$

也可以变为

$$h(p^e)=p^e(1-p)=-\phi(p^{e+1})$$

$$h(n)=-n\prod_{p|n}p\prod_{p|n}p(p-1)=n\prod 1-p$$

但其实，我们有时不用真的求出具体的。比如该式子。

因为只要得到最小素数幂的递推式子，直接计算即可。

LCMSUM1

求$\sum_{i=1}^{n}[i,n]$

像上一题，我们设$f(n)=\frac{1}{x}$，$g(x)$为$f(x)$的卷积函数。

$$n\sum_{d|n}g(d)\sum_{1\ n中\ d的倍数}\frac{d\frac{n}{d}(\frac{n}{d}+1)}{2}$$

$$\frac{n^2}{2}\sum_{d|n}g(d)(\frac{n}{d}+1)$$

其实就可以写了，但是时间还是卡的比较紧，因为题目是
$T\leq 10^5,n\leq 10^7$

我们考虑调整。

$$\frac{n}{2}+(\frac{n^2}{2}\sum_{d|n}g(d)\frac{n}{d})$$

我们只求后面的部分。

$$\frac{n^2}{2}\sum_{d|n}g(d)\frac{n}{d}$$

$$\frac{n^3}{2}\sum_{d|n}g(d)/d$$

$$g(p^e)=\frac{1}{p^e}-\frac{1}{p^{e-1}}$$

$$g(p^e)/p^e=\frac{1}{p^{2e}}-\frac{1}{p^{2e-1}}$$

$$h(p^e)=p^{2e}(\sum_{d|p^e}g(d)/d)+p^{2e}-p^{2e-1}+p^{2e-2}-\ldots+1$$
$$=h(p^{e-1})+p^{2e}-p^{2e-1}$$

因此原式改为

$$\frac{n}{2}(1+h(n))$$

我们就可以O(1)的求出答案。

Dirichlet卷积快速幂

形式为，$h(n)=\sum_{d|n}f(d)g(\frac{n}{d})$

一些性质

1. **交换律**，$f*g=g*f$
2. **结合律**，$(f*g)*k=g*(f*k)$
3. **当f与g都为积性函数时，$f*g$也为积性函数**

$O(nlogn)$

```
struct Dirichlet
{
    int h[N],n;
    Dirichlet(int _n):n(_n)
    {
```

```
        memset(h,0,sizeof h);
    }
    Dirichlet operator*(const Dirichlet& b) const
    {
        Dirichlet c(n);
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n/i;j++)
                c.h[i*j] = (1ll*c.h[i*j] + 
1ll*h[i]*b.h[j]%mod)%mod;
        return c;
    }
};

Dirichlet ksm_Dirichlet(Dirichlet a,int k)
{
    Dirichlet res(n);res.h[1] = 1;
    while(k)
    {
        if(k&1) res = res*a;
        a = a*a;
        k>>=1;
    }
    return res;
}
```

拉格朗日插值

给n+1个点求一个n次多项式f(x)以及求f(k)

$O(n^2)$

```
LL ksm(LL a,LL b)
{
    LL res = 1;
    while(b)
    {
        if(b&1) res = 1ll*res*a%mod;
        a = 1ll*a*a%mod;
```

```
        b>>=1;
    }
    return res;
}

struct Polylglr
{
    int n;
    LL a[N],b[N],c[N],temp[N],x[N],y[N];
    Polylglr(int _n):n(_n){}
    void mul(LL *f, int len, LL t) { //len为多项式的次数+1，函数让多项式f变成f*(x+t)
            for (int i = len; i > 0; --i)
                temp[i] = f[i], f[i] = f[i - 1];
            temp[0] = f[0], f[0] = 0;
            for (int i = 0; i <= len; ++i)
                f[i] = (f[i] + t * temp[i]) % mod;
        }
        void dev(LL *f, LL *r, LL t) { //f是被除多项式的系数，r保存f除以x+t的结果
        for (int i = 0; i <= n; ++i)
            temp[i] = f[i];
        for (int i = n; i > 0; --i) {
            r[i - 1] = temp[i];
            temp[i - 1] = (temp[i - 1] - t * temp[i]) % mod;
        }
        return;
    }
    void lglr()
    {
        memset(a, 0, sizeof a);
        b[1] = 1, b[0] = -x[1];
        for (int i = 2; i <= n; ++i) {
            mul(b, i, -x[i]);
        }//预处理(x-x1)*(x-x2)...*(x-xn)
        for (int i = 1; i <= n; ++i) {
            LL fz = 1;
```

```
            for (int j = 1; j <= n; ++j) {
                if (j == i)
                    continue;
                fz = fz * (x[i] - x[j]) % mod;
            }
            fz = ksm(fz, mod - 2);
            fz = fz * y[i] % mod; //得到多项式系数
            dev(b, c, -x[i]);//得到多项式，保存在b数组
            for (int j = 0; j < n; ++j)
                a[j] = (a[j] + fz * c[j]) % mod;
        }
    }
    LL f(LL k)
    {
        LL ans = 0,res = 1;
        for(int i=0;i<n;i++)
        {
            ans = (1ll*ans + 1ll*res*a[i]%mod)%mod;
            res = 1ll*res*k%mod;
        }
        ans = (1ll*ans + mod)%mod;
        return ans;
    }
};
```

自然数k次幂的和

拉格朗日插值

时间复杂度$O(n)$

```
#include <bits/stdc++.h>
using namespace std;

const int N = 1100007, mod = 1e9 + 7;
```

```
int n, k;
int s[N], pre[N], suf[N], infact[N], fact[N];

int qpow(int a, int b)
{
    int res = 1;
    while(b) {
        if(b & 1) res = 1ll * res * a % mod;
        a = 1ll * a * a % mod;
        b >>= 1;
    }
    return res;
}

void init()
{
    infact[0] = fact[0] = 1;
    for(int i = 1; i <= k + 10; ++ i)
        fact[i] = 1ll * fact[i - 1] * i % mod;
    infact[k + 10] = qpow(fact[k + 10], mod - 2);
    for(int i = k + 9; i >= 0; -- i)
        infact[i] = 1ll * infact[i + 1] * (i + 1) %
mod;
    infact[0] = 1;
}

int f(int n,int k)
{
    for(int i = 1; i <= k + 2; ++ i)
        s[i] = (s[i - 1] + qpow(i, k)) % mod;
    if(n<=k+2) return s[n];
    int ans = 0;
    pre[0] = 1;
    for(int i = 1; i <= k + 2; ++ i)
        pre[i] = 1ll * pre[i - 1] * (n - i) % mod;
    suf[k + 3] = 1;
    for(int i = k + 2; i; -- i)
        suf[i] = 1ll * suf[i + 1] * (n - i) % mod;
```

```
    for(int i = 1; i <= k + 2; ++ i) {
        s[i] = 1ll * s[i] * pre[i - 1] % mod * suf[i + 1] % mod * infact[i - 1] % mod * infact[k + 2 - i] % mod;
        if((k + 2 - i) & 1)
            ans = (1ll * ans - s[i] + mod) % mod;
        else ans = (1ll * ans + s[i]) % mod;
    }
    return ans;
}

int main()
{
    init();
    scanf("%d%d", &n, &k);
    printf("%d\n", f(n,k));
}
```

FFT

```
const double PI = acos(-1);
struct Complex
{
    double x,y;
    Complex(double x = 0,double y = 0) : x(x), y(y) {}
};

//复数乘法：模长相乘，幅度相加
Complex operator * (Complex J, Complex Q) {return Complex(J.x * Q.x - J.y * Q.y, J.x * Q.y + J.y * Q.x);}
Complex operator - (Complex J, Complex Q) {return Complex(J.x - Q.x, J.y - Q.y);}
```

```
Complex operator + (Complex J, Complex Q) {return
Complex(J.x + Q.x, J.y + Q.y);}

namespace FFT
{
    typedef vector<Complex> poly;
    int R[N],L,limit=1;//二进制翻转 二进制位数 补齐的2的整数
幂N
    int FFT_init(int n)
    {
        while(limit<=n) limit <<= 1,L ++ ;
        // 补成2的整次幂，也就是N
        for(int i = 0; i < limit; ++ i)
            R[i] = (R[i >> 1] >> 1) | ((i & 1) << (L -
1));
        return limit;
    }
    void FFT(poly &A, int type,int limit)
    {
        A.resize(limit);
        for(int i = 0; i < limit; ++ i)
            if(i < R[i])
                swap(A[i], A[R[i]]);
            //i小于R[i]时才交换，防止同一个元素交换两次，回到
它原来的位置。

        //从底层往上合并
        for(int mid = 1; mid < limit; mid <<= 1) {
            //待合并区间长度的一半，最开始是两个长度为1的序列
合并,mid = 1;
            Complex wn(cos(PI / mid), type * sin(PI /
mid));//单位根w_n^1;

            for(int len = mid << 1, pos = 0; pos <
limit; pos += len) {
                //len是区间的长度，pos是当前的位置,也就是合
并到了哪一位
```

```
                Complex w(1, 0);//幂,一直乘，得到平方，三次方...

                for(int k = 0; k < mid; ++ k, w = w * wn) {
                    //只扫左半部分，蝴蝶变换得到右半部分的答案,w 为 w_n^k
                    Complex x = A[pos + k];//左半部分
                    Complex y = w * A[pos + mid + k];//右半部分
                    A[pos + k] = x + y;//左边加
                    A[pos + mid + k] = x - y;//右边减
                }
            }
        }
        if(type == 1) return ;
        for(int i = 0; i <= limit; ++ i)
            A[i].x /= limit;
            //最后要除以limit也就是补成了2的整数幂的那个N，将点值转换为系数
            //（前面推过了点值与系数之间相除是N）
    }
    poly poly_mul(poly A,poly B)
    {
        int deg = A.size() + B.size() - 1;
        int limit = FFT_init(deg);
        poly C(limit);
        FFT(A,1,limit),FFT(B,1,limit);
        for(int i=0;i<limit;++i) C[i] = A[i]*B[i];
        FFT(C,-1,limit);
        C.resize(deg);
        return C;
    }
};
```

