数学

杂货

哥德巴赫猜想:任何一个大于2的偶数,都可以表示为两个素数之和.另外还有,任何一个大于5的奇数都可以表示为三个素数之和.

本福特定律:在b进制中,以数a起头的数出现的几率为$\log_b(a+1)-\log_b a$

在有通项公式的情况下,基于对数理论,可实现求一个大数的前几位

设$S(n)$是n在p进制下各位数字之和，那么$n!$中质数p的幂指数为$\frac{n-S(n)}{p-1}$

$(A/B)\%C = (A\%BC)/B$

递推求i模P的逆元:$inv[i] = (P - P/i) * inv[P\%i]\%P$

欧拉公式优化任意数取模:$A^B\%C = A^{B\%\varphi(C)+\varphi(C)}\%C, B \geq \varphi(C)$
多重指数复合使用时注意底数是1的幂时可能不满足公式

若$a^k \equiv c\%p^i$
则存在$0 \leq j \leq p-1$满足$a^{k+j*\varphi(p^j)} \equiv c\%p^{i+1}$

矩阵模质数p的循环节$A^{\prod_{i=0}^{n-1}p^{n-p^i}}\%p = I$，一般最小循环节一定是它的因子

$\gcd(a^m-b^m, a^n-b^n) = a^{\gcd(m,n)}-b^{\gcd(m,n)}, a>b>0, m>0, n>0$

设$\pi(n)$是小于等于n的素数个数，则有$\lim_{n\to\infty}\frac{\pi(n)}{n/\ln n} = 1$

设$G = \gcd(C_n^1, C_n^2, \cdots C_n^{n-1})$，
若n为素数，$G = n$
若n只有一个质因子，那么G就等于那个质因子
若n只有多个质因子，那么$G = 1$

若p是质数，那么有$C_p^i\%p = 0$故有$(a+b+\cdots+z)^p\%p = a^p+\cdots+z^p$

$(n+1)\,lcm(C_n^0, C_n^1, \cdots C_n^n) = lcm(1, 2, \cdots, n+1)$

$\varphi(ab) = \frac{\varphi(a)\varphi(b)gcd(a,b)}{\varphi(gcd(a,b))}$

$\sum_{d|n}\varphi(d) = n$
$\sum_{i=1}^{n}\gcd(i,n) = \sum_{d|n}d\varphi(\frac{n}{d})$
$\sum_{i=1}^{n}[\gcd(i,n)=1]*i = (n\varphi(n)+[n=1])/2$

对于$\mu(d)$,若$d=1$则为1,若d为无平方因子数则为-1的质因子个数的次方，其它则为0

$\sum_{d|n}\mu(d)=[n=1]$

$\sum_{i=1}^{n}\mu^2(i)=\sum_{i=1}^{i^2\le n}\mu(i)\frac{n}{i^2}$

$\sum_{d|n}\frac{\mu(d)}{d}=\frac{\varphi(n)}{n}$

$\sum_{d=1}^{n}d\cdot\lfloor\frac{n}{d}\rfloor=\sum_{d=1}^{n}\frac{\lfloor\frac{n}{d}\rfloor(\lfloor\frac{n}{d}\rfloor+1)}{2}$

$\sum_{d|m}\varphi(d)n^{m/d}=0\%m$

$\sum_{i=1}^{m}\varphi(n*i)=p^{k-1}((p-1)\sum_{i=1}^{m}\varphi(\frac{n}{p^k}*i)+\sum_{i=1}^{\lfloor\frac{m}{p}\rfloor}\varphi(\frac{n}{p^{k-1}}*i))$,p为n的任意一个质因子,k为p的最高次幂

若$h(n)=\sum_{i=1}^{n}f(i)$,有$\sum_{i=1}^{n}f(i)\lfloor\frac{n}{i}\rfloor=\sum_{i=1}^{n}h(\lfloor\frac{n}{i}\rfloor)=h(n)+\sum_{i=2}^{n}h(\lfloor\frac{n}{i}\rfloor)$,故

$h(n)=\sum_{i=1}^{n}f(i)\lfloor\frac{n}{i}\rfloor-\sum_{i=2}^{n}h(\lfloor\frac{n}{i}\rfloor)$

另，也有$\sum_{i=1}^{n}\sum_{d|i}f(d)\lfloor\frac{i}{d}\rfloor=\sum_{i=1}^{n}i\cdot h(\lfloor\frac{n}{i}\rfloor)$,故

$h(n)=\sum_{i=1}^{n}\sum_{d|i}f(d)\lfloor\frac{i}{d}\rfloor-\sum_{i=2}^{n}i\cdot h(\lfloor\frac{n}{i}\rfloor)$

其实就是尝试找个函数和原函数进行狄利克雷卷积,找的那个函数的前缀和很好求,卷积后的函数的前缀和也很好求,就可以了

杜教筛本质是二重n/i的分段求和,利用线性筛预处理前$n^{2/3}$项,可使总复杂度降为$O(n^{2/3})$

若$F(n)=\sum_{k=0}^{n}C_n^kf(k)$,
则有$f(n)=\sum_{k=0}^{n}(-1)^{n-k}C_n^kF(k)$

若$F(n)=\sum_{d|n}f(d)$,
则有$f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$
另有若$F(n)=\sum_{n|d}f(d)$,
则有$f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$

若$f(S)=\sum_{T\subseteq S}g(T)$,
则有$g(S)=\sum_{T\subseteq S}(-1)^{|S|-|T|}f(T)$
子集反演实现时可考虑每一位相互独立，对每一位分别容斥

设$f(n)=\sum_{i=1}^{n}\lceil\frac{n}{i}\rceil$,则有$f(n)=f(n-1)+d(n-1)+1(n>1)$,其中$d(n)$为n的约数个数.设$g(n)=\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$,有
$g(n)=\sum_{i=1}^{n}d(i),f(n)=g(n-1)+n$

陈立杰-rng58恒等式???

$\sum_{i=1}^{a}\sum_{j=1}^{b} d(ij) = \sum_{gcd(i,j)=1} \lfloor \frac{a}{i} \rfloor \lfloor \frac{b}{j} \rfloor$ (d[i]为因子个数)

可推广至更多维,如$\sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{c} d(ijk) = \sum_{gcd(i,j)=gcd(i,k)=gcd(j,k)=1} \lfloor \frac{a}{i} \rfloor \lfloor \frac{b}{j} \rfloor \lfloor \frac{c}{k} \rfloor$

阿贝尔变换

给定两个数列，a_i和b_i,记$S_k = \sum_{i=1}^{k} a_i, S_0 = 0$

那么有$\sum_{k=1}^{n} a_k b_k = S_n b_n + \sum_{i=1}^{n-1} S_i (b_i - b_{i+1})$

关于k次方前缀和

$S_1 = \frac{1}{2} n(n+1)$

$S_2 = \frac{1}{6} n(n+1)(2n+1)$

$S_3 = \frac{1}{4} n^2(n+1)^2$

$S_4 = \frac{1}{30} n(n+1)(6n^3+9n^2+n-1)$

$S_5 = \frac{1}{12} n^2(n+1)(2n^3+4n^2+n-1)$

$S_6 = \frac{1}{42} n(n+1)(6n^5+15n^4+6n^3-6n^2-n+1)$

$S_7 = \frac{1}{24} n^2(n+1)(3n^5+9n^4+5n^3-5n^2-2n+2)$

$S_8 = \frac{1}{90} n(n+1)(10n^7+35n^6+25n^5-25n^4-17n^3+17n^2+3n-3)$

$S_9 = \frac{1}{20} n^2(n+1)(2n^7+8n^6+7n^5-7n^4-7n^3+7n^2+3n-3)$

$S_{10} = \frac{1}{66} n(n+1)(6n^9+27n^8+28n^7-28n^6-38n^5+38n^4+28n^3-28n^2-5n+5)$

记$S(n,k) = \sum_{i=1}^{n} i^k$

因为$S(n,k)$是一个$k+1$阶的多项式，可以用拉格朗日插值法求解;

有递推公式$S(n,k) = \frac{1}{k+1}[(n+1)^{k+1} - 1 - \sum_{i=2}^{k+1} C_{k+1}^{i} S(n,k+1-i)]$,其中

$S(n,0) = n$;

另有伯努利数法.

有公式可在$O(k)$时间内求得:$S(n,k-1) = \sum_{i=0}^{k} (-1)^{k-i} S(i,k-1) \frac{n(n-1)\cdots(n-k)}{(k-i)!i!(n-i)}$

已知n个数的k次方和$a_k = \sum_{i=1}^{n} x_i^k$,求$b_k = \sum_{1 \le i_1 < i_2 \cdots < i_k \le n} x_{i_1} x_{i_2} \ldots x_{i_m}$ (即k个不同数的乘积和)

由容斥,有$b_0 = 1, b_k = \frac{\sum_{i=1}^{k} (-1)^{i+1} a_i b_{k-i}}{k}$

五边形数和划分数

第n个五边形数为$f(n) = \frac{n(3n-1)}{2}$

*广义五边形数$f_{2n-1} = \frac{n(3n-1)}{2}$和$f_{2n} = \frac{n(3n+1)}{2}$

划分数$p(n)$是指将n拆分为一个或多个互不相同的数的形式的方案数

由五边形数定理有$\prod_{n=1}^{\infty} (1-x^n) = \sum_{k=0}^{\infty} (-1)^k x^{\frac{k(3k\pm1)}{2}}$,即每一项的指数为广义五边形数

另有$\prod_{n=1}^{\infty} \frac{1}{1-x^n} = \sum_{k=0}^{\infty} p(k) x^k$

两式相乘，对比系数，可得$p(n) = \sum (-1)^{i-1} (p(n - f_{2i-1}) + p(n - f_{2i}))$
因为五边形数是平方级别的，所以求$p(n)$只有$\sqrt{n}$项，递推总体复杂度为$n\sqrt{n}$
初始化$p(0) = p(1) = 1, p(2) = 2$

三角平方数定理

方程$x^2 - 2y^2 = 1$的所有正整数解(x_k, y_k)满足$x_k + \sqrt{2}y_k = (3 + 2\sqrt{2})^k$
三角数为$\frac{m(m+1)}{2}$,平方数为n^2
三角平方数$n^2 = \frac{m(m+1)}{2}$的所有解满足$m_k = \frac{x_k - 1}{2}, n_k = \frac{y_k}{2}$

技巧

基于一段两元素乘积和如矩阵乘法的取模优化:
把乘积加起来,维持在**mod*mod**范围内,最后再取一次模

long long范围内任意模数乘法(利用自然溢出):

```
LL mul(LL a,LL b) {
    LL r = (a * b - (LL)(((long double)a * b) / mod) * mod);
    return add(r - r / mod * mod,mod);
}

using i64 = long long;
using u64 = unsigned long long;
using u128 = __uint128_t;

struct Mod64 {
  Mod64() : n_(0) {}
  Mod64(u64 n) : n_(init(n)) {}

  static u64 modulus() { return mod; }
  static u64 init(u64 w) { return reduce(u128(w) * r2); }
  static void set_mod(u64 m) {
    mod = m; assert(mod & 1);
    inv = m; for (int i = 0; i < 5; ++i) inv *= 2 - inv * m;
    r2 = -u128(m) % m;
  }
  static u64 reduce(u128 x) {
    u64 y = u64(x >> 64) - u64((u128(u64(x) * inv) * mod) >> 64);
    return i64(y) < 0 ? y + mod : y;
  }
  Mod64& operator += (Mod64 rhs) { n_ += rhs.n_ - mod; if (i64(n_) < 0) n_ += mod; return *
  Mod64 operator + (Mod64 rhs) const { return Mod64(*this) += rhs; }
```

```
  Mod64& operator *= (Mod64 rhs) { n_ = reduce(u128(n_) * rhs.n_); return *this; }
  Mod64 operator * (Mod64 rhs) const { return Mod64(*this) *= rhs; }
  u64 get() const { return reduce(n_); }
  static u64 mod, inv, r2;
  u64 n_;
};
u64 Mod64::mod, Mod64::inv, Mod64::r2;

Mod64::set_mod(mod);
Mod64 ret = Mod64(x);//把x赋值给ret
return ret.get();//返回ret的值



for(i = 1;i <= n; i = j + 1)
{j = n / (n / i);}
//则每次区间[i,j]的值是相同的 为 n / i

for(i=s;i;i=(i-1)&s);//枚举s子集

如果x满足(x & (-x)) == x, 那么x为2的幂

int __builtin_popcount(i)//计算无符号32位整型数有多少个1
int __builtin_ffs(i)//返回右起第一个1的位置(下标从1开始)
int __builtin_clz(i)//返回左起第一个1之前0的个数
int __builtin_ctz(i)//返回右起第一个1后面0的个数
//在函数名后加ll即对应64位无符号整型数的函数
```