阶与原根

阶：满足$a^x \equiv 1(mod\ m)$的最小x，gcd(a,m)=1

```
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false); cin.tie(0),
cout.tie(0)
using namespace std;
using ll = long long;

ll q[200],t;

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

int main()
{
    ios;
    int p,T;cin>>p>>T;
    int m = p - 1;
    for(int i=2;i<=m/i;i++)
        if(m%i==0)
        {
            q[t++] = i;
            while(m%i==0) m /= i;
        }
    if(m>1) q[t++] = m;
    while(T--)
    {
        ll a;cin>>a;
        ll d = p - 1;
```

```
        for(int i=0;i<t;i++)
            while(d%q[i]==0&&ksm(a,d/q[i],p)==1)
                d/=q[i];
        cout<<d<<'\n';
    }
    return 0;
}
```

原根

满足$g^{\phi(m)} \equiv 1(mod\ m)$**的g的值**

$g^0, g^1, \ldots, g^{\phi(m)-1}$构成了模m的简化剩余系。

只有$1, 2, 4, p^a, 2p^a$存在原根，其中`p`为奇素数。

NTT多项式

```
typedef long long ll;

const int N = 3000007;
const int mod = 998244353, gg = 3, ig = 332738118, img
= 86583718;

int qpow(int a, int b)
{
    int res = 1;
    while(b) {
        if(b & 1) res = 1ll * res * a % mod;
        a = 1ll * a * a % mod;
        b >>= 1;
    }
    return res;
}

namespace Poly
{
```

```
    #define mul(x, y) (1ll * x * y >= mod ? 1ll * x *
y % mod : 1ll * x * y)
    #define minus(x, y) (1ll * x - y < 0 ? 1ll * x - y
+ mod : 1ll * x - y)
    #define plus(x, y) (1ll * x + y >= mod ? 1ll * x +
y - mod : 1ll * x + y)
    #define ck(x) (x >= mod ? x - mod : x)//取模运算太慢
了

    typedef vector<int> poly;
    const int G = 3;//根据具体的模数而定，原根可不一定不一
样！！！
    //一般模数的原根为 2 3 5 7 10 6
    const int inv_G = qpow(G, mod - 2);
    int RR[N], deer[2][19][N], inv[N];
    //这个地方的deer第二维是根据最大项的次数定的，比如n =
1e5，则2^17>1e5，初始化的时候，我们多初始化一倍，则init里为
18，第二维为18+1，第三维为大于2^18的数字
    void init(const int t) {//预处理出来NTT里需要的w和wn，
砍掉了一个log的时间
        for(int p = 1; p <= t; ++ p) {
            int buf1 = qpow(G, (mod - 1) / (1 << p));
            int buf0 = qpow(inv_G, (mod - 1) / (1 <<
p));
            deer[0][p][0] = deer[1][p][0] = 1;
            for(int i = 1; i < (1 << p); ++ i) {
                deer[0][p][i] = 1ll * deer[0][p][i -
1] * buf0 % mod;//逆
                deer[1][p][i] = 1ll * deer[1][p][i -
1] * buf1 % mod;
            }
        }
        inv[1] = 1;
        for(int i = 2; i <= (1 << t); ++ i)
            inv[i] = 1ll * inv[mod % i] * (mod - mod /
i) % mod;
    }
```

```
    int NTT_init(int n) {//快速数论变换预处理
        int limit = 1, L = 0;
        while(limit <= n) limit <<= 1, L ++ ;
        for(int i = 0; i < limit; ++ i)
            RR[i] = (RR[i >> 1] >> 1) | ((i & 1) << (L - 1));
        return limit;
    }

    void NTT(poly &A, int type, int limit) {//快速数论变换
        A.resize(limit);
        for(int i = 0; i < limit; ++ i)
            if(i < RR[i])
                swap(A[i], A[RR[i]]);
        for(int mid = 2, j = 1; mid <= limit; mid <<= 1, ++ j) {
            int len = mid >> 1;
            for(int pos = 0; pos < limit; pos += mid) {
                int *wn = deer[type][j];
                for(int i = pos; i < pos + len; ++ i, ++ wn) {
                    int tmp = 1ll * (*wn) * A[i + len] % mod;
                    A[i + len] = ck(A[i] - tmp + mod);
                    A[i] = ck(A[i] + tmp);
                }
            }
        }
        if(type == 0) {
            for(int i = 0; i < limit; ++ i)
                A[i] = 1ll * A[i] * inv[limit] % mod;
        }
    }

    poly poly_mul(poly A, poly B) {//多项式乘法
        int deg = A.size() + B.size() - 1;
```

```
        int limit = NTT_init(deg);
        poly C(limit);
        NTT(A, 1, limit);
        NTT(B, 1, limit);
        for(int i = 0; i < limit; ++ i)
            C[i] = 1ll * A[i] * B[i] % mod;
        NTT(C, 0, limit);
        C.resize(deg);
        return C;
    }

    poly poly_inv(poly &f, int deg) {//多项式求逆
        if(deg == 1)
            return poly(1, qpow(f[0], mod - 2));

        poly A(f.begin(), f.begin() + deg);
        poly B = poly_inv(f, (deg + 1) >> 1);
        int limit = NTT_init(deg << 1);
        NTT(A, 1, limit), NTT(B, 1, limit);
        for(int i = 0; i < limit; ++ i)
            A[i] = B[i] * (2 - 1ll * A[i] * B[i] % mod
+ mod) % mod;
        NTT(A, 0, limit);
        A.resize(deg);
        return A;
    }

    poly poly_dev(poly f) {//多项式求导
        int n = f.size();
        for(int i = 1; i < n; ++ i) f[i - 1] = 1ll *
f[i] * i % mod;
        return f.resize(n - 1), f;//f[0] = 0，这里直接扔
了,从1开始
    }

    poly poly_idev(poly f) {//多项式求积分
        int n = f.size();
```

```
        for(int i = n - 1; i ; -- i) f[i] = 1ll * f[i
- 1] * inv[i] % mod;
        return f[0] = 0, f;
    }

    poly poly_ln(poly f, int deg) {//多项式求对数
        poly A = poly_idev(poly_mul(poly_dev(f),
poly_inv(f, deg)));
        return A.resize(deg), A;
    }

    poly poly_exp(poly &f, int deg) {//多项式求指数
        if(deg == 1)
            return poly(1, 1);

        poly B = poly_exp(f, (deg + 1) >> 1);
        B.resize(deg);
        poly lnB = poly_ln(B, deg);
        for(int i = 0; i < deg; ++ i)
            lnB[i] = ck(f[i] - lnB[i] + mod);

        int limit = NTT_init(deg << 1);//n -> n^2
        NTT(B, 1, limit), NTT(lnB, 1, limit);
        for(int i = 0; i < limit; ++ i)
            B[i] = 1ll * B[i] * (1 + lnB[i]) % mod;
        NTT(B, 0, limit);
        B.resize(deg);
        return B;
    }

    poly poly_sqrt(poly &f, int deg) {//多项式开方
        if(deg == 1) return poly(1, 1);
        poly A(f.begin(), f.begin() + deg);
        poly B = poly_sqrt(f, (deg + 1) >> 1);
        poly IB = poly_inv(B, deg);
        int limit = NTT_init(deg << 1);
        NTT(A, 1, limit), NTT(IB, 1, limit);
        for(int i = 0; i < limit; ++ i)
```

```
            A[i] = 1ll * A[i] * IB[i] % mod;
        NTT(A, 0, limit);
        for(int i =0; i < deg; ++ i)
            A[i] = 1ll * (A[i] + B[i]) * inv[2] % mod;
        A.resize(deg);
        return A;
    }

    poly poly_pow(poly f, int k) {//多项式快速幂
        f = poly_ln(f, f.size());
        for(auto &x : f) x = 1ll * x * k % mod;
        return poly_exp(f, f.size());
    }

    poly poly_cos(poly f, int deg) {//多项式三角函数
（cos）
        poly A(f.begin(), f.begin() + deg);
        poly B(deg), C(deg);
        for(int i = 0; i < deg; ++ i)
            A[i] = 1ll * A[i] * img % mod;

        B = poly_exp(A, deg);
        C = poly_inv(B, deg);
        int inv2 = qpow(2, mod - 2);
        for(int i = 0; i < deg; ++ i)
            A[i] = 1ll * (1ll * B[i] + C[i]) % mod *
inv2 % mod;
        return A;
    }

    poly poly_sin(poly f, int deg) {//多项式三角函数
（sin）
        poly A(f.begin(), f.begin() + deg);
        poly B(deg), C(deg);
        for(int i = 0; i < deg; ++ i)
            A[i] = 1ll * A[i] * img % mod;

        B = poly_exp(A, deg);
```

```
        C = poly_inv(B, deg);
        int inv2i = qpow(img << 1, mod - 2);
        for(int i = 0; i < deg; ++ i)
            A[i] = 1ll * (1ll * B[i] - C[i] + mod) % mod * inv2i % mod;
        return A;
    }

    poly poly_arcsin(poly f, int deg) {
        poly A(f.size()), B(f.size()), C(f.size());
        A = poly_dev(f);
        B = poly_mul(f, f);
        for(int i = 0; i < deg; ++ i)
            B[i] = minus(mod, B[i]);
        B[0] = plus(B[0], 1);
        C = poly_sqrt(B, deg);
        C = poly_inv(C, deg);
        C = poly_mul(A, C);
        C = poly_idev(C);
        return C;
    }

    poly poly_arctan(poly f, int deg) {
        poly A(f.size()), B(f.size()), C(f.size());
        A = poly_dev(f);
        B = poly_mul(f, f);
        B[0] = plus(B[0], 1);
        C = poly_inv(B, deg);
        C = poly_mul(A, C);
        C = poly_idev(C);
        return C;
    }
}

using Poly::poly;
```