组合数学

组合数

$C_{n-1}^{r} + C_{n-1}^{r-1} = C_{n}^{r}$
$\sum_{k=0}^{n} C_{n}^{k} = 2^{n}$
范德蒙恒等式$\sum_{k=0}^{p} C_{n}^{k} C_{m}^{p-k} = C_{n+m}^{p}$
$\sum_{k=0}^{n} kC_{n}^{k} = n * 2^{n-1}$
$\sum_{k=0}^{n} k^{2} C_{n}^{k} = n(n+1)2^{n-2}$
$\sum_{k=0}^{n} k(C_{n}^{k})^{2} = nC_{2n-1}^{n-1}$
$\sum_{k=0}^{n} k^{3} C_{n}^{k} = n^{2}(n+3)2^{n-3}$
$\sum_{k=0}^{n} k^{2}(C_{n}^{k})^{2} = nC_{2n-1}^{n-1} + n(n-1)C_{2n-2}^{n-2}$
$\sum_{k=0}^{n} k^{3}(C_{n}^{k})^{2} = n^{2}C_{2n-2}^{n-1}$

$\sum_{k=0}^{n} C_k^a * C_{n-k}^b = C_{n+1}^{a+b+1}$
n拆成k个非负整数的和的方案数C_{n+k-1}^{k-1}
其生成函数为$\frac{1}{(1-x)^k}$

卡特兰数

$Cat(n) = \sum_{i=0}^{n-1} Cat(i)Cat(n-1-i) = \frac{(4n-2)Cat(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$
考虑从$(0,0)$走到$(2n,0)$,走法为$(x,y) -> (x+1,y+1),(x+1,y-1)$ 且不能走到x轴下的方案数.如果可以走到x轴下面,有C_{2n}^n种方案.对于每种走到了x轴下面的方路线,考虑第一次到达$y=-1$的位置为$(a,-1)$,把$[0,a]$这段路径关于$y=-1$对称,可以得到一条从$(0,-2)$开始的路线,而从$(0,-2)$开始的方案数为C_{2n}^{n-1},故有$Cat(n) = C_{2n}^n - C_{2n}^{n-1}$

应用

n对括号的合法序列的个数
凸n边形三角剖分方案数
n个数的出栈顺序的方案数
n个点的二叉搜索树个数
在一个$2*n$的格子中填入1到2n这些数值使得每个格子内的数值都比其右边和上边的所有数值都小的情况数
平面上连接可以形成凸包的2n个点分成2个一组连成n条线段，两两线段之间不相交的情况总数

生成函数

普通生成函数$Cat(x) = \frac{2}{1+\sqrt{1-4x}}$

斯特林数

第一类斯特林数

$s_{\underline{u}}(n,m)$ (无符号) 表示n个不同元素构成m个圆排列的方案数
$x^{\underline{n}} = x(x+1)\dots(x+n-1) = \sum_{k=0}^{n} s_u(n,k)x^k$
(该式可用于分治fft求斯特林数,正常分治法([l,r] -> [l,mid] + [mid + 1, r])慢于类快速幂分治法([1,n] -> [1,mid]))
$s_u(0,0) = 1$
$s_u(n,0) = 0$
$s_u(n,n) = 1$
$s_u(n,1) = (n-1)!$
$s_u(n,n-1) = C_n^2$
$s_u(n,2) = (n-1)!\sum_{i=1}^{n-1}\frac{1}{i}$
$s_u(n,n-2) = 2C_n^3 + 3C_n^4$

$s_u(n+1,m)=s_u(n,m-1)+n*s_u(n,m)$
$\sum_{k=0}^{n} s_u(n,k)=n!$

$s_s(n,m)$(有符号) $x^{\underline{n}}=x(x-1)\dots(x-n+1)=\sum_{k=0}^{n} s_s(n,k)x^k$
$s_s(n,m)=(-1)^{n+m}s_u(n,m)$
$\sum_{k=0}^{n} s_s(n,k)=[n=0]$
可表示n个元素组成m阶上升排列的方案数(从第一个数开始每次跳到下一个比当前数大的数的位置,能跳m次)

第二类斯特林数

$S(n,m)$ 表示将n个不同元素拆分成m个集合的方案数
$S(n,m)=\frac{1}{m!}\sum_{k=0}^{m}(-1)^k C_m^k (m-k)^n$
$S(n+1,m)=S(n,m-1)+m*S(n,m)$
$S(n,0)=[n=1]$
$S(n,1)=S(n,n)=1$
$S(n,2)=2^{n-1}-1$
$S(n,n-1)=C_n^2$
$S(n,n-2)=C_n^3+3C_n^4$
$S(n,3)=(3^{n-1}+1)/2-2^{n-1}$
$S(n,n-3)=C_n^4+10C_n^5+15C_n^6$
$\sum_{k=0}^{n} S(n,k)=B_n$ (贝尔数)

默慈金数

$M_n=M_{n-1}+\sum_{i=0}^{n-2} M_i M_{n-i-2}=\frac{(2n+1)M_{n-1}+3(n-1)M_{n-2}}{n+2}=\sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} C_n^{2i}\text{Catalan}(i)$

应用

一个圆上有n个点，以这些点为端点画出一些不相交的弦的方案数(不存在公共端点,点可以不用完,不要求弦数量最多的情况下)
平面网格上在(x,y)点只能移动到$(x+1,y-1),(x+1,y),(x+1,y+1)$,且不能移动到$y=0$以下的地方,从$(0,0)$到$(n,0)$的路径总数

生成函数

普通生成函数$M(x)=\frac{1-x-\sqrt{1-2x-3x^2}}{2x^2}=\frac{2}{1-x+\sqrt{1-2x-3x^2}}$

那罗延数

$N(n,k)=\frac{1}{n}C_n^k C_n^{k-1}$
$\sum_{i=1}^{n} N(n,k)=\text{Catalan}(n)$
在由n对(和)组成的合法字符串中，共有k对(与)相邻，这样的字符串一共有$N(n,k)$个

平面网格上在(x, y)点只能移动到$(x-1, y-1), (x+1, y+1)$,且不能移动到$y=0$以下的地方,从$(0, 0)$到$(2n, 0)$的路径中有k个山峰的路径总数
坐标轴旋转45°后,可证明其和等价于从$(0, 0)$到(n, n)的非降路径数$Catalan(n)$

贝尔数

$B_n = \sum_{i=0}^{n-1} B_i C_{n-1}^i, B_0 = 1$
$B_n = \sum_{i=1}^{n} S(n, i)$ (第二类斯特林数)

定义

集合$\{1, 2, 3, \ldots, n\}$ 的划分方案数

生成函数

指数型生成函数e^{e^x-1}

Bell三角

第一行第一个元素是1，即$a[1][1] = 1$
对于$n > 1$，第n行第一项等于第$n-1$行最后一项，即$a[n][1] = a[n-1][n-1]$
对于$m, n > 1$，第n行第m项等于它左边和左上方的两个数之和，即
$a[n][m] = a[n][m-1] + a[n-1][m-1]$
每行首项是贝尔数

同余性质

$(B[n] + B[n+1])\%p = B[p+n]\%p$,其中$p$为任意质数
$(mB[n] + B[n+1])\%p = B[p^m + n]\%p$, 其中$p$为任意质数
Bell数模素数p的循环节为$(p^p - 1)/(p-1)$

伯努利数

$B_0 = 1$
$\sum_{k=0}^{n} B_k C_{n+1}^k = 0$
$B_n = -\frac{1}{n+1} \sum_{k=0}^{n-1} B_k C_{n+1}^k$
k次方前缀和$\sum_{i=1}^{n} i^k = \frac{1}{k+1} \sum_{i=0}^{k} (-1)^i C_{k+1}^i B_i * n^{k+1-i}$

生成函数

指数型生成函数$B(x) = \frac{x}{e^x-1}$

Polya定理

Burnside引理：
对于置换群G，$c_1(a_i)$表示在a_i种置换下(一般为旋转x度，翻转什么的)的不动点个数(这个点指的是一种染色方案)，即循环节为1的点个数(即所有染色方案中在该置换下状态不变的方案数，不会通过置换变成其它状态或由其它状态变换而来)，那么本质不同的方案数为$\frac{\sum c_1(a)}{|G|}$
Polya定理：
用m种颜色对n元集G的本质不同的染色方案是$\sum \frac{m^{c(a_i)}}{|G|}$，令$c_k(a_i)$表示在第$a_i$种置换下循环节大小为$k$的循环节个数，那么$c(a_i) = \sum c_k(a_i)$，即所有循环节的个数。
$|G|$是置换集大小,a_i表示的是置换集中的所有置换,是一步可达的置换
对于颜色置换,考虑Burnside引理求不动点个数,如把颜色i替换成颜色i后相同的方案视作相同的话,可行的颜色置换需满足颜色的循环节大小能整除转置的循环节大小

梅森素数

可以表示成$2^p - 1$的素数(p是素数)
前49位梅森素数的指数的值为:
2,3,5,7,13,17,19,31,61,89,107,127,521,607,1279,2203,2281,3217,4253,4423,9689,9941,11213,19937,21701,23209,44497,86243,110503,132049,216091,756839,859433,1257787,1398269,2976221,3021377,6972593,13466917,20996011,24036583,25964951,30402457,32582657,37156667,42643801,43112609,57885161,74207281

完全数

定义:所有小于本身的因子之和等于本身的数
性质:是三角形数;其所有因子的倒数和为整数;除6外均可表示成连续奇数的立方和;均可表示为2的连续整数次幂之和;都以6或8结尾;各位数字之和辗转相加为1;除3余1,除9余1,一半除27余1
推导公式:若p是质数，$2^p - 1$也是质数，那么$(2^p - 1) * 2^{p-1}$是完全数

反素数

定义:如果n为反素数,那么对于任意一个$i(i < n)$都满足n的因子个数大于i的因子个数
性质:反素数n的质因子必然是从2开始的连续质数,且满足
$n = 2^{k1}3^{k2}5^{k3}7^{k4}\dots, k_1 \geq k_2 \geq k_3 \geq k_4 \dots$
求解方法:dfs爆搜

多对点不相交路径数

对于一张无边权的DAG方格图，给定n个起点和对应的n个终点,起点a_i对应的终点为b_i,这n条不相交路径的方案数为

$$\begin{matrix} f(a_1,b_1) & f(a_1,b_2) & f(a_1,b_3) & \cdots & f(a_1,b_h) \\ f(a_2,b_1) & f(a_2,b_2) & f(a_2,b_3) & \cdots & f(a_2,b_h) \\ f(a_3,b_1) & f(a_3,b_2) & f(a_3,b_3) & \cdots & f(a_3,b_h) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f(a_n,b_1) & f(a_n,b_2) & f(a_n,b_3) & \cdots & f(a_n,b_h) \end{matrix}$$

上述矩阵的行列式的值(其中,$f(x,y)$ 表示从x到y的方案数)

Sperner定理

集合$X = 1, 2, \ldots\ldots, n, A1, A2, \ldots\ldots Ap$ 是X 的子集,且
$\forall Ai, Aj$ 均有:$Ai \not\subseteq Aj$ 成立,称$F = A1, A2, \ldots\ldots Ap$ 为Sperner系
Sperner引理:若F 为Sperner系,则$p = |F| \le C(n, \lfloor n/2 \rfloor)$.(最大值可以取到上界)
推广:设$m = (p1)^{e1} * (p2)^{e2} \ldots\ldots (pn)^{en}$ 的度为$d = Deg(m) = e1 + e2 + \ldots\ldots en$.
若T 为m的因数组成的子集,S为T 的子集,且任意S中的两个元素没有整除关系,则$|S|$最大值为m的因子中度为$\lfloor d/2 \rfloor$ 的数个数.(可等价代换为n维空间中任意两点不存在通路的点集)

farey序列

farey序列F_n表示分母不大于n的小于等于1的分数组成的集合
对于F_n中的连续三个分数$\frac{a}{b}, \frac{c}{d}, \frac{e}{f}$， 满足$\frac{a+e}{b+f} = \frac{c}{d}$
F_n的大小等于欧拉函数和加1
可用于求解给定范围的所有最简分数

```
typedef long double LD;
typedef pair<LL,LL> PL;
//寻找分母不大于100000的最接近u的最简分数
PL find_frac(LD u,LL la,LL lb,LL ra,LL rb) {
    LL ma = la + ra,mb = lb + rb;
    if(mb > 100000) {
        if(fabs((LD)la / lb - u) < fabs((LD)ra / rb - u)) return PL(la,lb);
        else return PL(ra,rb);
    }
    LD mu = (LD)ma / mb;
    if(mu < u) return find_frac(u,ma,mb,ra,rb);
    else return find_frac(u,la,lb,ma,mb);
}
```

区间最简分数

求在区间$(\frac{a}{b}, \frac{x}{y})$ 内的最简分数(开区间)

```
void calc(LL a,LL b,LL x,LL y,LL &p,LL &q) {
    if (a == 0) {
        p = 1;
        q = y / x + 1;
        return;
    }
    LL k = y / x + 1;
    if(k * a < b) {
        q = y / x + 1;
        p = a * q / b + 1;
        return;
    }
    LL t = min(y / x, b / a);
    calc(y - t * x,x,b - t * a,a,q,p);
    q += t * p;
}
```

类欧几里德

求$\sum_{i=0}^{n} \frac{a*i+b}{c}$
其它的可类似推导

```
int ff(int a, int b, int c, LL n) {
    if(a == 0) return mul(b / c, (n + 1) % mod);
    if(a >= c || b >= c) return add(mul(b / c, (n + 1) % mod), add(mul(a / c, sum(n % mod)),
    else {
        LL m = ((__int128)a * n + b) / c;
        return add(mul(n % mod, m % mod), mod - ff(c, c - b - 1, a, m - 1));
    }
}
```

勾股定理

勾股定理的整数解：$a = (m^2 - n^2) * t, b = 2mnt, c = (m^2 + n^2) * t$,$m, n, t$为整数
当$gcd(n, m) = 1, gcd(n - m, 2) = 1$时，该解满足
$gcd(a, b) = 1, gcd(a, c) = 1, gcd(b, c) = 1$即所有的基础勾股数.
对于勾股定理这一类方程一般都有几个转移矩阵,即把最小解写成一个向量,其它解可以通过乘这些矩阵得到

马尔可夫方程

$x^2 + y^2 + z^2 = 3xyz$
它有无穷多解
若(a, b, c)是它的一组解,那么$(a, b, 3ab - c)$也是,所有解都可以由$(1, 1, 1)$通过这个迭代方式产生
另丢番图方程$x^2 + y^2 + z^2 = xyz$的每一个解都可以由一组马尔可夫方程的解(a, b, c)产生解$(3a, 3b, 3c)$
对于方程$x^2 + y^2 + z^2 = kxyz$来说,只有$k = 1$或$k = 3$时有非零解

三角平方数定理

方程$x^2 - 2y^2 = 1$的所有正整数解(x_k, y_k)满足$x_k + \sqrt{2}y_k = (3 + 2\sqrt{2})^k$
三角数为$\frac{m(m+1)}{2}$,平方数为n^2
三角平方数$n^2 = \frac{m(m+1)}{2}$的所有解满足$m_k = \frac{x_k - 1}{2}$, $n_k = \frac{y_k}{2}$
推广即为佩尔方程

复数

欧拉公式

$e^{ix} = \cos x + i\sin x$
$e^{inx} = \cos(nx) + i\sin(nx) = (\cos x + i\sin x)^n$

范数

$N(a + ib) = a^2 + b^2$,范数即复数模的平方
满足积性$N(\alpha\beta) = N(\alpha)N(\beta)$

高斯整数

复数$a + ib$中a, b为整数的复数称为高斯整数
单位高斯整数
$1, -1, i, -i$
这四个复数是高斯整数集中仅有的拥有乘法逆元的复数
单位高斯整数α满足$N(\alpha) = 1$
高斯素数
(1)$1 + i$是高斯素数
(2)设p是整数域的普通素数,且$p\%4 = 3$,则p是高斯素数
(3)设p是整数域的普通素数,且$p\%4 = 1$,则将p表示为$u^2 + v^2$,那么$u + iv$是高斯素数
若$a + ib$是高斯素数,那么$-a + ib, a - ib, -a - ib, b + ia, -b + ia, b - ia, -b - ia$都是高斯素数
规范高斯整数
满足$a + ib, (a > 0, b \geq 0)$的高斯整数称为规范高斯整数,相当于复平面上第一象限内的点集

唯一分解定理

同整数域的质因数分解一样,每一个非零高斯整数都可以唯一的分解为一个单位高斯整数乘以一些规范高斯整数的乘积

NTT

对于长度为2的幂的多项式,这个多项式的n次方

```
#include<bits/stdc++.h>
#define MAX 1000000007
using namespace std;

typedef long long LL;
const int N = 1 << 18; // N应该为2倍以上向量长度，后一半为0，不然可能会算错
const int P = 998244353; //985661441 原根3   1004535809 原根3
const int G = 3;
const int NUM = 20;

int wn[NUM];
int a[N], b[N];

int mul(int x,int y) {
    LL z = 1LL * x * y;
    return z - z / P * P;
}

int add(int x,int y) {
    x += y;
    if(x >= P) x -= P;
    return x;
}

int quick_mod(int a, int b) {
    int ans = 1;
    while(b) {
        if(b & 1) ans = mul(ans,a);
        b >>= 1;
        a = mul(a,a);
    }
    return ans;
}

void GetWn() {
```

```
    for(int i = 0; i < NUM; i++) {
        int t = 1 << i;
        wn[i] = quick_mod(G, (P - 1) / t);
    }
}

void NTT(int a[],int len,int t) {
    for (int i = 0, j = 0; i < len; i++) {
        if (i > j) swap(a[i], a[j]);
        for (int l = len >> 1; (j ^= l) < l; l >>= 1);
    }
    int id = 0;
    for(int h = 2; h <= len; h <<= 1) {
        id++;
        for(int j = 0; j < len; j += h) {
            int w = 1;
            for(int k = j; k < j + h / 2; ++k) {
                int u = a[k];
                int t = mul(w,a[k + h / 2]);
                a[k] = add(u,t);
                a[k + h / 2] = add(u,P - t);
                w = mul(w,wn[id]);
            }
        }
    }
    if(t == -1) {
        for(int i = 1; i < len / 2; i++) swap(a[i], a[len - i]);
        LL inv = quick_mod(len, P - 2);
        for(int i = 0; i < len; i++)  a[i] = mul(a[i],inv);
    }
}

void Conv(int len) {
    NTT(a,len,1);
    NTT(b,len,1);
    for(int i = 0; i < len; ++i) a[i] = mul(a[i],b[i]);
    NTT(a,len,-1);
}

//输入向量a,b 输出向量a
void work(int n1,int n2){
    GetWn();
    int i,len = 1;
```

```
    while(len < 2 * n1 || len < 2 * n2) len <<= 1;
    for(i = 0;i < len; i++){
        if(i >= n1) a[i] = 0;
        if(i >= n2) b[i] = 0;
    }
    Conv(len);
}

void NTT_2D(int a[][N],int len,int op){
    for(int i = 0;i < len; i++) NTT(a[i],len,op);
    for(int i = 0;i < len; i++){
        for(int j = i + 1;j < len; j++) swap(a[i][j],a[j][i]);
    }
    for(int i = 0;i < len; i++) NTT(a[i],len,op);
    for(int i = 0;i < len; i++){
        for(int j = i + 1;j < len; j++) swap(a[i][j],a[j][i]);
    }
}

void Conv_2D(int a[][N],int b[][N],int len){
    NTT_2D(a,len,1);
    NTT_2D(b,len,1);
    for(int i = 0;i < len; ++i){
        for(int j = 0;j < len; j++) a[i][j] = mul(a[i][j],b[i][j]);
    }
    NTT_2D(a,len,-1);
}
```

多项式各种运算

```
inline void get_dao(int n,int f[]) {
    for(int i = 0; i < n; ++i) f[i] = mul(f[i + 1],i + 1);
    f[n] = 0;
}
inline void get_fen(int n,int f[]) {
    for(int i = n - 1; i >= 0; --i) f[i] = mul(f[i - 1],inv[i]);
    f[0] = 0;
}
void get_inv(int n,int f[]) {
    static int g[N];
    if(n == 1) {
        Inv[0] = inv[f[0]];
```

```
        return;
    }
    get_inv((n + 1) >> 1,f);
    int len = n << 1;
    for(int i = 0; i < n; ++i) g[i] = f[i];
    fill(g + n,g + len,0);
    Conv(g,Inv,f1,len);
    Conv(f1,Inv,f2,len);
    for(int i = 0; i < n; i++) Inv[i] = add(mul(Inv[i],2),mod - f2[i]);
    fill(Inv + n,Inv + len,0);
}
void get_ln(int n,int f[]) {
    int len = n << 1;
    fill(Inv,Inv + len,0);
    get_inv(n,f);
    get_dao(n,f);
    Conv(f,Inv,Ln,len);
    fill(Ln + n,Ln + len,0);
    get_fen(n,Ln);
}

void get_exp(int n,int f[]) {
    static int g[N];
    if(n == 1) {
        Exp[0] = 1;
        return;
    }
    get_exp(n >> 1,f);
    int len = n << 1;
    for(int i = 0; i < n; ++i) g[i] = Exp[i];
    fill(g + n,g + len,0);
    get_ln(n,g);
    for(int i = 0; i < n; ++i) Ln[i] = add(f[i],mod - Ln[i]);
    Ln[0] = add(Ln[0],1);
    Conv(Exp,Ln,f2,len);
    for(int i = 0; i < n; i++) Exp[i] = f2[i];
    fill(Exp + n,Exp + len,0);
}



void get_sqrt(int n,int f[]) {
    static int g[N];
    if(n == 1) {
```

```
        Sqrt[0] = sqrt(f[0] + 0.5);
        return;
    }
    get_sqrt(n >> 1,f);
    int len = n << 1;
    fill(Inv,Inv + (len << 1),0);
    get_inv(n,Sqrt);
    for(int i = 0; i < n; ++i) g[i] = f[i];
    fill(g + n,g + len,0);
    Conv(g,Inv,f2,len);
    for(int i = 0;i < n; ++i) Sqrt[i] = mul(add(Sqrt[i],f2[i]),inv_2);
}
void get_pow(int len,int f[],int p) {
    fill(Ln,Ln + (len << 1),0);
    get_ln(len,f);
    for(int i = 0;i < len; ++i) f[i] = mul(p,Ln[i]);
    fill(Exp,Exp + (len << 1),0);
    get_exp(len,f);
}
int main() {
    for(len = 1;len <= n;len <<= 1);

    fill(Sqrt,Sqrt + (len << 1),0);
    get_sqrt(len,f);//开根号
    for(int i = 0;i < len; ++i) f[i] = Sqrt[i];

    fill(Inv,Inv + (len << 1),0);
    get_inv(len,f);//求逆元
    for(int i = 0;i < len; ++i) f[i] = Inv[i];

    get_fen(len,f);
    fill(f + n,f + len,0);//求积分

    fill(Exp,Exp + (len << 1),0);
    get_exp(len,f);
    for(int i = 0;i < len; ++i) f[i] = Exp[i]; //求指数幂

    fill(Ln,Ln + (len << 1),0);
    get_ln(len,f);
    for(int i = 0;i < len; ++i) f[i] = Ln[i]; //取ln

    get_pow(len,f,k);//求k次幂
```

```
}
```

FFT

```
//N应该为2倍以上向量长度，后一半为0，不然可能会算错
const int N = 1 << 18;
const double pi = acos(-1.0);

char s1[N],s2[N];
int len,res[N];

struct Complex {
    double r, i;
    Complex(double _r = 0.0, double _i = 0.0) {
        r = _r, i = _i;
    }
    Complex operator + (const Complex &b) const {
        return Complex(r + b.r, i + b.i);
    }
    Complex operator - (const Complex &b) const {
        return Complex(r - b.r, i - b.i);
    }
    Complex operator * (const Complex &b) const {
        return Complex(r * b.r - i * b.i, r * b.i + i * b.r);
    }
} va[N],vb[N];

void FFT(Complex p[], int N, int op) {
    for (int i = 0, j = 0; i < N; i++) {
        if (i > j) swap(p[i], p[j]);
        for (int l = N >> 1; (j ^= l) < l; l >>= 1);
    }
    double p0 = pi * op;
    for (int h = 2; h <= N; h <<= 1, p0 *= 0.5) {
        int hf = h >> 1;
        Complex unit(cos(p0), sin(p0));
        for (int i = 0; i < N; i += h) {
            Complex w(1.0, 0.0);
            for (int j = i; j < i + hf; j++) {
                Complex u = p[j], t = w * p[j + hf];
                p[j] = u + t;
```

```
                p[j + hf] = u - t;
                w = w * unit;
            }
        }
    }
    if (op == 1) return ;
    for (int i = 0; i < N; ++i) p[i].r /= N;
}

void Conv(Complex a[],Complex b[],int len)
{
    FFT(a,len,1);
    FFT(b,len,1);
    for(int i=0;i<len;++i) a[i] = a[i]*b[i];
    FFT(a,len,-1);
}

/*
//二维FFT
可以转化为一维 (i,j)转化为i*m+j
*/

//输入向量a,b 输出向量res
void work(int a[],int b[],int n1,int n2){
    int i,len = 1;
    while(len < 2 * n1 || len < 2 * n2) len <<= 1;
    for(i = 0;i < n1; ++i){
        va[i].r = a[i];
        va[i].i = 0;
    }
    while(i < len){
        va[i].r = va[i].i = 0;
        ++i;
    }
    for(i = 0;i < n2; ++i){
        vb[i].r = b[i];
        vb[i].i = 0;
    }
    while(i < len){
        vb[i].r = vb[i].i = 0;
        ++i;
    }
    Conv(va,vb,len);
```

```
    memset(res,0,sizeof res);
    for(int i=0;i<len;++i) res[i]=va[i].r + 0.5;
}
```

分治FFT

已知序列a和$f[k] = \sum_{i=0}^{k-1} f[i]a[k-i]$,求序列f

```
int a[maxn],f[maxn];
void cdq(int l,int r) {
    if(r - l <= 20) {
        for(int i = l + 1;i <= r; i++) {
            int tmp = 0;
            for(int j = l;j < i; j++) tmp = add(tmp,mul(f[j],a[i - j]));
            f[i] = add(f[i],tmp);
        }
        return;
    }
    int mid = (l + r) >> 1;
    cdq(l,mid);
    int nn = r - l + 1,mm = 1;
    while(mm <= nn) mm <<= 1;
    for(int i = 0;i < mid - l + 1; i++) va[i] = Complex{(double)f[i + l],0.0};
    for(int i = mid - l + 1;i < mm; i++) va[i] = Complex{0.0,0.0};
    vb[0] = Complex{0.0,0.0};
    for(int i = 1;i < nn; i++) vb[i] = Complex{(double)a[i],0.0};
    for(int i = nn;i < mm; i++) vb[i] = Complex{0.0,0.0};
    Conv(va,vb,mm);
    for(int i = mid + 1;i <= r; i++) f[i] = add(f[i],fmod(va[i - l].r + 0.5));
    cdq(mid + 1,r);
}
```