普通多项式转下降沿多项式

原理

假设原函数为$f(x)=\sum_{i=0}^{n}a_ix^i$

我们来推导一下。

我们知道

$$x^n=\sum_{i=0}^{n}S(n,i)x^{\underline{i}}$$

因此，当我们带入并交换求和次序时，可以得到。

$$\begin{aligned}\sum_{i=0}^{n}a_ix^i&=\sum_{i=0}^{n}a_i\sum_{j=0}^{i}S(i,j)x^{\underline{j}}\\&=\sum_{i=0}^{n}k^{\underline{i}}\sum_{j=i}^{n}S(j,i)a_j\end{aligned}$$

也就是说

$$b_i=\sum_{j=i}^{n}S(j,i)a_j$$

递推求

$O(n^2)$

```
int s[N][N],a[N],b[N];
int n;

void init(int n)
{
    s[0][0] = 1;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            s[i][j] = (1ll*s[i-1][j-1] + 1ll*j*s[i-1][j]%p)%p;
}
```

```
for(int i=0;i<=n;i++)
    for(int j=i;j<=n;j++)
        b[i] = (1ll*b[i] + 1ll*s[j][i]*a[j]%p)%p;
```

线性基

一般处理区间异或信息

```
bool insert(int x) {
    for(int i = 30; i >= 0; i--) {
        if(x & (1 << i)) {
            if(b[i]) x ^= b[i];
            else {
                b[i]=x;
                return 0;
            }
        }
    }
    return 1;
}
```

EXGCD

```
template <class T>
T exgcd(T a, T b, T& x, T& y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }
    T d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

结论

我们求出的一组解，|x|<=b,|y|<=a

逆元

gcd(a , p) = 1

```
template <class T>
T ksm(T a,T b,T p)
{
    T res = 1;
    while(b)
    {
        if(b&1) res = 1ll*res*a%p;
        a = 1ll*a*a%p;
        b>>=1;
    }
    return res;
}


template <class T>
T get_inv(T a,T p)
```

```
{
    return ksm(a,p-2,p);
}
```

gcd(a , p) = c (c>1)

```
template <class T>
T exgcd(T a, T b, T& x, T& y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }
    T d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

template <class T>
T get_inv(T a,T p)
{
    T x,y;
    T d = exgcd(a,p,x,y);
    return (x%p+p)%p;
}
```

求1~n的逆元

时间复杂度：$O(n)$

原理：$inv[i] = (p - p/i) * inv[p\%i]\%p$

其中$n \leq p-1$且`p为任意数`

```
ll inv[N];
int n,p;

cin>>p>>n;
inv[1] =1;
for(int i=2;i<=n;i++) inv[i] = (p-p/i)*inv[p%i]%p;
```

求$a_1, a_2, ..., a_n$的逆元

时间复杂度：$O(n)$

```
ll s[N],t[N],inv[N],a[N];
ll p,n;

s[0] = 1;
for(int i=1;i<=n;i++)
{
    s[i] = s[i-1];
    if(!a[i]) continue;
    s[i] = s[i]*a[i]%p;
}
t[n] = get_inv(s[n],p);
for(int i=n-1;i;i--) t[i] = t[i+1]*a[i+1]%p;
for(int i=1;i<=n;i++) inv[i] = s[i-1]*t[i]%p;
```

BSGS

求$a^x \equiv b(modp)$的最小整数解x，传入a，p，b

gcd(a , p) = 1

```
int bsgs(int a,int p,int b)
{
    if(1%p==b%p) return 0;
```

```
    int k = sqrt(p) + 1;
    unordered_map<int,int> list;
    for(int i=0,j=b;i<k;i++)
    {
        list[j] = i;
        j = 1ll*j*a%p;
    }
    int ak = 1;
    for(int i=0;i<k;i++) ak = 1ll*ak*a%p;
    for(int i=1,j=ak;i<=k;i++)
    {
        if(list.count(j)) return 1ll*i*k-list[j];
        j = 1ll*j*ak%p;
    }
    return -INF;//表示无解
}
```

gcd(a , p) = c (c>1)

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

int get_inv(int a,int p)
{
    int x,y;
    int d = exgcd(a,p,x,y);
    return x%p;
}
```

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

int bsgs(int a,int p,int b)
{
    if(1%p==b%p) return 0;
    int k = sqrt(p) + 1;
    unordered_map<int,int> list;
    for(int i=0,j=b;i<k;i++)
    {
        list[j] = i;
        j = 1ll*j*a%p;
    }
    int ak = ksm(a,k,p);
    for(int i=1,j=ak;i<=k;i++)
    {
        if(list.count(j)) return 1ll*i*k-list[j];
        j = 1ll*j*ak%p;
    }
    return -INF;//表示无解
}

int exbsgs(int a, int p, int b)
{
    b = (b % p + p) % p;
    if (1 % p == b % p) return 0;
    int x, y;
    int d = exgcd(a, p, x, y);
    if (d > 1)
```

```
    {
        if (b % d) return -INF;
        int inv = get_inv(a / d, p / d);
        return exbsgs(a, p / d, 1ll * b / d * inv % (p / d)) + 1;
    }
    return bsgs(a, p, b);
}

//无解判断是<-INF/2
```

线性同余方程

image-20220903214503583

常用

我们只求x的非负整数解下的一组解

```
template <class T>
T exgcd(T a, T b, T& x, T& y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }
    T d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

template <class T>
bool get_ans(T &a,T &b,T &c,T &x,T &y)
{
    T d = exgcd(a,b,x,y);
    if(c%d) return 0;
    c/=d,a/=d,b/=d;
```

```
    x = 1ll*x*(c%b)%b;
    if(x<0) x += b;
    y = (c - 1ll*a*x)/b;//这里的式子，根据求的式子的形式决定，看下边吧。
    return 1;
}
```

根据求的式子

求的式子是ax+by=d，我们就正常求。

求的式子是ax-by=d，我们就正常求**ax+by=d**，接下来，但是记住我们求的式子是**ax-by=d**。

则我们的通解其实变成了

$$\begin{cases} x = x_0 + \frac{b}{(a,b)}t \\ y = y_0 - \frac{a}{(a,b)}t \end{cases} t \in Z$$

因此，我们此时求非负整数解的时候，可以先算出y在x是非负整数解的时候，是否是非负整数，不是的话都同时向上加，**x+b**，**y+a**。

求的式子是-ax+by=d，跟上一个同理了。

求的式子是-ax+by=d，跟第一个同理。

完全版

```
template<class T>
struct Linear
{
    T _a,_b,_c,_d;
    T _x0,_y0;//x最小非负整数解，y最小非负整数解
    T _x1,_y1;//在有正整数解下，x最小正整数解，y最小正整数解
    T _x2,_y2;//在有正整数解下，x最大正整数解，y最大正整数解
    T _x3,_y3;//x最小正整数解，y最小正整数解
    T _cnt;
    Linear(T a,T b,T c): _a(a),_b(b),_c(c){}
    T exgcd(T a,T b,T &x,T &y)
```

```
    {
        if(!b)
        {
            x=1,y=0;
            return a;
        }
        T d=exgcd(b,a%b,y,x);
        y-=a/b*x;
        return d;
    }
    /**
     * @brief
     * -1 无解 0 无正整数解 1 有正整数解
     * @return int
     */
    int get()
    {
        T x,y,k;
        _d = exgcd(_a,_b,x,y);
        if(_c%_d!=0) return -1;
        x *= _c/_d,y *= _c/_d;//可能爆long long，注意！
        _a /= _d,_b /= _d;
        _x0 = (x%_b+_b)%_b,_y0 = (y%_a+_a)%_a;
        //此处，我们可以求出当x取非负整数解时，y的值，只需将1换为0
        if(x<0) k = ceil((1.0-x)/_b),x += _b*k,y -= _a*k;
        else k = (x-1)/_b,x -= _b*k,y += _a*k;
        _x3 = x,_y3 = y + _a*(LL)ceil((1.0-y)/_a);
        int t = 0;
        if(y>0)//在x取最小正整数解的时候，若y无正整数解，则直接无正整数解。
        {
            _cnt = (y-1)/_a+1;
            _x1 = x,_y1 = (y-1)%_a+1;
            _x2 = x+(y-1)/_a*_b,_y2 = y;
            t = 1;
        }
```

```
        return t;
    }
};
typedef Linear<LL> Lin;
```

中国剩余定理

$$
\begin{cases}
a \equiv b_0 \qquad (mod\ w_0) \\
a \equiv b_1 \qquad (mod\ w_1) \\
\quad . \\
\quad . \\
\quad . \\
a \equiv b_{k-1} \quad (mod\ w_{k-1})
\end{cases}
$$

其中`w,b`已知,`w[i]>0`且`w[i]`与`w[j]`互质，求`a`.解得范围`[1,n]`，`n = w[0]*w[1]*...w[k-1]`

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

LL crt(LL b[],LL w[],int k)
{
    LL n = 1,a = 0;
    for(int i=0;i<k;i++) n *= w[i];
    for(int i=0;i<k;i++)
    {
```

```
        LL m = n/w[i];LL x,y;
        exgcd(m,w[i],x,y);x = (x%w[i]+w[i])%w[i];
        a = (a + x*m%n*b[i]%n)%n;//注意这个部分，可能会爆
longlong，如果爆了，就把这一块强转成i128
    }
    if(a>0) return a;
    else return (a+n);
}
```

扩展中国剩余定理

$$
\begin{cases}
x \equiv b_0 & (mod\ a_0) \\
x \equiv b_1 & (mod\ a_1) \\
\quad . \\
\quad . \\
\quad . \\
x \equiv b_{k-1} & (mod\ a_{k-1})
\end{cases}
$$

其中`a,b`已知,`a[i]>0`且`a[i]`与`a[j]`可以不互质，求`x`.解得范围`[1,n]`，`n = lcm(a[0],a[1],...,a[k-1])`

```
template <class T>
T exgcd(T a, T b, T& x, T& y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }
    T d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

template <class T>
```

```
T get_inv(T a,T p)
{
    T x,y;
    T d = exgcd(a,p,x,y);
    return (x%p+p)%p;
}

template <class T>
T excrt(T b[],T a[],int n) {
    T x,y,k;
    T M=a[0],ans=b[0];//第一个方程的解特判
    for(int i=1;i<n;i++)
    {
        T bi=M,ai=a[i],c=(b[i]-
ans%ai+ai)%ai;//ax≡c(mod b)
        T d = exgcd(bi,ai,x,y),ag=ai/d;
        if(c%d!=0) return -1; //判断是否无解，然而这题其实
不用

        x=x*c/d%ag;//注意这里可能会爆longlong，如果用这个可
能的话加上i128
        ans+=x*M;//更新前k个方程组的答案
        M*=ag;//M为前k个m的lcm
        ans=(ans%M+M)%M;
    }
    return (ans%M+M)%M;
}
```

应用

我们利用CRT的主要方向是依托于，将对合数取模，转变为对素数幂取模，进而操作。

判断`n`个同余方程是否有解

给定`n`个方程，$x \equiv a_i(mod\ m_i)$，判断方程式不是有解。

$1 \le n \le 50, 0 \le a_i < m_i \le 10^5$。

这个问题不能直接用CRT解，因为这些`m`乘起来会很大。

我们利用上方的思维。

将合数拆为素数幂，则判断是否有解，即变为了，对某个素数而言，其最高幂次所得到的余数，用其去验证其余的幂次的方程的余数是否合法。

```
int n;cin>>n;
map<int,vector<PII>> eqns;
for(int i=0;i<n;i++)
{
    int a,m;cin>>a>>m;
    for(int j=2;j<=m/j;j++)
        if(m%j==0)
        {
            int p = 1;
            while(m%j==0) m/=j,p *= j;
            eqns[j].push_back({p,a%p});
        }
    if(m>1) eqns[m].push_back({m,a%m});
}
for(auto eq:eqns)
{
    auto eqn = eq.second;
    int v = max_element(eqn.begin(),eqn.end()) ->
second;
    for(auto p:eqn)
        if(v%p.first!=p.second)
        {
            cout<<"No\n";
            return ;
        }
}
cout<<"Yes\n";
```

大质数指数判断

```
constexpr int mod = 998244353;
LL mul(LL a, LL b, LL mod) {
    return a * b % mod;
}
LL power(LL a, LL r, LL mod) {
    LL res = 1;
    for (; r; r >>= 1, a = mul(a, a, mod))
        if (r & 1) res = mul(res, a, mod);
    return res;
}
LL p[9] = {2, 3, 5, 7, 11, 13, 17, 19, 23};
bool miller_rabin(LL n){
    if(n == 1) return false;
    if(n == 2) return true;
    if(not(n & 1)) return false;
    LL d = n - 1, r = 0;
    for(; not(d & 1); d >>= 1) r += 1;
    bool res = true;
    for(int i = 0; i < 9 and p[i] < n and res; i += 1)
{
        LL x = power(p[i], d, n);
        if(x == 1 or x == n - 1) continue;
        for(int j = 1; j < r; j += 1){
            x = mul(x, x, n);
            if(x == n - 1) break;
        }
        if(x != n - 1) res = false;
    }
    return res;
};


-- By Heltion
```

```
using i64 = long long;
```

```
i64 mul(i64 a, i64 b, i64 m) {
    return static_cast<__int128>(a) * b % m;
}
i64 power(i64 a, i64 b, i64 m) {
    i64 res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}
bool isprime(i64 n) {
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13,
17, 19, 23};
    int s = __builtin_ctzll(n - 1);
    i64 d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n)
            return true;
        i64 x = power(a, d, n);
        if (x == 1 || x == n - 1)
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}
```

```
-- By Jiangly
```

1~n的逆元

```
infact[1] = 1;
int MOD;
for(int i = 2; i <= n; i ++ ) {
    infact[i] = 1LL * (MOD - MOD / i) * infact[MOD % i] % MOD;
}
```

牛顿迭代法

$a^{\frac{1}{m}}$

```
double Newton(double a, int m){
    if(a==0) return 0;
    double x0=a/2;
    double x1=x0-x0*(1-a*pow(x0,-m))/m;
    do{
        x0=x1;
        x1=x0-x0*(1-a*pow(x0,-m))/m;
    }
    while(fabs(x0-x1)>=1e-6);
    return x0;
}
```

求[L,R]所有的质数（区间筛）

筛出在区间$[L,R]$中的所有素数（$0 \le L,R \le 10^14$，$0 \le R-L \le 10^7$）

时间复杂度：$O(10^7 loglog10^7)$

```
const int N = 1e7 + 10;
ll primes[N/5],cnt;
```

```
bool st[N];

void get_primes(int n)
{
    memset(st,0,sizeof st);
    cnt = 0;
    for(int i=2;i<=n;i++)
    {
        if(!st[i]) primes[cnt++] = i;
        for(int j=0;primes[j]*i<=n;j++)
        {
            st[primes[j]*i] = 1;
            if(i%primes[j]==0) break;
        }
    }
}

int main()
{
    ll l,r;cin>>l>>r>>a>>b;
    get_primes(10000000);
    memset(st,0,sizeof st);
    for(int i=0;i<cnt;i++)
    {
        ll p = primes[i];
        for(ll j=max(2*p,(l+p-1)/p*p);j<=r;j+=p)
            st[j-l] = 1;
    }
    cnt = 0;
    for(ll i=0;i<=r-l;i++)//注意这个区间内可能是有1的，并且1还未被标记，因此要特判一下
        if(!st[i]&&i+l>=2)
            primes[cnt++] = i + l;
    return 0;
}
```

阶乘

阶乘分解

将`n!`拆分为质因数乘积

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
const int N = 1e6 + 10;
int primes[N],cnt;
bool st[N];

void init(int n)
{
    for(int i=2;i<=n;i++)
    {
        if(!st[i]) primes[cnt++] = i;
        for(int j=0;primes[j]*i<=n;j++)
        {
            st[primes[j]*i] = 1;
            if(i%primes[j]==0) break;
        }
    }
}

int main()
{
    int n;cin>>n;
    init(n);
    for(int i=0;i<cnt;i++)
    {
        LL p = primes[i];
        int cnt = 0;
        for(int j=n;j;j/=p) cnt += j/p;
        if(cnt) cout<<primes[i]<<" "<<cnt<<endl;
    }
```

```
    return 0;
}
```

阶乘之乘

求`1!2!...n!`的尾部0个数

$O(n)$
利用连续的性质，我们只需要看看新增的数字会提供多少个新的5即可。

```
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false); cin.tie(0), cout.tie(0)
using namespace std;
typedef long long LL;

int main()
{
    ios;
    int n;cin>>n;
    LL ans = 0,res = 0;
    for(int i=1;i<=n;i++)
    {
        int t = i;
        while(t%5==0)
        {
            res++;
            t/=5;
        }
        ans += res;
    }
    cout<<ans<<endl;
    return 0;
}
```

$O(logn)$

```
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false); cin.tie(0),
cout.tie(0)
using namespace std;
typedef long long LL;

int main()
{
    ios;
    int n;cin>>n;
    LL ans = 0;
    for(int j=5;j<=n;j*=5)
    {
        ans += 1ll*j*(n/j)*(n/j-1)>>1;
        ans += (n/j)*(n%j+1);
    }
    cout<<ans<<endl;
    return 0;
}
```

求组合数

$$C(n,m)\ mod\ p, 1 \le m \le n \le 10^7$$

p为素数

时间复杂度：预处理$O(10^7)$，询问$O(1)$

```
ll fact[N],infact[N];
void init()
{
    fact[0] = infact[0] = 1;
    for(int i=1;i<N;i++) fact[i] = fact[i-1]*i%mod;
    infact[N-1] = get_inv(fact[N-1],mod);
    for(int i=N-2;i;i--) infact[i] = infact[i+1]*
(i+1)%mod;
```