任意模数FFT

```
//任意模数fft,模数在int范围内,数组长度1e5可用
//该fft写法用于一般情况下的fft会出错

const int LN = 18;
const int N = 1 << LN;
const double pi = acos(-1.0);
int fa[N],fb[N],fc[N],mod;
```

```
int fmod(LL x) {
    return x - x / mod * mod;
}

int mul(int x,int y) {
    return fmod(1LL * x * y);
}

int add(int x,int y) {
    x += y;
    if(x >= mod) x -= mod;
    return x;
}

struct Complex {
    double r,i;
    Complex(double r = 0.0,double i = 0.0):r(r),i(i) {};
    Complex operator + (const Complex &rhs) {
        return Complex(r + rhs.r,i + rhs.i);
    }
    Complex operator - (const Complex &rhs) {
        return Complex(r - rhs.r,i - rhs.i);
    }
    Complex operator * (const Complex &rhs) {
        return Complex(r * rhs.r - i * rhs.i,i * rhs.r + r * rhs.i);
    }
    Complex conj() {
        return Complex(r,-i);
    }
} wn[N];
int bitrev[N];

void fft_prepare() {
    for(int i = 0; i < N; i++) bitrev[i] = bitrev[i >> 1] >> 1 | ((i & 1) << (LN - 1));
    for(int i = 0; i < N; i++) wn[i] = Complex(cos(2 * pi * i / N),sin(2 * pi * i / N));
}

void FFT(Complex a[],int l,int op) {
    int d = 0;
    while((1 << d) * l != N) d++;
    for(int i = 0; i < l; i++) {
        if(i < (bitrev[i] >> d)) swap(a[i],a[(bitrev[i] >> d)]);
    }
```

```
    for (int i = 2; i <= l; i <<= 1){
        int lyc = N / i;
        for (int j = 0; j < l; j += i) {
            Complex *l = a + j, *r = a + j + (i >> 1), *p = wn;
            for(int k = 0; k < (i >> 1); k++) {
                Complex tmp = *r * *p;
                *r = *l - tmp, *l = *l + tmp;
                ++l, ++r, p += lyc;
            }
        }
    }
    if(op == -1) {
        for(int i = 0;i < l; i++) {
            a[i].r /= l;
            a[i].i /= l;
        }
    }
}

Complex va[N],vb[N],da[N],db[N],dc[N],dd[N];
void Conv(int fa[],int fb[],int fc[],int l) {
    int i;
    for(i = 0; i < l; i++) va[i] = Complex(fa[i] & 32767,fa[i] >> 15);
    for(i = 0; i < l; i++) vb[i] = Complex(fb[i] & 32767,fb[i] >> 15);
    FFT(va,l,1);
    FFT(vb,l,1);
    for(i = 0; i < l; i++) {
        int j = (l - i) & (l - 1);
        Complex qa,qb,qc,qd;
        qa = (va[i] + va[j].conj()) * Complex(0.5,0.0);
        qb = (va[i] - va[j].conj()) * Complex(0.0,-0.5);
        qc = (vb[i] + vb[j].conj()) * Complex(0.5,0.0);
        qd = (vb[i] - vb[j].conj()) * Complex(0.0,-0.5);
        da[j] = qa * qc;
        db[j] = qa * qd;
        dc[j] = qb * qc;
        dd[j] = qb * qd;
    }
    for(i = 0; i < l; i++) va[i] = da[i] + db[i] * Complex(0.0,1.0);
    for(i = 0; i < l; i++) vb[i] = dc[i] + dd[i] * Complex(0.0,1.0);
    FFT(va,l,-1);
    FFT(vb,l,-1);
    for(i = 0; i < l; i++) {
```

```
        int wa = fmod(va[i].r + 0.5);
        int wb = fmod(va[i].i + 0.5);
        int wc = fmod(vb[i].r + 0.5);
        int wd = fmod(vb[i].i + 0.5);
        fc[i] = add(wa,add(fmod(((LL)add(wb,wc)) << 15),fmod(((LL)wd) << 30)));
    }
}

void solve() {
    fft_prepare();
    //输入数组fa，fb,长度为l,返回fc
    Conv(fa,fb,fc,l);
}
```

多项式是否可以约分，即拆成两个多项式乘积

给出整系数多项式 $f(x) = \sum_{k=0}^{n} a_k x^k$ ，如果存在素数 p，使得 p 不整除 a_n ,但整除其它 a_k ,且 p^2 不整除 a_0 ，那么 $f(x)$ 不可约

分圆多项式

$x^n - 1 = 0$在复数域上有n个解，分别为$\omega_n^k = e^{\frac{2\pi ik}{n}}$
定义分圆多项式$\Phi_n(x) = \prod_{gcd(n,k)=1}(x - \omega_n^k)$
那么有$\prod_{d|n} \Phi_d(x) = \prod_{1\le k\le n}(x - \omega_n^k) = x^n - 1$
可以证明,分圆多项式是不可约分的整系数多项式.
由莫比乌斯反演,可得$\Phi_n(x) = \prod_{d|n}(x^d - 1)^{\mu(\frac{n}{d})}$
如果n是质数,那么有$\Phi_n(x) = \sum_{i=0}^{n-1} x^i$

拉格朗日插值法

对于n次多项式$f(x)$，已知$n + 1$个点值$f(x_0), f(x_1) \cdots f(x_n)$,那么可以构造
$l_k(x) = \prod_{i=0,i\neq k}^{n} \frac{x-x_i}{x_k-x_i}$,于是有$f(x) = \sum f(x_k) l_k(x)$

重心型拉格朗日插值公式

令$w_j = \frac{1}{\prod_{i\neq j}(x_j - x_i)}$
得

$$f(x) = \frac{\sum \frac{w_j}{x-x_j} f(x_j)}{\sum \frac{w_j}{x-x_j}}$$

牛顿插值法

由线性代数知识可知,任意一个n次多项式都可表示为
$f(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1) \quad , \ldots (x - x_{n-1})$
该多项式称为牛顿插值多项式,其中$a_i = g[x_0, x_1, \ldots, x_{i+1}]$,
$g[x_i, x_{i+1}, \ldots, x_{i+j+1}] = \frac{g[x_i, x_{i+1}, \ldots, x_{i+j}] - g[x_{i+1}, x_{i+2}, \ldots, x_{i+j+1}]}{x_i - x_{i+j+1}}$, $g[x_i] = f(x_i)$
可以证明,该系数与拉格朗日插值法的系数等价.
定义差分$\Delta y_i = y_{i+1} - y_i, \Delta^m y_i = \Delta^{m-1} y_{i+1} - \Delta^{m-1} y_i$
若已知的点值存在$x_i = x_0 + ki$,那么有$a_i = \frac{\Delta^i y_0}{i! k^i}$
特别的,若$k = 1$,则有$\Delta^i y_0 = \sum_{j=0}^{i} (-1)^{i-j} C_i^j y_j$,这就是函数的i次差分

牛顿迭代法

设u是$f(x) = 0$ 的根，选x_0作为u的初始近似值，过点$(x_0, f(x_0))$ 作曲线$y = f(x)$ 的切线L,求出L与x轴的交点$(x_1, 0)$,然后求出关于x_1的切线重复上述过程，可求得u的n次近似值x_n.

拉格朗日反演

若$f(x), g(x)$ 可以表示成多项式,且满足$f(g(x)) = x$,那么有
$[x^n]g(x) = \frac{1}{n}([x^{-1}]\frac{1}{f(x)^n})$
推广:在满足上述条件下有$[x^n]h(g(x)) = \frac{1}{n}[x^{-1}](h'(x)\frac{1}{f(x)^n})$
其中$[x^n]f(x)$ 表示x^n项的系数
显然有公式$[x^a](f(x) * x^b) = [x^{a-b}]f(x)$

生成正态分布

```
default_random_engine generator(time(NULL));//产生种子
normal_distribution<double> distribution(0.0,1.0);//形成分布第一个为均值，第二个为方差
uniform_int_distribution //整数范围的均匀分布
uniform_real_distribution //实数范围的均匀分布
cout<<distribution(generator)<<endl;//输出随机数
```

组合数取模

```
//lucas可推广到多项式系数n!/(a!b!c!.....)  a+b+c+.....=n
long long m,q[100010];//模数为m
long long pow(long long a,long long b){
```

```
    long long r=1,base=a;
    while(b!=0){
        if(b&1) r=r*base%m;
        base=base*base%m;
        b>>=1;
    }
    return r;
}

long long gcd(long long a,long long b){
    if(b==0) return a;
    else return gcd(b,a%b);
}

long long cc(long long a,long long b){
    if(a<b) return 0;
    return q[a]*pow(q[b]*q[a-b]%m,m-2)%m;
}

long long Lucas(long long a,long long b)//输入a为底的组合数{
    if(b==0)    return 1;
    return (cc(a%m,b%m)*Lucas(a/m,b/m))%m;
}


//任意模数取模
int powt(int a, int b, int mod) {
    int r = 1;
    while(b) {
        if(b & 1) r = mul(r, a, mod);
        a = mul(a, a, mod);
        b >>= 1;
    }
    return r;
}

void extend_Euclid(int a, int b, int &x, int &y) {
    if(b == 0) {
        x = 1;
        y = 0;
        return;
    }
    extend_Euclid(b, a % b, x, y);
```

```
    int tmp = x;
    x = y;
    y = tmp - (a / b) * y;
}

int Inv(int a, int b) {
    int x, y;
    extend_Euclid(a, b, x, y);
    return (x % b + b) % b;
}

int CRT(int a[], int m[], int n, int M) {
    int ans = 0;
    for(int i = 0; i < n; i++) {
        int x, y;
        int Mi = M / m[i];
        extend_Euclid(Mi, m[i], x, y);
        ans = (ans + 1LL * Mi * x * a[i]) % M;
    }
    if(ans < 0) ans += M;
    return ans;
}

int c(int n,int p,int mod){
    if(n == 0) return 1;
    int i,s = 1,d = 1,r = n % mod;
    for(i = 2;i < mod; i++){//如果模数固定，这一部分可以预处理，用时将减少很多
        if(i % p) d = d * i % mod;
        if(i == r) s = d;
    }
    return s * powt(d,n / mod,mod) % mod * c(n / p,p,mod) % mod;
}

int cal(int n,int m,int p,int k,int mod){
    int x,s,t = 0;
    x = n;
    while(x) x /= p,t += x;
    x = m;
    while(x) x /= p,t -= x;
    x = n - m;
    while(x) x /= p,t -= x;
    if(t >= k) return 0;
    else s = powt(p,t,mod);
```

```
    s = s * c(n,p,mod) % mod * Inv(c(m,p,mod),mod) % mod * Inv(c(n - m,p,mod),mod) % mod;
    return s;
}

int exlucas(int n,int m,int mod){
    if(m > n) return 0;
    int i,t,x,k = 0;
    int a[105],b[105];
    for(x = mod,i = 2;i * i <= x; i++){
        if(x % i == 0){
            t = 0;
            b[k] = x;
            while(x % i == 0){
                t++;
                x /= i;
            }
            b[k] /= x;
            a[k] = cal(n,m,i,t,b[k]);
            k++;
        }
    }
    if(x > 1){
        b[k] = x;
        a[k] = cal(n,m,b[k],1,b[k]);
        k++;
    }
    return CRT(a,b,k,mod);
}
```

BSGS

BSGS算法方法如下：
求解$a^x \equiv b\%p$
令$x = im - j, m = ceil(\sqrt{p})$，则$a^{im-j} \equiv b(\%p)$
移项，得$(a^m)^i \equiv b * a^j (\%p)$
首先，从$0 - m$枚举j，将得到的$b * a^j$的值存入hash表；
然后，从$1 - m$枚举i，计算$(a^m)^i$，查表，如果有值与之相等，则当时得到的$im - j$是最小值。
方程有解的充要条件是p为质数且$gcd(a, p) = 1$
如果p不是质数,改写成$a^{x-1} * a \equiv b\%p$
设$d = gcd(a, p)$,有$a^{x-1} * \frac{a}{d} \equiv \frac{b}{d} \% \frac{p}{d}$
此时可能存在$gcd(a, \frac{p}{d}) \neq 1$
递归下去,继续除去gcd

直到$gcd(a, \frac{p}{\prod_{i=1}^{k} d_i}) = 1$

若中途存在gcd不能整除b,则只存在$x = 0$的解

否则,先暴力判断$[0, k]$范围内的解

然后得新方程$\frac{a^k}{d} * a^x \equiv \frac{b}{d} \% \frac{p}{d}, d = \prod_{i=1}^{k} d_i$

此时存在逆元可以用BSGS求解

BSGS适用于矩阵,要求矩阵有逆,矩阵有逆的充要条件是$gcd(|A|, p) = 1$,行列式的值与模数的$gcd = 1$

连分数

$$\omega = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \cdots}}$$

一个实数ω可以用一个正整数数列$[a_i]$表示，这样的表示方法称为连分数表示.有理数的连分数序列有限，无理数的连分数序列无限.