```
}

ll C(int a,int b)
{
    if(b<0||b>a) return 0;
    return fact[a]*infact[a-b]%mod*infact[b]%mod;
}
```

$C(n,m)\ mod\ p, 1 \le m \le n \le 10^9$

p为素数

利用分段打表的思想

我们本地跑出来，在模数范围内，10^6所有**倍数**的阶乘值。

例如我们要求，$10^8+12345$，则我们可以直接知道10^8的阶乘值，接下来递推就可以获得我们想要的结果。

时间复杂度：询问$O(10^6)$。

```
ll res = 1;
cout<<res<<',';
for(int i=1;i<mod;i++)
{
    res = res*i%mod;
    if(i%1000000==0) cout<<res<<',';
}
```

```
ll fact[] = {....}

ll fac(int n)
{
    ll v = fact[n/T];
    for(int i=n/T*T+1;i<=n;i++)
        v = v*i%mod;
    return v;
}
```

```
ll C(int a,int b)
{
    if(b<0||b>a) return 0;
    return fac(a)*get_inv(fac(b)*fac(a-
b)%mod,mod)%mod;
}
```

Lucas求组合数

p小，n，m大，p为素数

```
mint fact[mod],infact[mod];

void init()
{
    fact[0] = infact[0] = 1;
    for(int i=1;i<mod;i++) fact[i] = fact[i-
1]*mint(i);
    infact[mod-1] = fact[mod-1].inv();
    for(int i=mod-2;i;i--) infact[i] =
infact[i+1]*mint(i+1);
}

mint c(LL a,LL b)
{
    if(a<b) return mint(0);
    return fact[a]*infact[a-b]*infact[b];
}

mint lucas(LL a,LL b)
{
    if(b<0||b>a) return 0;
    if(a<mod&&b<mod) return c(a,b);
    return c(a%mod,b%mod)*lucas(a/mod,b/mod);
}
```

常见应用：考虑`p=2`，则此时答案即为`n&m==m`

或者与数位DP组合

Ex-Lucas求组合数

p小，n，m大，p为合数

基本上模合数都是这个方法

```
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false); cin.tie(0),
cout.tie(0)
using namespace std;
using ll = long long;
typedef pair<ll,ll> PII;
const int N = 1e5 + 10;

int m,T,M;
PII x[110];
ll pr[110],a[N],b[N],fac[1010000],ans[N],phipe;
ll cntp,cnts;

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

ll calc(ll a,int p,int pe,int w)//a的阶乘对，某一个整数
幂，拆分。
{
    ll val = 1;
    while(a)
```

```
    {
        cntp += (a/p)*w;
        cnts += (a/pe)*w;
        val = val*fac[a%pe]%pe;
        a /= p;
    }
    return  val;
}

ll C(ll a,ll b,int p,int pe)
{
    cntp = 0,cnts = 0;
    auto f1 = calc(a,p,pe,1);
    auto f2 = calc(b,p,pe,-1);
    auto f3 = calc(a-b,p,pe,-1);
    ll v1 = f1*ksm(f2*f3%pe,phipe - 1,pe)%pe;
    ll v2 = ksm(p,cntp,pe);
    ll v3 = ksm(fac[pe],cnts,pe);
    return v1*v2%pe*v3%pe;
}

int main()
{
    ios;
    cin>>m>>T;
    M = m;
    int t = 0;
    for(int i=2;i<=m;i++)
        if(m%i==0)
        {
            int p = i,pe = 1;
            while(m%i==0) m/=i,pe *= i;
            x[t++] = {p,pe};
        }

    for(int i=0;i<t;i++)
    {
        int pe = x[i].second;
```

```
        int Mi = M / pe;
        for(int c=0;c<M;c+=Mi)
            if(c%pe==1)
            {
                pr[i] = c;
                break;
            }
    }
    for(int i=0;i<T;i++) cin>>a[i]>>b[i];
    for(int i=0;i<t;i++)
    {
        int p = x[i].first,pe = x[i].second;
        fac[0] = 1;
        for(int j=1;j<=pe;j++)
        {
            if(j%p==0) fac[j] = fac[j-1];
            else fac[j] = fac[j-1]*j%pe;
        }
        phipe = pe/p*(p-1);
        for(int j=0;j<T;j++)
            ans[j] =
(ans[j]+C(a[j],b[j],p,pe)*pr[i])%M;
    }
    for(int i=0;i<T;i++) cout<<ans[i]<<'\n';
    return 0;
}
```

我们还可以也能用这个完成计算多重集的组合

其中是将，n个礼物分发到m个人手里，每个人分到的依次为w_i

```
#include<bits/stdc++.h>
#define ios ios::sync_with_stdio(false); cin.tie(0),
cout.tie(0)
using namespace std;
using ll = long long;
typedef pair<ll,ll> PII;
const int N = 1e5 + 10,mod = 999911659;
```

```
int m,T,M,n;
PII x[110],a[110];
ll pr[110],fac[1010000],ans,phipe;
ll cntp,cnts;

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

ll calc(ll a,int p,int pe,int w)
{
    ll val = 1;
    while(a)
    {
        cntp += (a/p)*w;
        cnts += (a/pe)*w;
        val  = val*fac[a%pe]%pe;
        a /= p;
    }
    return  val;
}

ll calc(int p,int pe)
{
    cntp = 0,cnts = 0;
    ll v1 = 1;
    for(int i=0;i<T;i++)
    {
        ll f = calc(a[i].first,p,pe,a[i].second);
        v1 = v1*ksm(f,phipe + a[i].second,pe)%pe;
```

```
    }
    ll v2 = ksm(p,cntp,pe);
    ll v3 = ksm(fac[pe],cnts,pe);
    return v1*v2%pe*v3%pe;
}

int main()
{
    ios;
    cin>>m;
    M = m;
    int t = 0;
    for(int i=2;i<=m;i++)
        if(m%i==0)
        {
            int p = i,pe = 1;
            while(m%i==0) m/=i,pe *= i;
            x[t++] = {p,pe};
        }

    for(int i=0;i<t;i++)
    {
        int pe = x[i].second;
        int Mi = M / pe;
        for(int c=0;c<M;c+=Mi)
            if(c%pe==1)
            {
                pr[i] = c;
                break;
            }
    }
    cin>>n>>m;
    a[T++] = {n,1};
    int s = n;
    for(int i=0;i<m;i++)
    {
        int x;cin>>x;
        a[T++] = {x,-1};
```

```
        s-=x;
    }
    if(s<0)
    {
        cout<<"Impossible\n";
        return 0;
    }
    if(s>0) a[T++] = {s,-1};
    for(int i=0;i<t;i++)
    {
        int p = x[i].first,pe = x[i].second;
        fac[0] = 1;
        for(int j=1;j<=pe;j++)
        {
            if(j%p==0) fac[j] = fac[j-1];
            else fac[j] = fac[j-1]*j%pe;
        }
        phipe = pe/p*(p-1);
        ans = (ans + calc(p,pe)*pr[i])%M;
    }
    cout<<ans<<'\n';
    return 0;
}
```

高精算组合数

```
const int N=5010;

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

void get_primes(int n)
{
    for(int i=2;i<=n;i++)
    {
        if(!st[i])primes[cnt++]=i;
```

```
        for(int j=0;primes[j]*i<=n;j++)
        {
            st[primes[j]*i]=true;
            if(i%primes[j]==0)break;//==0每次漏
        }
    }
}
// 对p的各个<=a的次数算整除下取整倍数
int get(int n,int p)
{
    int res =0;
    while(n)
    {
        res+=n/p;
        n/=p;
    }
    return res;
}
//高精度乘
vector<int> mul(vector<int> a, int b)
{
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i ++ )
    {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }
    while (t)
    {
        c.push_back(t % 10);
        t /= 10;
    }
    // while(C.size()>1 && C.back()==0)
C.pop_back();//考虑b==0时才有pop多余的0 b!=0不需要这行
    return c;
}
```

```
int main()
{
    int a,b;
    cin >> a >> b;
    get_primes(a);

    for(int i=0;i<cnt;i++)
    {
        int p = primes[i];
        sum[i] = get(a,p)-get(a-b,p)-get(b,p);//是a-b不是b-a
    }

    vector<int> res;
    res.push_back(1);

    for (int i = 0; i < cnt; i ++ )
        for (int j = 0; j < sum[i]; j ++ )//primes[i]的次数
            res = mul(res, primes[i]);

    for (int i = res.size() - 1; i >= 0; i -- ) printf("%d", res[i]);
    puts("");

    return 0;
}
```

burnside引理与polya定理

置换：排列的相互之间的映射。

$12......n$映射为$p_1p_2......p_n$

循环置换：映射封闭

$12......n$映射为$23......1$

burnside引理：每个置换的不动点个数的平均值。

不动点：对于某个置换来说，某一种染色方案经过置换后，方案不变，即为一个不动点。

image-20220713182041818

polya定理：对于染色没有要求，则每个置换内的各各循环置换的颜色肯定相同，则一个置换拆分为k个循环置换后，其不动点数为c^k

image-20220713182318462

这里给一道最简单例题。

m种颜色的佛珠，选出n个组成一个项链，其中翻转和旋转后相同算一种

看到多重集的圆排列可以想到

则其推导过程为

1. **旋转**

   假设旋转k后相同，k取值范围为0,1,2...n-1,则答案为$\sum_{k=0}^{m} m^{(n,k)}$

2. **翻转**

   奇数：$nm^{\frac{n+1}{2}}$

   偶数：$\frac{n}{2}m^{\frac{n}{2}+1}$

斯特林数

第一类斯特林数

将1~n划分成k个圆排列的方案数，记作s(n,k) 成 $\begin{bmatrix} n \\ k \end{bmatrix}$

递推式：$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix}$

性质：

1. **s(0,0) = 1**
2. **s(n,0) = 0**
3. **s(n,n) = 1**
4. **s(n,1) = (n - 1)!**
5. **s(n,n-1) = C(n,2)**
6. **s(n,2) =** $\sum_{i=1}^{n-1}\frac{C(n,i)(i-1)!(n-i-1)!}{2}=(n-1)!\sum_{i=1}^{n-1}\frac{1}{i}$
7. **s(n,n-2) = 2C(n,3) + 3C(n,4)**
8. $\sum_{k=0}^{n}s(n,k)=n!$

应用

有n个仓库，每个仓库有两把钥匙，共2n把钥匙。同时又有n位官员。问如何放置钥匙使得所有官员都能够打开所有仓库？（只考虑钥匙怎么放到仓库中，而不考虑官员拿哪把钥匙。）那如果官员分成m个不同的部，部中的官员数量和管理的仓库数量一致。那么有多少方案使得，同部的所有官员可以打开所有本部管理的仓库，而无法打开其他部管理的仓库？（同样只考虑钥匙的放置。）

第一问很经典，就是打开将钥匙放入仓库构成一个环：1号仓库放2号钥匙，2号仓库放3号钥匙......n号仓库放1号钥匙。这种情况相当于钥匙和仓库编号构成一个圆排列方案数是**(n-1)!**种。而第二问就对应的将n个元素分成m个圆排列，方案数就是第一类无符号Stirling数**s(n,m)**。如要要考虑官员的情况，只需再乘上 **n!** 即可。

第二类斯特林数

将1~n划分为k个非空子集的方案数，记作S(n,k)成$\begin{Bmatrix}n\\k\end{Bmatrix}$

递推式：$\begin{Bmatrix}n\\k\end{Bmatrix}=\begin{Bmatrix}n-1\\k-1\end{Bmatrix}+k\begin{Bmatrix}n-1\\k\end{Bmatrix}$

初值`s[0][0]=1`

性质

1. **S(n,0) =** 0^n

2. **S(n,1) = 1**
3. **S(n,n) = 1**
4. **S(n,2) =** $2^{n-1} - 1$
5. **S(n,n-1) = C(n, 2)**
6. **S(n,n-2) = C(n,3) + 3C(n,4)**
7. **S(n,n-3) = C(n,4) +10C(n,5) +15C(n,6)**
8. $\sum_{k=0}^{n} S(n,k) = B_n$
9. **通项公式** $S(n,m) = \frac{1}{m!}\sum_{k=0}^{m}(-1)^k C(m,k)(m-k)^n$
10. $x^n = \sum_{i=0}^{n} S(n,i)x^{\underline{i}}$
11. $m^n = \sum_{k=0}^{m} P(m.k)S(n,k)$

通项公式求单项

$O(m)$

```
mint fact[N],infact[N],qpow[N];
int primes[N],cnt;
bool st[N];

void init(int n)
{
    fact[0] = 1, infact[0] = 1;
    for(int i=1;i<N;i++)
        fact[i] = fact[i-1]*i;
    infact[N-1] = fact[N-1].pow(mod-2);
    for(int i=N-2;i;i--)
        infact[i] = infact[i+1]*(i+1);
    qpow[1] = 1;
    for(int i=2;i<N;i++)
    {
        if(!st[i])
        {
            primes[cnt++] = i;
            qpow[i] = mint(i).pow(n);
        }
        for(int j=0;primes[j]*i<N;j++)
        {
```

```
            st[primes[j]*i] = 1;
            qpow[primes[j]*i] =
qpow[primes[j]]*qpow[i];
            if(i%primes[j]==0) break;
        }
    }
}

mint C(int a,int b)
{
    return fact[a]*infact[a-b]*infact[b];
}

mint S(int n,int m)
{
    init(n);//由于每次n的不同，用通项求每次都要init一下
    mint ans = 0;
    for(int i=0;i<=m;i++)
    {
        mint f = 1;if(i&1) f = -1;
        ans += f*C(m,i)*qpow[m-i];
    }
    ans *= infact[m];
    return ans;
}
```

行

第二类斯特林数$\begin{Bmatrix} n \\ m \end{Bmatrix}$表示把$n$个**不同**元素划分成$m$个**相同**的集合中（不能有空集）的方案数。

给定n，对于所有的整数$i \in [0, n]$，你要求出$\begin{Bmatrix} n \\ i \end{Bmatrix}$。

由于答案会非常大，所以你的输出需要**对**167772161 **（**$2^{25} \times 5 + 1$**，是一个质数）取模。**

应用

（1）n个不同的球，放入m个无区别的盒子，不允许盒子为空。

方案数：**S(n,m)**

。这个跟第二类Stirling数的定义一致。

（2）n个不同的球，放入m个有区别的盒子，不允许盒子为空。

方案数：**S(n,m)*m!**

。因盒子有区别，乘上盒子的排列即可。

（3）n个不同的球，放入m个无区别的盒子，允许盒子为空。

方案数：$\sum_{k=0}^{m} S(n,k)$

。枚举非空盒的数目便可。

（4）n个不同的球，放入m个有区别的盒子，允许盒子为空。

①方案数：方案数：$\sum_{k=0}^{m} P(m,k)S(n,k)$ 。同样可以枚举非空盒的数目，注意到盒子有区别，乘上一个排列系数。

②既然允许盒子为空，且盒子间有区别，那么对于每个球有m种选择，每个球相互独立。有方案数：m^n 。

则$m^n = \sum_{k=0}^{m} P(m,k)S(n,k)$

两类斯特林数之间的关系

$\sum_{k=0}^{n} S(n,k)s(k,m) = \sum_{k=0}^{n} s(n,k)S(k,m)$

卡特兰数

$\frac{\mathrm{C}_{2n}^{n}}{n+1}$

$\mathrm{C}_{2n}^{n} - \mathrm{C}_{2n}^{n-1}$

若是n行m列(n>=m)

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

求导公式

基本初等函数求导公式

(1) $(C)' = 0$

(2) $(x^{\mu})' = \mu x^{\mu-1}$

(3) $(\sin x)' = \cos x$

(4) $(\cos x)' = -\sin x$

(5) $(\tan x)' = \sec^2 x$

(6) $(\cot x)' = -\csc^2 x$

(7) $(\sec x)' = \sec x \tan x$

(8) $(\csc x)' = -\csc x \cot x$

(9) $(a^x)' = a^x \ln a$

(10) $(e^x)' = e^x$

(11) $(\log_a x)' = \dfrac{1}{x \ln a}$

(12) $(\ln x)' = \dfrac{1}{x}$,

(13) $(\arcsin x)' = \dfrac{1}{\sqrt{1-x^2}}$

(14) $(\arccos x)' = -\dfrac{1}{\sqrt{1-x^2}}$

(15) $(\arctan x)' = \dfrac{1}{1+x^2}$

(16) $(\operatorname{arccot} x)' = -\dfrac{1}{1+x^2}$

函数的和、差、积、商的求导法则

设 $u = u(x)$， $v = v(x)$ 都可导，则

(1) $(u \pm v)' = u' \pm v'$

(2) $(Cu)' = Cu'$（C 是常数）

(3) $(uv)' = u'v + uv'$

(4) $\left(\dfrac{u}{v}\right)' = \dfrac{u'v - uv'}{v^2}$

反函数求导法则

若函数 $x = \varphi(y)$ 在某区间 I_y 内可导、单调且 $\varphi'(y) \neq 0$，则它的反函数

$y = f(x)$ 在对应区间 I_x 内也可导，且

$$f'(x) = \frac{1}{\varphi'(y)} \quad 或 \quad \frac{dy}{dx} = \frac{1}{\frac{dx}{dy}}$$

Fibonacci Numbers

- $f_i = f_{i-1} * f_2 + fi - 2 * f_1$
- $\sum_{i=1}^{n} f_i = f_n * f_1 + (f_{n+1} - 1) * f_2$
- **设a数组符合Fibonacci数列的递推式，其中a_1, a_2为任意值**

  其有$a_i = f_{i-1} * a_2 + fi - 2 * a_1$，以及
  $\sum_{i=1}^{n} a_i = f_n * a_1 + (f_{n+1} - 1) * a_2$

- **通项公式为**

$$F_n=\frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^n-(\frac{1-\sqrt{5}}{2})^n]$$

- $f_{n+m}=f_{n+1}f_m+f_nf_{m-1}$

矩阵

常用

```
struct Matrix{
    int m[4][4];

    void clear(){
        for(int i=0;i<4;i++){
            for(int j=0;j<4;j++)
                m[i][j]=0;
        }
    }

    void init(){
        clear();
        for(int i=0;i<4;i++)
            m[i][i]=1;
    }

    void print(){
        for(int i=1;i<=2;i++){
            for(int j=1;j<=2;j++)
                printf("i=%d,j=%d,m=%d\n",i,j,m[i]
[j]);
        }
    }

    bool empty(){
        if(m[1][1]!=1) return 0;
        if(m[1][2]!=0) return 0;
        if(m[2][1]!=0) return 0;
```

```
        if(m[2][2]!=1) return 0;
        return 1;
    }

    Matrix operator*(const Matrix &y) const {
        Matrix z; z.clear();
        for(int i=1;i<=2;i++){
            for(int k=1;k<=2;k++){
                for(int j=1;j<=2;j++)
                    z.m[i][j]=(z.m[i][j]+1ll*m[i]
[k]*y.m[k][j])%mod;
            }
        }
        return z;
    }

    friend Matrix operator+(Matrix a,Matrix b){
        Matrix c;c.clear();
        for(int i=1;i<=2;i++){
            for(int j=1;j<=2;j++)
                c.m[i][j]=(1ll*a.m[i][j]+b.m[i]
[j])%mod;
        }
        return c;
    }

    int* operator[](int x)
    {
        return m[x];
    }
};