给定$[a_i]$数列，求原数$\frac{p_n}{q_n}$的递推公式：

$p_n = a_n p_{n-1} + p_{n-2}, p_{-1} = 1, p_{-2} = 0$

$q_n = a_n q_{n-1} + q_{n-2}, q_{-1} = 0, q_{-2} = 1$

一个实数的序列长度为n连分数表示亦称为一个数的n渐近分数.

佩尔(Pell)方程

$x^2 - Ny^2 = 1$

佩尔方程有解的充要条件是正整数N不是完全平方数

定理：若$\sqrt{N}$的渐近分数为$\frac{p_k}{q_k}$,则存在$n > 0$，使得$p_n^2 - Nq_n^2 = 1$

递推寻找最小解：

初始化：

$p_{-1} = 1, p_{-2} = 0$

$q_{-1} = 0, q_{-2} = 1$

$a_0 = \lfloor \sqrt{N} \rfloor$

$g_{-1} = 0, h_{-1} = 1$

递推寻找：

$g_i = -g_{i-1} + a_i h_{i-1}$

$h_i = \frac{N - g_i^2}{h_{i-1}}$

$a_{i+1} = \lfloor \frac{g_i + a_0}{h_i} \rfloor$

$p_i = a_i p_{i-1} + p_{i-2}$

$q_i = a_i q_{i-1} + q_{i-2}$

然后check p_i, q_i

如果求得了最小解x_1, y_1,那么可以递推出第k大解：

$x_k = x_{k-1} x_1 + N y_{k-1} y_1$

$y_k = x_{k-1} y_1 + y_{k-1} x_1$

并且第k大解(x_k, y_k)满足$x_k + \sqrt{N} y_k = (x_1 + \sqrt{N} y_1)^k$
所以当k比较大时,可以使用二元域的快速幂(代码见二次剩余部分)求解

第二类佩尔方程

$x^2 - Ny^2 = -1$
如果N并且N为质数,那么一定有解
若上式有解,设其最小解为(x_0, y_0),那么上式的所有解(x_k, y_k)满足
$x_k + y_k\sqrt{N} = (x_0 + y_0\sqrt{N})^{2k+1}$
设$x^2 - Ny^2 = 1$的最小解为(a, b),满足$a + b\sqrt{N} = (x_0 + y_0\sqrt{N})^2$(猜测这是第二类佩尔方程有解的充要条件)

二次互反定理

勒让德符号 (a/p)= 1,a是模p的二次剩余；-1，a不是模p的二次剩余;0，a是p的倍数
勒让德符号是积性函数
设p是奇素数，则有 a^((p-1)/2) = (a/p) mod p
~~可以用三次剩余代替~~$\sqrt{x}$
求解$x^2 = n\%p$,p为奇素数:
令$w = a^2 - n$是p的非二次剩余,则有$(a + \sqrt{w})^p = a - \sqrt{w}\%p$,故有
$(a + \sqrt{w})^{p+1} = a^2 - w = n\%p$,所以$x = (a + \sqrt{w})^{(p+1)/2}$,

求$x^2 = n\%p^m$
设r是方程$x^2 = n\%p$的解
有$r^2 - a = k * p, (r^2 - a)^m = (k * p)^m, (r^2 - a)^m\%p^m = 0$
因为$(r + \sqrt{a})^m = t + u\sqrt{a}, (r - \sqrt{a})^m = t - u\sqrt{a}$
那么有$(r^2 - a)^m \equiv t^2 - u^2 a \equiv 0\%p^m$
故$x \equiv t * Inv(u)\%p$

```
LL mod,wf;
default_random_engine generator(time(NULL));
uniform_int_distribution<LL> df(1,1e18);

LL add(LL x,LL y) {
    x += y;
    if(x >= mod) x -= mod;
    return x;
}

LL mul(LL a,LL b) {
    LL r = (a * b - (LL)(((long double)a * b) / mod) * mod);
    return add(r - r / mod * mod,mod);
}
```

```
LL powt(LL a,LL b) {
    LL r = 1;
    while(b) {
        if(b & 1) r = mul(r,a);
        a = mul(a,a);
        b >>= 1;
    }
    return r;
}

struct node {
    LL p,d;
    node operator * (const node &b) const {
        node r;
        r.p = add(mul(p,b.p),mul(mul(d,b.d),wf));
        r.d = add(mul(p,b.d),mul(d,b.p));
        return r;
    }
};
//二元域快速幂
node powt(node a,LL b) {
    node r = node{1,0};
    while(b) {
        if(b & 1) r = r * a;
        a = a * a;
        b >>= 1;
    }
    return r;
}
//欧拉准则判断勒让德符号
//返回1表示是二次剩余，返回mod-1表示不是
LL Legendre(LL a) {
    return powt(a,(mod - 1) >> 1);
}

//递归求雅可比符号法，-1表示不是二次剩余，但是1并不表示是二次剩余关系
LL Legendre(LL a,LL p) {
    if(a == 1) return 1;
    if(a % p == 0) return 0;
    if(a & 1) {
        LL u = ((p - 1) & 15) * ((a - 1) & 15);
        int k = ((u >> 2) & 1) ? -1 : 1;
```

```
        return Legendre(p % a,a) * k;
    } else {
        LL u = ((p - 1) & 31) * ((p + 1) & 31);
        int k = ((u >> 3) & 1) ? -1 : 1;
        return Legendre(a >> 1,p) * k;
    }
}
//不是二次剩余返回-1
LL get_ans(LL x) {
    if(powt(x,(mod - 1) >> 1) + 1 == mod) return -1;
    LL a;
    while(true) {
        a = df(generator) % mod;
        wf = add(mul(a,a),mod - x);
        if(Legendre(wf,mod) == -1) break;
    }
    node tmp = node{a,1};
    return powt(tmp,(mod + 1) >> 1).p;
}
```

丢番图方程

费马平方和定理:奇素数能表示为两个平方数之和的充要条件是该素数模4余1
一个正整数能表示成两个平方数之和的充要条件是它的质因数分解式中,形如$4k + 3$的质因子的指数为偶数.
若要使得$n = a^2 + b^2, gcd(a, b) = 1$那么n的质因子中不含形如$4k + 3$的质因子.

勒让德平方和定理:
设$D_1 = \sum_{d|n}[d\%4 = 1]$,$D_3 = \sum_{d|n}[d\%4 = 3]$
那么将n表示成两数平方和的方案数为$R(n) = 4(D_1 - D_3)$

Brahmagupta–Fibonacciidentity如果两个数都能表示成两个平方数之和,那么它们的积也能表示成两个平方数之和.$(u^2 + v^2) * (x^2 + y^2) = (ux + vy)^2 + (ux - vy)^2$
任意正整数都能表示成4个整数的平方和.
一个正整数能表示成3个平方数之和的充要条件是它不能表示成$4^m(8k + 7)$的形式.(m,k为非负整数)

x*x+y*y=n的整数解个数

$ans = 4\sum_{d|n} H(d)$
$H(d) = 0, d为偶数; (-1)^{\frac{d-1}{2}}, d为奇数$
做法:分解n,枚举因子

x*x+y*y=p(p为质数)的解

特判$p = 2$
当且仅当$p\%4 = 1$时有解，且解唯一
先求出-1关于p的二次剩余x_0,对p和x_0进行辗转相除,记录余数r(初始余数为x_0),当$r < \sqrt{p}$时结束,此时r为丢番图方程$x^2 + y^2 = n$中的x,而$y = \sqrt{n - x^2}$

费马降阶法:先求出-1关于p的二次剩余x,有$x^2 + 1 = m * p$
然后反复降m的值直到为1
若当前为$x^2 + y^2 = mp$
取$u = \min(x, m - x), v = \min(y, m - y)$
得新值$x = (ux + vy)/m, y = (vx - uy)/m, m = (u^2 + v^2)/m$
最多$\log$次

如果p不是质数,根据费马平方和定理……将p的次数为奇数的质因子提出来求解,然后根据Brahmagupta−Fibonacciidentity合并

```
//求方程的解
void Solve(LL p, LL &_x, LL &_y)  {
    LL x = Work(p - 1, p); //求解二次剩余
    if(x >= p - x) x = p - x;
    LL t = p;
    LL r = x;
    LL limit = (LL)sqrt(1.0 * p);
    while(r > limit)  {
        x = r;
        r = t % x;
        t = x;
    }
    _x = r;
    _y = (LL)sqrt((p - r * r));
    if(_x > _y) swap(_x, _y);
}

// 费马降阶法
void Solve(int p, LL &x, LL &y) {
    x = Work(p - 1, p);
    y = 1;
    int m = (x * x + y * y) / p;
    while(m > 1) {
        LL _x = x % m,_y = y % m;
        if(_x * 2 > m) _x -= m;
        if(_y * 2 > m) _y -= m;
        if(abs(_x * x + _y * y) % m) swap(_x,_y);
        LL u = (_x * x + _y * y) / m;
        LL v = (_y * x - _x * y) / m;
```

```
        m = (u * u + v * v) / p;
        x = abs(u);
        y = abs(v);
    }
    if(x > y) swap(x,y);
}
```

原根

若m存在原根，则有φ(φ(m))个原根
如果a是m的原根,则$a^x \equiv 1\%m$的最小正整数解为φ(m)
如果a是m的原根,则$a^1, a^2, \dots, a^{\varphi(m)-1}$ 互不相同,并构成了%m的简化剩余系(即包含了所有与m互质的数)
存在原根的数:$2, 4, p^a, 2p^a$,a为大于一的数,p为任意素数

寻找原根：1)暴力，2)若模数为p,可对φ (p)质因数分解，求出所有质因子p_i,若对所有的质因子满足$g^{\frac{\varphi(p)}{p_i}} \neq 1\%p$, 那么g为原根，原根一般都小于100

线性同余方程

裴蜀定理:若要使得$ax\%m = b$有解,则必有$\gcd(a, m) \mid b$
裴蜀定理可以拓展到多元线性同余方程,即$\sum a_i x_i \%m = b$有解必须满足$\gcd(a_i, m) \mid b$
由裴蜀定理，可以得到满足$kx + a\%m = ky + b\%m$的条件是$\gcd(k, m) \mid (a - b)$

线性丢番图方程是否存在非负整数解

线性丢番图方程($\sum a_i x_i = n$) 是否存在解只需要用裴蜀定理判定
对于二元方程$(ax + by = c)$,利用扩展欧几里德求出一组解(x, y), 将x归约到最小非负值,然后判断$ax \le c$是否成立即可
若a_i比较小，那么可以建立模a_i意义下的最短路，用d[i]表示模a余i的数中能被表示出来的最小的数，跑最短路即可.

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

```
//ax%m=b
int get_modans(int a,int m,int b){
    if(b == 0) return 0;
    int d = __gcd(a,m);
    if(b % d) return MAX;//无解
    int x,y;
    extend_Euclid(a / d,m / d,x,y);
    x = 1LL * x * b / d % m;
    x = (x % m + m) % m;
//    x %= m / d;  //求非负最小解
    return x;
}
```

中国剩余定理

如果方程组的模数互质,则解唯一
故有推论若$f(x) \equiv d\%a\ (0 \le x < a)$ 的解的个数为w_1,$f(x) \equiv d\%b\ (0 \le x < b)$的解的个数为$w_2$,且$gcd(a, b) = 1$那么$f(x) \equiv d\%ab\ (0 \le x < ab)$的解的个数为$w_1 w_2$

```
#include<bits/stdc++.h>
#define MAX 1000000007
using namespace std;
typedef long long LL;
const int N = 1005;

LL a[N], m[N];

LL gcd(LL a,LL b){
    return b? gcd(b, a % b) : a;
}

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

```
}

LL Inv(LL a, LL b){
    LL d = gcd(a, b);
    if(d != 1) return -1;
    LL x, y;
    extend_Euclid(a, b, x, y);
    return (x % b + b) % b;
}

bool merge(LL a1, LL m1, LL a2, LL m2, LL &a3, LL &m3){
    LL d = gcd(m1, m2);
    LL c = a2 - a1;
    if(c % d) return false;
    c = (c % m2 + m2) % m2;
    m1 /= d;
    m2 /= d;
    c /= d;
    c *= Inv(m1, m2);
    c %= m2;
    c *= m1 * d;
    c += a1;
    m3 = m1 * m2 * d;
    a3 = (c % m3 + m3) % m3;
    return true;
}

LL CRT(LL a[], LL m[], int n){
    LL a1 = a[1];
    LL m1 = m[1];
    for(int i=2; i<=n; i++){
        LL a2 = a[i];
        LL m2 = m[i];
        LL m3, a3;
        if(!merge(a1, m1, a2, m2, a3, m3))
            return -1;
        a1 = a3;
        m1 = m3;
    }
    return (a1 % m1 + m1) % m1;
}

int main(){
```

```
    int i,n;
    scanf("%d",&n);
    for(i = 1; i <= n; i++)
        scanf("%lld%lld",&m[i], &a[i]);
    LL ans = CRT(a, m, n);
    printf("%lld\n",ans);
    return 0;
}