Matrix dw,fir;

Matrix mpow(Matrix a,int n)
{
    Matrix c;c.init();
    while(n)
```

```
    {
        if(n&1) c = c*a;
        a = a*a;
        n>>=1;
    }
    return c;
}
```

全，上边不够用从下边取。

```
namespace Matrix
{
    #define type int
    const int Xsize = 80;
    const int Ysize = 80;
    const type mod = 1e9+7;
    struct matrix
    {
        vector<vector<type>> a;
        int xlen, ylen;
        matrix(int x=Xsize, int y=Ysize)
        {
            xlen = x, ylen = y;
            a.resize(x+1);
            for(int i = 1; i <= x; i++)
            {
                a[i].resize(y+1);
                a[i].assign(y+1, 0);
            }
        }
        vector<type>& operator [] (int x)
        {
            return a[x];
        }
    };
    void throw_error()
    {
        cout << "Matrix error!";
```

```cpp
        std::exit(0);
    }
    matrix operator + (matrix a, matrix b)
    {
        if(a.xlen != b.xlen or a.ylen != b.ylen)
            throw_error();
        for(int i = 1; i <= a.xlen; i++)
        for(int j = 1; j <= a.ylen; j++)
            (a[i][j] += b[i][j]) %= mod;
        return a;
    }
    matrix operator - (matrix a, matrix b)
    {
        if(a.xlen != b.xlen or a.ylen != b.ylen)
            throw_error();
        for(int i = 1; i <= a.xlen; i++)
        for(int j = 1; j <= a.ylen; j++)
            (a[i][j] -= b[i][j]) %= mod;
        return a;
    }
    matrix operator * (matrix a, matrix b)
    {
        if(a.ylen != b.xlen) throw_error();
        matrix ans(a.xlen, b.ylen);
        for(int i = 1; i <= a.xlen; i++)
        for(int j = 1; j <= b.ylen; j++)
        for(int k = 1; k <= a.ylen; k++)
            (ans[i][j] += a[i][k]*b[k][j]) %= mod;
        return ans;
    }
    matrix operator * (matrix a, type k)
    {
        for(int i = 1; i <= a.xlen; i++)
        for(int j = 1; j <= a.ylen; j++)
            (a[i][j] *= k) %= mod;
        return a;
    }
    matrix operator % (matrix a, type k)
```

```
    {
        for(int i = 1; i <= a.xlen; i++)
        for(int j = 1; j <= a.ylen; j++)
            a[i][j] %= k;
        return a;
    }
    matrix& operator += (matrix &a, matrix b){ return
(a = a+b); }
    matrix& operator -= (matrix &a, matrix b){ return
(a = a-b); }
    matrix& operator *= (matrix &a, matrix b){ return
(a = a*b); }
    matrix& operator *= (matrix &a, type k){ return (a
= a*k); }
    matrix& operator %= (matrix &a, type k){ return (a
= a%k); }
    matrix pow(matrix a, long long p, type k=mod)
    {
        if(a.xlen != a.ylen) throw_error();
        matrix ans(a.xlen, a.ylen);
        for(int i = 1; i <= a.xlen; i++)
            ans[i][i] = 1;
        for(; p; p >>= 1, (a *= a) %= k)
            if(p&1) (ans *= a) %= k;
        return ans;
    }
    void print(matrix a)
    {
        for(int i = 1; i <= a.xlen; i++)
        {
            for(int j = 1; j <= a.ylen; j++)
                cout << a[i][j] << ' ';
            cout << endl;
        }
    }
}
using namespace Matrix;
```

博弈论

定义 $Position$
P： 当前局面下先手必败
N： 当前局面下先手必胜

N , P 状态的转移满足如下性质:

1. 合法操作集合为空的局面为 P
2. 可以移动到 P 的局面为 N ，这个很好理解，以为只要能转换到 P 局面，那么先手只需要使操作后变成 P 局面，那么后手就面临一个必败的状态。
3. 所有移动只能到达 N 的局面为 P。无论怎么选取都会给队友留下一个必胜的状态。

其实知道这个做法就可以通过记忆化搜索或者用 sg 函数来求解，如果范围非常大，就没有办法做了。

一、 Nim 游戏

给定 n 堆石子 $a_1, a_2, ..., a_n$ ，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。

结论：若 $a_1 \ \wedge \ a_2 \wedge ... \wedge a_n = 0$ ，则先手必败，否则先手必胜。

在讲 Nim 游戏之前，先了解一下什么是 (ICG) 。若一个游戏满足以下条件，则该游戏称为公平组合游戏:

- 由两名玩家交替行动
- 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
- 不能行动的玩家判负

Nim 游戏属于公平组合游戏，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，胜负判定也比较复杂，不满足条件 2 和 3 。

什么是**先手必胜状态**和**先手必败状态**？

- 先手必胜状态：先手进行**某一个操作**，留给后手的是一个必败状态时，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。
- 先手必败状态：先手**无论如何操作**，留给后手的都是一个必胜状态的时候，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

为什么 $a_1 \ \wedge \ a_2 \wedge ... \wedge a_n = 0$，则先手必败，否则先手必胜？

1. 当到达结束状态的时候，即无法再进行任何操作，此时 $0 \ \wedge \ 0 \wedge ... \wedge 0 = 0$;
2. 如果其中某一个状态 $a_1 \ \wedge \ a_2 \wedge ... \wedge a_n = x \ (x \neq 0)$，那么一定有一种拿法使得剩下的数字异或和为 0。证明如下：假设 x 的二进制表示里最高的一位是在第 k 位，那么 a_1 ~ a_n 中至少有一个数字 a_i 的第 k 位是 1 （如果全为 0，那么第 k 位一定是 0），那么显然 $a_i \wedge x < a_i$，那么我们可以从中拿走 $a_i - (a_i \wedge x)$ 个石子，那么第 i 堆石子就剩下 $a_i \wedge x$ 个石子，因此此时所有数字的异或和为 0。
3. 如果 $a_1 \ \wedge \ a_2 \wedge ... \wedge a_n = 0$，那么不管怎么拿，剩下所有数字的异或和一定不为 0。

综上所述，当先手局面为 $a_1 \ \wedge \ a_2 \wedge ... \wedge a_n = 0$ 的时候，抛给后手的状态一定是 $a_1 \ \wedge \ a_2 \wedge ... \wedge a_n \neq 0$ 当先手局面是 $a_1 \ \wedge \ a_2 \wedge ... \wedge a_n \neq 0$，那么他一定会使得后手的状态是 $a_1 \ \wedge \ a_2 \wedge ... \wedge a_n = 0$。因此当$a_1 \ \wedge \ a_2 \wedge ... \wedge a_n = 0$，则先手必败，否则先手必胜

二、$Moore's \ Nimk$ （尼姆博弈问题的拓展）

n 堆石子，每次从不超过 k 堆中取任意多个石子，最后不能取的人失败。

这是一个 nim 游戏的变形，也具有结论。

结论为：把 n 堆石子数用二进制表示，统计每个二进制位上 1 的个数，若每个位上 1 的个数 $mod\ (k+1)$ 全部为 0，则必败，否则必胜。

证明如下：

1. 全为 0 的局面一定是必败态。
2. 任何一个 P 状态，经过一次操作以后一定为到达 N，在某一次移动中，至少有一堆被改变，也就是至少有一个二进制位被改变。由于最多 k 堆石子，也就是对于一个二进制位，1 的个数至多改变 k。而由于原先的总数为 $k+1$ 的整数倍，所以改变之后必然不可能是 $k+1$ 的整数倍。故在 P 状态下一次操作的结果必然是 N 状态。
3. 任何 N 状态，总有一种操作使其变化成 P 状态。从高位到低位考虑所有的二进制位。假设用了某种方法，改变了 m 堆，使 i 为之前的所有位都回归到 $k+1$ 的整数倍。现在要证明总有一种方法让第 i 位也恢复到 $k+1$ 的整数倍。

有一个比较显然的性质，对于那些已经改变的 m 堆，当前位可以自由选择 1 或 0.

设除去已经更改的 m 堆，剩下堆 i 位上 1 的总和为 sum

分类讨论：

1. $sum \le k-m$, 此时可以将这些堆上的 1 全部拿掉，然后让那 m 堆得 i 位全部置成 0.
2. $sum > k-m$ 此时我们在之前改变的 m 堆中选择 $k+1-sum$ 堆，将他们的第 i 位设置成 1。剩下的设置成 0. 由于 $k+1-sum < k+1-(k-m) < m+1$, 也就是说 $k+1-sum \le m$，故这是可以达到的.

三、 $anti-nim$ （反 Nim 游戏）

反 nim 游戏。正常的 nim 游戏是取走最后一颗的人获胜，而反 nim 游戏是取走最后一颗的人输。

一个状态为必胜态，当且仅当

1. 所有堆的石子个数为 1 ，且**异或和**为0
2. 至少有一堆的石子个数大于 1，且**异或和**不为0

四、威佐夫博弈

两堆石子，每次可以取一堆或两堆，从两堆中取得时候个数必须相同，先取完的获胜

五、巴什博奕

只有一堆石子共 n 个。每次从最少取 1 个，最多取 m 个，最后取光的人取胜。

如果 $n=(m+1)\times k+s(s\neq 0)$ 那么先手一定必胜，因为一次取走 s 个，接下来无论怎么取，我们都能保证取到所有 $m+1$ 倍数的点，循环下去一定可以取到最后一个。

六、$Take-and-BreakGame$

n 堆石子，每次可以取走一堆石子，然后放入两堆规模更小的石子 (可以为 0). 最后不能操作的人输。

使用 SG 函数求解。

`f[i]` 表示 还剩一堆 i 颗石子的状态， `f[i][j]` 表示两堆的状态，然后依次类推。

根据 `SG` 函数的定义有 `f[i] = min{x ∈ N | n != f(p, q) (i > p >= 0 && i - q >= 0) }`

然后递推求得子游戏的任意状态

七、$staircasenim$

gcd的一些性质

- **gcd(a,b) = gcd(a, a+b) = gcd(a, ka+b)**
- **gcd(ka, kb) = k·gcd(a, b)**
- **定义多个整数的最大公约数：gcd(a, b, c) = gcd(gcd(a, b), c)**
- **若gcd(a, b) = d，则gcd(a/d, b/d) = 1，即a/d与b/d互素。**
- **gcd(a+cb, b) = gcd(a, b)**

数论的一些公式及结论

- **欧拉定理**

  $a^{\phi(p)} \equiv 1(mod\ p)$，`a,p`互质

- **扩展欧拉定理**
  - $b \geq \phi(p)\ a^b \equiv a^{b\ mod\ \phi(p)+\phi(p)}(mod\ p)$
  - $b < \phi(p)\ a^b \equiv a^{b\ mod\ \phi(p)}(mod\ p)$

  其中`a,p`可以不互质。

  利用这个式子，可以快减少幂的大小。

- $\phi(i)一定是偶数,且一定 \leq \frac{1}{2}i（i > 2）$
- **裴蜀定理**

  若$a, b \in N, gcd(a, b) = d$，则任意的整数x,y,ax+by都一定是d的倍数，同时可以推出一定存在$x, y \in N$使得$ax + by = d$成立

  注意：最常用的是`gcd(a,b)=1`时，一定要想到裴蜀定理，看看能不能从这个角度下手

- **下边两个的前提是，选择环形数组的某个起始点`s`，每次跳步长`k`**
- **如果`k`与`n`互质，这`n`次将取遍所有下标。**
  - **进一步的，对于不同的`k1，k2`，如果`gcd(k1,n)=gcd(k2,n)`，对于相同的`s`而言，答案一致**
- **一般的，对于多个数$a_1, a_2, a_3, ..., a_n$做欧几里得的时间复杂度是** $O(n + logmaxa_i)$
- **素数无限**
- $lim_{n->\infty}\pi(n) = \frac{n}{lnn}$

- $P_n = O(nlogn)$
- $\sum_{i=1}^{n} \frac{1}{i} = O(logn)$
- $\sum_{1 \le p \le n} \frac{1}{p} = O(loglogn)$
- **a|c,b|c,(a,b) => ab|c**
- **a|bc,(a,b)=1 => a|c**
- **p|ab => p|a或p|b**
- $a \equiv b(mod\ m) \Leftrightarrow m|b-a$
- $a \equiv b(mod\ m), a \equiv b(mod\ n) \rightarrow a \equiv b(mod\ [m,n])$
- $(k,m)\ =\ d, ka \equiv ka'(mod\ m) \rightarrow\ a \equiv\ a'(mod\ \frac{m}{d})$

组合数学的一些公式及结论

组合公式

- **n个相同的小球，放到m个不同的盒子中，答案为** $C(m+n-1, m-1)$
- $\binom{n}{k} * k^{\underline{m}} = \binom{n-m}{k-m} * n^{\underline{m}}$
- $k * \binom{n}{k} = n * \binom{n-1}{k-1}$

组合结论

- `x1+x2+x3+...+xk=m`**，其中**`xi>=0`**，答案为** $C(m+k-1, k-1)$
- **一个序列有**`k`**个数，数字的取值范围为[l,r]，则若要使序列单调不降/不升的合法序列个数为多少**

  结论是，$C(r-l+1+k, r-l)$。
- **递推式为**`f[n][m] = f[n-1][m-1] + f[n-1][m]`**，但不是标准的杨辉三角，但可以有标准杨辉三角加起来得到。标准杨辉三角是由初状态为**`C(i,0)=1,C(i,i)=1`**得到的。因此我们求当前的**`f[n][m]`**时，考虑推出其初状态然后，用标准的杨辉三角拼凑得到。**

多项式的一些公式与结论

小技巧

long long 相乘取模

可以**龟速乘**，可以强转一下**i128**，或者像下面这样写。

```
ll mul(ll x,ll y, ll m)
{
    x %= m;y %= m;
    ll d = ((long double)x*y/m);
    d = x*y - d * m;
    if(d >= m) d -= m;
    if(d<0) d += m;
    return d;
}
```

上下取整

有正有负时

```
ll floordiv(ll a,ll b)
{
    if(a%b==0) return a/b;
    else if(a>0) return a/b;
    else return a/b - 1;
}

ll ceildiv(ll a,ll b)
{
    if(a%b==0) return a/b;
    else if(a>0) return a/b + 1;
    else return a/b;
}
```

考虑棋盘反射时，不妨将棋盘扩展开来考虑。

细碎结论

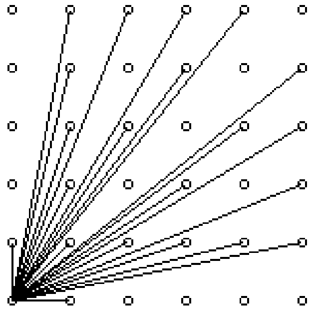
1. **一个圆上有n个点，从某一点x，每次跳k个点，则一定会回到x。设d = gcd(k, n)，总共跳了n/d个点。为什么呢？我们可以考虑，我们将**`k和n`**同时**`/d`**，这样整个图都被缩放了，接下来问题等价于，问一次跳**`k'`**，圆上点为**`n'`**且两者互质，会跳几次，那就很显然了，就是要跳**`n'`**次即为'n/d'，同时我们可以知道，总共走了**`k/d`**圈。我们可以发现，一个循环的相邻两个点的距离为d。则圆上所有点都会构成d个循环。**
2. **根号分治，处理某一个数的质因子种类个数，先把$\sqrt{n}$的质数处理出来，接下来对于1~n中的任意一个数可以在$\sqrt{i}/log(i)$时间内处理出所有质因子个数**
3. **相邻两个质数之间最多差300左右**
4. **2e9内约数最多的一个数，其约数个数不超过1600个**
5. **n以内的素数个数$\frac{n}{lnn}$**
6. **若 a, b 互质，那么不能由 $ax + by$ 凑出的最大数是 $(a-1) \times (b-1) - 1$**
7. **结论：直角边为i，j的直角三角形其斜边上落在格点上的点数为gcd(i,j)+1**
8. **求0~N中，互质对的数量**

   phi[1] = 1，其余正常。

   求1~N中，互质对的数量

   phi[1] = 0，其余正常。

   $$ans = 2 * (phi[1] + phi[2] + \cdots phi[N]) + 1$$
9. **求过(0,0)，这样的直线的个数，或者说是站到(0,0)可以看到的点数，方阵的范围是[0,N]，若是为[1,N]则$\phi(1) = 0$**



被照到的点的`x`和`y`都是互质的

问题则转换为了`[0,N]`中所有的互质对数量`(x,y)`的数量

以y=x为分界线

利用左上和右下对称分布，左上互质对数量==右下互质对数量

则只用看右下互质对数量

则右下所有横坐标对应的互质数个数为=$\sum_{i=1}^{N}\phi(i)$

则右下+左上所有横坐标对应的互质数个数为=$2*\sum_{i=1}^{N}\phi(i)$

则右下+左上+中间所有横坐标对应的互质数个数为=
$2*\sum_{i=1}^{N}\phi(i)+1$

10. **两个不超过n的数的最小公倍数最大是n*(n-1)**
11. **求min{**$\vec{a}\cdot\vec{b}+\vec{b}\cdot\vec{c}+\vec{c}\cdot\vec{a}$**}，其中**$|\vec{a}|=r1,|\vec{b}|=r2,|\vec{c}|=r3$**，**$r1\le r2\le r3$

    $\vec{a}\cdot\vec{b}+\vec{b}\cdot\vec{c}+\vec{c}\cdot\vec{a}=\frac{1}{2}(|\vec{a}+\vec{b}+\vec{c}|^2-|\vec{a}|^2-|\vec{b}|^2-|\vec{c}|^2)$

    - $r1+r2>r3$，则$|\vec{a}+\vec{b}+\vec{c}|$的最小值为`0`
    - $r1+r2\le r3$，则最小值为`r3-r2-r1`
12. **有一个长为n的01串，初始值为0，每次可以将它的一位取反，最终值为**2^n**-1，这样不相交的变换01串的方法的为n种。**

    对于n=3，例子为。

    0-1-3-7

    0-2-6-7

    0-4-5-7

    构造出一组合法解为

对于每一个n位01串，从右开始其每一位分别设为

$a_0 a_1 \ldots\ldots a_{n-1}$

于是我们可以轻松地给出这样nn条路径的一组构造（以下均写改变的a_{i}*a**i*的下标）：

0 -- 1 ...... -- n-1

1 -- 2 ...... -- n-1 -- 0

......

n-1 -- 0 ...... -- n-2

13. $a\%b = a - b * \lfloor \frac{a}{b} \rfloor$