//模数互质版本
LL CRT(int a[], int m[], int n) {
    LL M = 1, ans = 0;
    for(int i = 1; i <= n; i++) M *= m[i];
    for(int i = 1; i <= n; i++) {
        LL x, y, Mi = M / m[i];
        extend_Euclid(Mi, m[i], x, y);
        ans = fadd(ans, fmul(fmul(Mi, x + M, M), a[i]));
    }
    if(ans < 0) ans += M;
    return ans;
}
```

gcd快速询问

该程序可实现1e6范围内数的gcd询问

```
const int maxn = 1e6;
const int Sqrt_N = 1e3;
int pre[maxn + 1], decomp[maxn + 1][3], dp[Sqrt_N + 1][Sqrt_N + 1];

//询问
int Ask( int a, int y ) {
    int x[3], g = 1 ;
    for(int i = 0 ; i < 3;  ++ i) x[i]=decomp[a][i];
    for(int i = 0 ; i < 3 ; ++ i) {
        int d = 1;
        if( x[i] <= Sqrt_N ) d = dp[x[i]][y % x[i]];
        else if( y % x[i] == 0 ) d = x[i];
        g *= d;
        y /= d;
    }
    return g;
```

```
}
//预处理
void Construction() {
    for(int i = 2 ; i <= maxn ; ++ i)
        if(!pre[i])
            for(int j = i ; j <= maxn ; j += i) if( !pre[j] ) pre[j] = i;
    decomp[1][0]=decomp[1][1]=decomp[1][2]=1;
    for(int i = 2 ; i <= maxn ; ++ i) {
        int p = pre[i], pos = 0;
        for(int j = 0 ; j < 3 ; ++ j) {
            decomp[i][j]=decomp[i/p][j];
            if(decomp[i][j]<decomp[i][pos]) pos = j;
        }
        decomp[i][pos]*=p;
    }
    for(int i = 1 ; i <= Sqrt_N ; ++ i) dp[i][i] = dp[i][0] = dp[0][i] = i;
    for(int i = 1 ; i <= Sqrt_N ; ++ i)
        for(int j = 1 ; j <= i - 1 ; ++ j)
            dp[i][j] = dp[j][i] = dp[i - j][j];
}
```

斐波拉契相关

通项公式: $f_n = \frac{1}{\sqrt{5}}[(\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n]$

1.gcd(fib(n),fib(m))=fib(gcd(n,m))
证明：可以通过反证法先证**fibonacci**数列的任意相邻两项一定互素，然后可证**n>m**时，
gcd(fib(n),fib(m))=gcd(fib(n-m),fib(m))，递归可
求**gcd(fib(n),fib(m))=gcd(fib(k),fib(l))**，最后**k=l**，不然继续递归。
K是通过展转相减法求出，易证**k=gcd(n,m)**，
所以**gcd(fib(n),fib(m))=fib(gcd(n,m))**。
2.如果**fib(k)**能被**x**整除，则**fib(k*i)**都可以被**x**整除。
3.f(0)+f(1)+f(2)+…+f(n)=f(n+2)-1
4.f(1)+f(3)+f(5)+…+f(2n-1)=f(2n)
5.f(2)+f(4)+f(6)+…+f(2n) =f(2n+1)-1
6.[f(0)]^2+[f(1)]^2+…+[f(n)]^2=f(n)·f(n+1)
7.f(0)-f(1)+f(2)-…+(-1)^n·f(n)=(-1)^n·[f(n+1)-f(n)]+1
8.f(n+m)=f(n+1)·f(m)+f(n)*f(m-1)
9.[f(n)]^2=(-1)^(n-1)+f(n-1)·f(n+1)
10.f(2n-1)=[f(n)]^2-[f(n-2)]^2
11.3f(n)=f(n+2)+f(n-2)
12.f(2n-2m-2)[f(2n)+f(2n+2)]=f(2m+2)+f(4n-2m) [n〉 m≥-1,且n≥1]

```
13.f(2n+1)=[f(n)]^2+[f(n+1)]^2
14.f(i)f(i-1)-f(i+1)f(i-2)=(-1)^i
```

循环节:
求斐波那契数模n的循环节
把n质因数分解为$n = \prod p_i^{a_i}$
分别求出模$p_i^{a_i}$的循环节x_i
那么模n的循环节就是$lcm(x_i)$

设$G(p)$是模p的循环节
那么模p^a的循环节等于$G(p) * p^{a-1}$
如果5是p的二次剩余,循环节是$p - 1$的因子,否则是$2(p + 1)$的因子,dfs暴力求解即可

```
LL get_cycle(int a) {
    利用矩阵模质数循环节公式暴力求解
    matrix r = {a, 1, 1, 0};//转移矩阵
    matrix u = {1, 0, 0, 1};//单位矩阵
    fac.clear();//质因子清零
    get_fac(mod - 1);//添加质因子
    get_fac(mod + 1);
    fac.push_back(mod);
    int n = fac.size();
    LL d = 1;
    for(int i = 0; i < n; i++) {
        matrix v = r;
        for(int j = i + 1; j < n; j++) v = powt(v, fac[j]);
        if(u == v) continue;
        d *= fac[i];
        r = powt(r, fac[i]);
    }
    return d;//返回最小循环节大小
}
```

FWT

若二元运算中带有bit位限制，如子集卷积中为i ^ j = k[bit(i) + bit(j) = bit(k)] //bit(i)为i中1的个数
可以增加一维，令f[bit(i)][i] = a[i]，g[bit(i)][i] = b[i],对f[i],g[i]分别进行fwt，然后得h[k][i] = f[x][i] * g[k - x][i]，再对h[i]进行ufwt，h[bit(i)][i]即是答案

```
void FWT(int a[],int n)    {
    for(int d = 1;d < n;d <<= 1)
        for(int m = d << 1,i = 0;i < n;i += m)
            for(int j = 0;j < d; j++)    {
```

```
                int x = a[i + j],y = a[i + j + d];
                a[i + j] = (x + y) % mod,a[i + j + d]=(x - y + mod) % mod;
                //xor:a[i+j]=x+y,a[i+j+d]=(x-y+mod)%mod;
                //and:a[i+j]=x+y;
                //or:a[i+j+d]=x+y;
            }
}

void UFWT(int a[],int n)    {
    for(int d = 1;d < n;d <<= 1)
        for(int m = d << 1,i = 0;i < n;i += m)
            for(int j = 0;j < d; j++)     {
                int x = a[i + j],y = a[i + j + d];
                a[i + j] = 1LL * (x + y) * rev % mod;
                a[i + j + d] = (1LL * (x - y) * rev % mod + mod) % mod;
                //xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;
                //and:a[i+j]=x-y;
                //or:a[i+j+d]=y-x;
            }
}
void solve(int a[],int b[],int n)    {
    FWT(a,n);
    FWT(b,n);
    for(int i = 0;i < n; i++) a[i] = 1LL * a[i] * b[i] % mod;
    UFWT(a,n);
}
```

素数测试

小于4 759 123 141 用2,7,61
小于341 550 071 728 320 用2, 3, 5, 7, 11, 13和17
在1e16范围内用2, 3, 7, 61和24251只有46 856 248 255 981会判错

```
LL add(LL x,LL y,LL m) {
    x += y;
    if(x >= m) x -= m;
    return x;
}

LL mul(LL a,LL b,LL m) {
    LL r = (a * b - (LL)(((long double)a * b) / m) * m);
    return add(r - r / m * m,m,m);
```

```
}

LL powt(LL a,LL b,LL m) {
    LL r = 1;
    while(b) {
        if(b & 1) r = mul(r,a,m);
        a = mul(a,a,m);
        b >>= 1;
    }
    return r;
}

bool flag[maxn];
int pri[maxn],cnt = 0;
void GetPrime() {
    for (int i = 2; i < maxn; ++i) {
        if (!flag[i]) pri[cnt++] = i;
        for (int j = 0; j < cnt && pri[j] * i < maxn; ++j) {
            flag[pri[j] * i] = 1;
            if (i % pri[j] == 0) break;
        }
    }
}

LL Rand(LL n) {
    return (((LL)rand()<<31)|rand())%(n-1)+1;
}

bool test(LL n,LL a,LL d) {
    if(n == 2) return true;
    if(n == a) return true;
    if(!(n & 1)) return false;
    while(!(d & 1)) d >>= 1;
    LL t = powt(a,d,n);
    while(d != n - 1 && t != n - 1 && t != 1) {
        t = mul(t,t,n);
        d <<= 1;
    }
    return t == n - 1 || (d & 1) == 1;
}

bool isprime(LL n) {
    for(int i = 0; i < 6; i++) {
```

```
        if(!test(n,pri[Rand(cnt) - 1],n - 1)) return false;
    }
    return true;
}
```

大质因数分解

```
LL add(LL x,LL y,LL m) {
    x += y;
    if(x >= m) x -= m;
    return x;
}

LL mul(LL a,LL b,LL m) {
    LL r = (a * b - (LL)(((long double)a * b) / m) * m);
    return add(r - r / m * m,m,m);
}

LL powt(LL a,LL b,LL m) {
    LL r = 1;
    while(b) {
        if(b & 1) r = mul(r,a,m);
        a = mul(a,a,m);
        b >>= 1;
    }
    return r;
}

bool flag[maxn];
int pri[maxn],cnt = 0;
void GetPrime() {
    for (int i = 2; i < maxn; ++i) {
        if (!flag[i]) pri[cnt++] = i;
        for (int j = 0; j < cnt && pri[j] * i < maxn; ++j) {
            flag[pri[j] * i] = 1;
            if (i % pri[j] == 0) break;
        }
    }
}

LL Rand(LL n) {
    return (((LL)rand()<<31)|rand())%(n-1)+1;
```

```
}

bool test(LL n,LL a,LL d) {
    if(n == 2) return true;
    if(n == a) return true;
    if(!(n & 1)) return false;
    while(!(d & 1)) d >>= 1;
    LL t = powt(a,d,n);
    while(d != n - 1 && t != n - 1 && t != 1) {
        t = mul(t,t,n);
        d <<= 1;
    }
    return t == n - 1 || (d & 1) == 1;
}

bool isprime(LL n) {
    for(int i = 0; i < 6; i++) {
        if(!test(n,pri[Rand(cnt) - 1],n - 1)) return false;
    }
    return true;
}

LL Pollard_rho(LL n,LL c) {
    LL i = 1,k = 2;
    LL x = Rand(n);
    LL y = x;
    while(true) {
        x = add(mul(x,x,n),c,n);
        LL d = __gcd(abs(y - x),n);
        if(d != 1 && d != n) return d;
        if(x == y) return n;
        if(++i == k) y = x,k <<= 1;
    }
}

void findfac(LL n,map<LL,int> &fac) {
    if(n <= 1) return;
    if(isprime(n)) {
        fac[n]++;
        return;
    }
    LL p = n;
    while(p >= n) p = Pollard_rho(p,Rand(n));
```

```
    findfac(p,fac);
    findfac(n / p,fac);
}

map<LL,int> fac;

int main() {
    srand(772002);
    GetPrime();
    findfac(n,fac);
    return 0;
}
```

西普森积分

```
double A,B;
const double PI = acos(-1);

double f(double x){
    return PI * pow(A * exp(-x * x) + B * sqrt(x),2); // 带积分函数
}

double simpson(double a,double b){
    double c=(a + b) / 2.0;
    return (f(a)+4.0*f(c)+f(b))*(b-a)/6.0;
}

double asr(double a,double b,double EPS,double A){
    double c=(a + b) / 2.0;
    double L=simpson(a,c),R=simpson(c,b);
    if(fabs(L+R-A)<=15.0*EPS) return L+R+(L+R-A)/15.0;
    return asr(a,c,EPS/2.0,L)+asr(c,b,EPS/2.0,R);
}

double asr_main(double a,double b,double EPS) // 积分区间(a,b),精度eps{
    return asr(a,b,EPS,simpson(a,b));
}
```

线性筛

```
const int maxn = 1e7+5;
bool flag[maxn];
int phi[maxn],pri[maxn];
int cnt = 0;


//f[i]表示i的最大质因子
for (int i = 2; i < maxn; ++i){
    if(f[i]) continue;
    for (int j = i;j < maxn; j += i) f[j] = i;
}

//筛质数
void GetPrime(){
    for (int i = 2; i < maxn; ++i){
        if (!flag[i]) pri[cnt++] = i;
        for (int j = 0; j < cnt && 1LL * pri[j] * i < maxn; ++j){
            flag[pri[j] * i] = 1;
            if (i % pri[j] == 0) break;
        }
    }
}

//筛欧拉函数
void Get_phi(){
    phi[1] = 1;
    for(int i = 2; i < maxn; i++){
        if(!flag[i]){
            pri[cnt++] = i;
            phi[i] = i - 1;
        }
        for(int j = 0; j < cnt; j++){
            if(1LL * i * pri[j] > maxn) break;
            flag[i * pri[j]] = true;
            if(i % pri[j] == 0){
                phi[i * pri[j]] = pri[j] * phi[i];
                break;
            }
            else phi[i * pri[j]] = (pri[j] - 1) * phi[i];
        }
    }
}
```

```
//筛莫比乌斯函数
void Get_mu(){
    mu[1] = 1;
    for(int i = 2; i < maxn; i++){
        if(!flag[i]){
            pri[cnt++] = i;
            mu[i] = -1;
        }
        for(int j = 0; j < cnt; j++){
            if(1LL * i * pri[j] > maxn) break;
            flag[i * pri[j]] = true;
            if(i % pri[j] == 0){
                mu[i * pri[j]] = 0;
                break;
            }
            else mu[i * pri[j]] = -mu[i];
        }
    }
}

//筛约数个数
int pri[maxn],e[maxn],div[maxn];
int cnt = 0;
void init(){
    div[1] = 1;
    for(int i = 2; i < maxn; i++){
        if(!flag[i]){
            pri[cnt++] = i;
            e[i] = 1;
            div[i] = 2;
        }
        for(int j = 0; j < cnt; j++){
            if(1LL * i * pri[j] > maxn) break;
            flag[i * pri[j]] = true;
            if(i % pri[j] == 0){
                e[i * pri[j]] = e[i] + 1;
                div[i * pri[j]] = div[i] / (e[i] + 1) * (e[i] + 2);
                break;
            }
            e[i * pri[j]] = 1;
            div[i * pri[j]] = div[i] * 2;
        }
```

```
    }
}
```

扩展埃拉托斯特尼筛法

一种不同于杜教筛,理论优于洲阁筛的求积性函数前缀和的筛法
设求$S(n) = \sum_{i=1}^{n} f(i)$,$f(i)$ 为积性函数
令$m = \lceil \sqrt{n} \rceil$
设F 是i的最大质因子,p为质数

考虑从2开始枚举i
对于大于F 的质数p,计算$f(ip) = f(i)f(p)$
如果i含有至少两个F ,计算$f(i)$,并有$\sum_{F<p,ip\le n} f(p) = g(\lfloor \frac{n}{i} \rfloor) - g(p)$,其中$g(i) = \sum_{p\le i} f(i)$
最后额外加上所有$f(p)$和$f(1)$
可以证明以上过程刚好计算了所有数

具体实现
$dfs(x, p)$表示x的最大质因子为p
枚举$q > p$然后$dfs(y = x * q^k, q)$
对于每个x,求出$f(x)(g(\lfloor \frac{n}{i} \rfloor) - g(p))$
如果$k > 1$额外求出$f(y)$加入答案
该过程复杂度约$n^{3/4}/\log n$

求$g(n)$(该过程实质是求关于质数的函数的前缀和)
定义大小为m的数组$a[i] = g(i), b[i] = g(\lfloor \frac{n}{i} \rfloor)$
如果$x > m$,$g(x) = b[n/x]$
否则$g(x) = a[x]$
利用欧拉筛思想,初始$g(x) = \sum_{i=2}^{x} f(i)$
记忆化搜索$g(x) = g(x) - f(p)(g(\lfloor \frac{x}{p} \rfloor) - g(p-1))$ 更新值
该过程复杂度约$n^{3/4}/\log n$

代码实现:

```
int fg[maxn], pri[maxn];
int cnt = 0;

void GetPrime() {
    for (int i = 2; i < maxn; ++i) {
        if (!fg[i]) pri[cnt++] = i;
        for (int j = 0; j < cnt && pri[j] * i < maxn; ++j) {
            fg[pri[j] * i] = 1;
            if (i % pri[j] == 0) break;
        }
```

```
        }
    }
}

int fa[maxn], fb[maxn];

int Sum(LL n, int k) {
    //求前缀和,从2开始,不计算下标为1的那一项
    n %= mod;
    if(k == 0) return add(n, mod - 1);
    else return add(mul(mul(n, n + 1), (mod + 1) >> 1), mod - 1);
}
//第二部分求解
void solve2(LL n, int k) {
    LL i, r;
    for(r = 1; r * r <= n; r++) fa[r] = Sum(r, k);//从2开始的前缀和
    for(i = 1; i < r; i++) fb[i] = Sum(n / i, k);
    for(LL p = 2; p < r; p++) {
        if(!fg[p]) {
            int pw = k ? p : 1;//根据情况改变,本代码求sum(p^1)和sum(p^0)
            for(i = 1; i <= min(r - 1, n / p / p); i++) {
                if(p * i < r) fb[i] = add(fb[i], mod - mul(pw, add(fb[i * p], mod - fa[p - 1
                else fb[i] = add(fb[i], mod - mul(pw, add(fa[n / p / i], mod - fa[p - 1])));
            }
            for(i = r - 1; i >= p * p; i--) fa[i] = add(fa[i], mod - mul(pw, add(fa[i / p],
        }
    }
}

int sa[maxn], sb[maxn], res, m;
// sa sb 为线性组合后的解 res 为最终答案 m为sqrt(n)
//dfs 执行第一部分 调用dfs(1, 1, 0, n)
void dfs(LL x, int d, int k, LL n) {
    for(int i = k; i < cnt; i++) {
        LL y = x * pri[i];
        int u = mul(d, pri[i] - 1);
        if(y * pri[i] > n) break;
        res = add(res, mul(u, add(y < m ? sb[y] : sa[n / y], mod - sa[pri[i]])));
        dfs(y, u, i + 1, n);
        while(true) {
            y *= pri[i];
            u = mul(u, pri[i]);
            res = add(res, u);
            if(y * pri[i] > n) break;
```

```
            res = add(res, mul(u, add(y < m ? sb[y] : sa[n / y], mod - sa[pri[i]])));
            dfs(y, u, i + 1, n);
        }
    }
}

//另一种dfs写法 调用dfs(n,1,0,n)
void dfs(LL x, int d, int k, LL n) {
    if(k) res = add(res, mul(d, add((x < m ? sa[x] : sb[n / x]), mod - sa[pri[k - 1]])));

    for(int i = k; i < cnt; i++) {
        LL p = pri[i];
        if(p * p > x) break;
        int u = d;
        for(LL q = p, y = x; q * p <= x; q *= p) {
            u = mul(u, p - (p == q));//当前枚举值为n/x*q
            dfs(y /= p, u, i + 1, n);
            res = add(res, mul(u, p));//额外加n/x*q * p
        }
    }
}
//本代码求欧拉函数前n项和
//因为phi(p)=p-1 将其拆分为有求和公式的积性函数p和-1分别求解后组合
//因为我们只需要求和质数有关的部分的值,所以拆分出的函数只需要满足质数部分相同即可
//第一部分求的是质数项的和
void solve1(LL n) {
    LL r, i;
    solve2(n, 1);
    for(r = 1; r * r <= n; r++) sa[r] = fa[r];
    for(i = 1; i < r; i++) sb[i] = fb[i];
    solve2(n, 0);
    for(i = 1; i < r; i++) sa[i] = add(sa[i], mod - fa[i]);
    for(i = 1; i < r; i++) sb[i] = add(sb[i], mod - fb[i]);
    res = add(sb[1], 1); //加上所有质数的值和1的值
    m = r;
    dfs(1, 1, 0, n);
}
```

求单个值的欧拉函数

```
\\利用定义
int eular(int k){
    int i,s;
    s = k;
    for(i = 2;i * i <= k; i++){
        if(k % i == 0) s = s / i * (i - 1);
        while(k % i == 0) k /= i;
    }
    if(k > 1) s = s / k * (k - 1);
    return s;
}
```

各种容斥

```
*****************************已知f[n]=(求和d|n)g(d),求g nlogn
for (int i = 1; i <= n; ++i)
  for (int j = i + i; j <= n; j += i)
    f[j] -= f[i];

*****************************f[n]=(求和d|n)g(d),已知g,求f
for (int i = n; i >= 1; --i)
  for (int j = i + i; j <= n; j += i)
    f[j] += f[i];

*****************************已知f[n]=(求和n|d)g(d),求g nlogn
for (int i = n; i >= 1; --i)
  for (int j = i + i; j <= n; j += i)
    f[i] -= f[j];

*****************************f[n]=(求和n|d)g(d),已知g,求f
for (int i = 1; i <= n; ++i)
  for (int j = i + i; j <= n; j += i)
    f[i] += f[j];
**********************************
f[s]存原来的元素，之后f[s]存子集所有元素和
for (int i = 0; i < n; i++)
    for (int s = 0; s < (1 << n); s++)
        if (s >> i & 1)
            f[s] += f[s ^ 1 << i];



**************************************
```

f[s]存子集所有元素和，之后f[s]存原来的元素

```
for (int i = 0; i < n; i++)
    for (int s = 0; s < (1 << n); s++)
        if (s >> i & 1)
            f[s] -= f[s ^ 1 << i];
```

****

数论卷积nlogn 算1-n的h[x]=(求和d|x)(f[d]*g[x/d]) 已知f,g的1-n

```
int f[MAXN],g[MAXN],h[MAXN]={0};
void calc(int n)
{
    for (int i=1;i*i<=n;i++)
        for (int j=i;i*j<=n;j++)
            if(j==i)h[i*j]+=f[i]*g[i];
            else h[i*j]+=f[i]*g[j]+f[j]*g[i];
}
```

已知f[i],求g[i] = \sum [gcd(i,j)=1]f[i]

```
for(i = 1;i <= n; i++){
    for(j= i + i;j <= n; j += i) f[i] += f[j];
}
for(i = 1;i <= n; i++) f[i] *= mu[i];
for(i = n;i >= 1; i--){
    for(j = i + i;j <= n; j += i) f[j] += f[i];
}
```

新的f[i]即为g[i]

单纯形

```
// 过不了uoj179的变态hack数据……不过其他题水过似乎没什么问题
const int maxn = 1050;
const double eps = 1e-8;
struct Simplex {
    int n, m;
    int idx[maxn], idy[maxn];
    double a[maxn][maxn], ans[maxn];
    //复杂度大概 nm^2 ? 可能再乘一个大常数
    //n个变量 m条约束
    //a[i][j]:i表第几条约束 j表第几个元素
    //a[0][i] -> ci 目标函数中第i个元素系数
    //a[i][0] -> bi 第i条约束中的常数,sigma(Aij*xi) <= bi
```

```
//a[i][j] -> Aij 第i条约束中第j个元素的系数
//求最大化 sigma(ci*xi),i∈N
//约束 xj=bj-sigma(aji*xi) ,j∈B, xj>=0
void init(int _n, int _m) {
    n = _n;
    m =  _m;
    for(int i = 0; i <= n; i++) idx[i] = i;
    for(int i = 1; i <= m; i++) idy[i] = i + n;
}
void pivot(int u, int v) {
    swap(idx[v], idy[u]);
    double t = a[u][v];
    a[u][v] = 1.0;
    for(int i = 0; i <= n; i++) a[u][i] /= t;
    for(int i = 0; i <= m; i++) {
        if(i == u || fabs(a[i][v]) < eps) continue;
        t = a[i][v];
        a[i][v] = 0;
        for(int j = 0; j <= n; j++) a[i][j] -= a[u][j] * t;
    }
}
//是否有解 无解返回false
bool init_check() {
    while(true) {
        int u = 0, v = 0;
        double mn = -eps;
        for(int i = 1; i <= m; i++) {
            if(a[i][0] < mn && (!u || (rand() & 1))) {
                mn = a[i][0];
                u = i;
            }
        }
        if(!u) return true;
        mn = -eps;
        for(int i = 1; i <= n; i++) {
            if(a[u][i] < mn && (!v || (rand() & 1))) {
                mn = a[u][i];
                v = i;
            }
        }
        if(!v) return false;
        pivot(u, v);
    }
```

```
    }
    //返回false表示解无穷大
    bool do_simplex() {
        while(true) {
            int u = 0, v = 0;
            double mn = 1e15, mx = eps;
            for(int i = 1; i <= n; i++) {
                if(a[0][i] > mx && (!v || (rand() & 1))) {
                    mx = a[0][i];
                    v = i;
                }
            }
            if(!v) return true;
            for(int i = 1; i <= m; i++) {
                if(a[i][v] > eps && a[i][0] / a[i][v] < mn) {
                    mn = a[i][0] / a[i][v];
                    u = i;
                }
            }
            if(!u) return false;
            pivot(u, v);
        }
    }
    // ans[0]表示最终答案
    // ans[i]表示xi的值
    void get_ans() {
        memset(ans, 0, sizeof(ans));
        ans[0] = -a[0][0];
        for(int i = 1; i <= m; i++) {
            if(idy[i] <= n) ans[idy[i]] = a[i][0];
        }
    }
} sf;
```

矩阵

矩阵的逆

```
void inverse(){
    double temp;
    for(int i = 0;i < n; i++) g[i][i] = 1.0;
```

```
    for(int i = 0;i < n; i++){
        for(int j = i;j < n; j++){
            if(fabs(f[j][i]) > eps){
                for(int k = 0;k < n; k++){
                    swap(f[i][k],f[j][k]);
                    swap(g[i][k],g[j][k]);
                }
                break;
            }
        }
        temp = f[i][i];
        for(int k = 0;k < n; k++)
        {
            f[i][k] /= temp;
            g[i][k] /= temp;
        }
        for(int j = 0;j < n; j++)
        {
            if(j != i && fabs(f[j][i]) > eps)
            {
                temp = f[j][i];
                for(int k = 0;k < n; k++){
                    g[j][k] -= g[i][k] * temp;
                    f[j][k] -= f[i][k] * temp;
                }
            }
        }
    }
}
```

矩阵快速幂

```
const int msize = 2;
struct matrix {
    int a[msize][msize];
    void clear() {
        memset(a,0,sizeof(a));
    }
    void setI() {
        for(int i = 0; i < msize; i++) {
            for(int j = 0; j < msize; j++) a[i][j] = (i == j);
        }
    }
```

```
    matrix operator *(const matrix &b) const {
        matrix tmp;
        tmp.clear();
        for(int i = 0; i < msize; i++) {
            for(int j = 0; j < msize; j++) {
                for(int k = 0; k < msize; k++) {
                    tmp.a[i][j] = add(tmp.a[i][j], mul(a[i][k], b.a[k][j]));
                    //把乘积加起来,维持在mod*mod范围内,最后再取一次模 会快很多
                }
            }
        }
        return tmp;
    }
    bool operator ==(const matrix &b) const {
        for(int i = 0; i < msize; i++) {
            for(int j = 0; j < msize; j++) {
                if(a[i][j] != b.a[i][j]) return false;
            }
        }
        return true;
    }
};

matrix powt(matrix a,LL b) {
    matrix r;
    r.setI();
    while(b) {
        if(b & 1) r = r * a;
        a = a * a;
        b >>= 1;
    }
    return r;
}
```

Guass消元

```
double a[maxn][maxn],ans[maxn];
bool l[maxn];
// 输入矩阵大小，返回解空间维数，维数为0表示解唯一
//l[i]为true表示该数解唯一
int Gauss(int n) {
    int i,j,k,res = 0,r = 0;
    for(i = 0;i < n; i++) l[i] = false;
```

```
    for(i = 0;i < n; i++) {
        for(j = r;j < n; j++) {
            if(fabs(a[j][i]) > eps) {
                for(k = i;k <= n; k++) swap(a[j][k],a[r][k]);
                break;
            }
        }
        if(fabs(a[r][i]) < eps) {
            res++;
            continue;
        }
        for(j = 0;j < n; j++) {
            if(j != r && fabs(a[j][i] / a[r][i]) > eps) {
                double tmp = a[j][i] / a[r][i];
                for(k = i;k <= n; k++) a[j][k] -= tmp * a[r][k];
            }
        }
        l[i] = true;
        r++;
    }
    for(i = 0;i < n; i++) {
        if(l[i]) {
            for(j = 0;j < n; j++) {
                if(fabs(a[j][i]) > 0) ans[i] = a[j][n] / a[j][i];
            }
        }
    }
    return res;
}
```

求模任意数的矩阵行列式

复杂度n^3logC
主要思想：矩阵第i行乘以k加在第j 行，行列式的值不变，由辗转相除法，可使任意两数中的一个数变为0，那么就可以实现模意义下将矩阵转化成下三角矩阵

```
LL a[205][205];
int N,sign;
LL MOD;
LL get_det() {
    LL ans = 1;
    for(int i = 0; i < N; i++) { //当前行
        for(int j = i + 1; j < N; j++) { //当前之后的每一行，因为每一行的当前第一个数要
```

```
            int x = i,y = j;
            while(a[y][i]) { //利用gcd的方法，不停地进行辗转相除
                LL t = a[x][i] / a[y][i];
                for(int k = i; k < N; k++) a[x][k] = (a[x][k] - a[y][k] * t) % MOD;
                swap(x,y);
            }
            if(x != i) { //奇数次交换，则D=-D'整行交换
                for(int k = 0; k < N; k++)swap(a[i][k],a[x][k]);
                sign ^= 1;
            }
        }
        if(a[i][i] == 0)  return 0;//斜对角中有一个0，则结果为0
        else ans = ans * a[i][i] % MOD;
    }
    if(sign != 0) ans *= -1;
    if(ans < 0)  ans += MOD;
    return ans;
}
```

循环矩阵

定义:第i行的向量为第$i-1$行的向量的各元素依次右移一位得到的结果
例如:

$$\begin{matrix} a_0 & a_1 & a_2 \\ a_2 & a_0 & a_1 \\ a_1 & a_2 & a_0 \end{matrix}$$

性质:两个循环矩阵的乘积仍是循环矩阵

快速求解循环矩阵乘积

设有$n*n$的循环矩阵A,B，求$A*B=C$
定义$f(x)=\sum_{i=0}^{n-1} a_i x^i, g(x)=\sum_{i=0}^{n-1} b_i x^i$
其中a_i, b_i分别为矩阵A,B第一行的元素
设$h(x)=f(x)*g(x)=\sum_{i=0}^{n-1} c_i x^i, c_i=\sum_{(j+k)\%n=i} a_j b_k$ (即**循环卷积**)
则$h(x)$的系数即为矩阵C的第一行的元素
故求A^m即为求$f(x)^m$,多项式快速幂即可

对于长度为2的幂的多项式的m次循环卷积,可以直接以当前长度作为fft 长度,不需要满足2倍关系

```
\\ 数组b为数组a进行n次卷积后答案
        b[0] = 1;
```

```
        NTT(a,k,1);
        NTT(b,k,1);
        for(; n; n >>= 1) {
            if(n & 1) {
                for(i = 0; i < k; i++) b[i] = mul(a[i],b[i]);
            }
            for(i = 0; i < k; i++) a[i] = mul(a[i],a[i]);
        }
        NTT(b,k,-1);
```

线性齐次递推

线性齐次递推递推式不含常数项

复杂度m^2logn

```
//求第n项,a[]中记录0~m-1项,c[]记录转移式
//f(n)=f(n-1)c(m-1)+f(n-2)c(m-2)+...+f(n-m)c(0)
int Solve(int n, int m, int a[], int c[]) {
    int v[maxn] = {1}, u[maxn * 2], an = 0;
    int W = !!n, x, b;
    int i, j, t;
    for(x = n; x > 1; x >>= 1) W <<= 1;
    for(x = 0; W; W >>= 1, x <<= 1) {
        memset(u, 0, sizeof(u));
        if(b = !!(n & W)) x |= 1;
        if(x < m) u[x] = 1;
        else {
            for(i = 0; i < m; i++) {
                for(j = 0, t = i + b; j < m; j++, t++) u[t] = add(u[t], mul(v[i], v[j]));
            }
            for(i = m * 2 - 1; i >= m; i--) {
                for(j = 0, t = i - m; j < m; j++, t++) u[t] = add(u[t], mul(u[i], c[j]));
            }
        }
        copy(u, u + m, v);
    }
    for(i = 0; i < m; i++)an = add(an, mul(v[i], a[i]));
    return an;
}

//输入序列前n项 a 返回 系数序列c和长度
int BM(int n, int a[], int c[]) {
```

```
    static int b[maxn], ct[maxn];
    b[0] = c[0] = 1;
    int L = 0, m = 1, cb = 1, szb = 1, szc = 1, d, i, j;
    for(i = 0; i < n; i++) {
        for(j = d = 0; j <= L; j++) d = add(d, mul(c[j], a[i - j]));
        if(d == 0) ++m;
        else if(2 * L <= i) {
            memcpy(ct, c, sizeof(int) * szc);
            int tmp = szc, res = mod - mul(d, powt(cb, mod - 2));
            while(szc < szb + m) c[szc++] = 0;
            for(j = 0; j < szb; j++) c[j + m] = add(c[j + m], mul(res, b[j]));
            memcpy(b, ct, sizeof(int) * tmp);
            szb = tmp;
            L = i + 1 - L;
            cb = d;
            m = 1;
        } else {
            d = mod - mul(d, powt(cb, mod - 2));
            while(szc < szb + m) c[szc++] = 0;
            for(j = 0; j < szb; j++) c[j + m] = add(c[j + m], mul(d, b[j]));
            ++m;
        }
    }
    szc--;
    for(i = 0;i < szc; i++) c[i] = add(0, mod - c[i + 1]);
    reverse(c, c + szc);
    return szc;
}
```

特征方程递推

若已知序列满足$x_n = ax_{n-1} + bx_{n-2}$
有特征方程$x^2 = ax + b$
设该方程的解为q, p
若$q \neq p$则$x_n = Ap^n + Bq^n, A = \frac{x_2 - qx_1}{p(p-q)}, B = \frac{px_1 - x_2}{q(p-q)}$
若$q = p$则$x_n = (A + Bn)p^n, A = \frac{px_1 - x_2}{p^2}, B = \frac{x_2 - px_1}{p^2}$
更高阶递推式可以采用类似的方法,把x_n换成x^k,解方程即可
如果$f_n = (a + \sqrt{b})^n + (a - \sqrt{b})^n$
那么有$f_n = 2af_{n-1} + (b - a^2)f_{n-2}$

博弈

SG游戏

走最后一步的赢
定义赢的状态(空集)为$sg = 0$,可递推所有情况sg,异或和为0先手胜

Anti-SG游戏

走最后一步的输
定义输的状态(空集)为$sg = 0$
先手胜当且仅当:sg异或和不为0且存在单一游戏sg大于1
或 sg异或和为0且不存在单个游戏sg大于1

威佐夫博弈

有两堆各若干个物品,两个人轮流从某一堆或同时从两堆中取同样多的物品,规定每次至少取一个,多者不限,最后取光者得胜
先手必败局面为$(a_k, b_k), a_k = \frac{k(1+\sqrt{5})}{2}, b_k = a_k + k, (k = 0, 1, 2, \ldots$

翻硬币游戏

一排n个硬币,有的正面朝上,有的反面朝上
游戏者根据某些约束翻硬币（如：每次只能翻一或两枚，或者每次只能翻连续的几枚），但他所翻动的硬币中，最右边的必须是从正面翻到反面,不能翻为输
总的sg值为每个正面朝上的硬币单独存在时(即只有这颗硬币正面朝上,其它都是反面朝上)的sg异或和

树上删边游戏

每次操作选择一条边并删掉一棵子树,无边(只有一个点)时输
叶子结点$sg = 0$,其它结点为其儿子结点异或和+1

无向图删边游戏

存在一个点是根,每次操作后不与根相连的联通块删掉,无边(只有一个点)时输
偶环缩成一个点,原来与环相连的边全部连到点上,
奇环缩成两个点加一条边,与dfs序下的父亲相连的边连到一个点上,原来与环相连的其它边全部连到另一个点上
然后变成一个树上删边游戏

k倍动态减

两人取一堆石子，石子有n个。 先手第一次不能全部取完但是至少取一个。之后每人取的个数不能超过另一个人上一次取的数的K倍。拿到最后一颗石子的赢
思路:
（1）首先k=1的时候，必败态是2^i,因为我们把数二进制分解后，拿掉二进制的最后一个1，那么对方必然不能拿走倒数第二位的1，因为他不能拿的比你多。你只要按照这个策略对方一直都不可能拿完。所以你就会赢。
（2）k=2的时候，必败态是斐波那契数列。因为任何一个整数n都可以写成若干项不相邻的斐波那契数的和，所以我们拿掉1，对方永远取不完再高位的1。因为斐波那契数列两项$f[i]$和$f[i + 2]$满足

$f[i+2] > 2 * f[i]$。比如设斐波那契数列为$1, 2, 3, 5, 8, 13 \ldots\ldots 12 = 8 + 3 + 1$，化成二进制就好比是10101,那么你拿走最右边的1(其实就是1)，那么对方不可能拿走第三位的1(这个1其实是3)，这样就和 k=1一个道理，对方不可能拿完，所以你就能拿完；

（3）k>=3的时候，我们必须构造数列，将n写成数列中一些项的和，使得这些被取到的项的相邻两个倍数差距>k 那么每次去掉最后一个1 还是符合上面的条件。设这个数列已经被构造了i 项,第i项为a[i], 前i项可以完美对1到b[i] 编码使得每个编码的任意两项倍数>K 那么有$a[i+1] = b[i] + 1$,这是显然的。因为$b[i] + 1$没法构造出来。只能新建一项表示。然后计算$b[i+1]$, 既然要使用$a[i+1]$ 那么下一项最多只能是某个a[t]使得$a[t] * K < a[i+1]$，于是$b[i+1] = b[t] + a[i+1]$,

最后判断n是否在数列a里面。如果在，那么先手必败。

```
scanf("%d%d",&n,&k);
        int i=0,j=0;
        a[0]=b[0]=1;
        while(a[i]<n){
            i++;
            a[i]=b[i-1]+1;
            while(a[j+1]*k<a[i])
                j++;
            if(a[j]*k<a[i])
                b[i]=b[j]+a[i];
            else
                b[i]=a[i];
        }
        if(a[i]==n)
            puts("lose");
```

不平等博弈

1.爆搜dp

2.surreal number(超现实数)

每种局面用一个数字x表示

如果$x > 0$不论先后手第一个人获胜

如果$x < 0$不论先后手第二个人获胜

如果$x = 0$谁后手谁获胜

如果存在多个子游戏,总的值就是子游戏的值的加法和

定义一个局面的surreal number 为$x = L|R = L_{max}|R_{min}$,L 为第一个人先走能到达的局面的值的集合,R为第二个人先走能到达的局面的值的集合

且保证$L_{max} < R_{min}$

如果L 为空,$x = R_{min} - 1$

如果R为空,$x = L_{max} + 1$

否则$x = (L_{max} + R_{min})/2$

另一种线性基写法

能一直维护出一个上三角矩阵,不过顺序是乱的

```
vector<int> base;

void add(int x) {
    for(auto u:base) x = min(x, u ^ x);
    if(!x) return;
    for(auto &u:base) u = min(u, u ^ x);
    base.push_back(x);
}

int query(int x = 0) { //求x与基异或的最值
    for(auto u:base) x = min(x, u ^ x); // x = max(x, u ^ x);
    return x;
}
```