ACM/ICPC 比赛资料

数学

目录

概率知识

概率

条件概率： $P(A|B)=\dfrac{P(AB)}{P(B)}$

全概率公式：$P(A)=\sum_{i=1}^{n}P(B_i)P(A|B_i)$，其中$\mathrm{B}_i(1\le \mathrm{i}\le \mathrm{n})$为 Ω 的一个划分

贝叶斯公式：$P(B_i|A)=\dfrac{P(AB_i)}{P(A)}=\dfrac{P(B_i)P(A|B_i)}{\sum_{k=1}^{n}P(B_k)P(A|B_k)}$

在（0，1）随机取出 n 个数，则第 k 小的数的期望为 k / (n+1)

分赌注问题：在可列重 Bernoulli 试验中，n 次成功发生在 m 次失败之前的概率

$$\sum_{k=n}^{n+m-1}f(k;n,p)=\sum_{k=n}^{n+m-1}C_{k-1}^{n-1}p^nq^{k-n}=\left(\frac{p}{q}\right)^n\sum_{k=n}^{n+m-1}C_{k-1}^{n-1}q^k$$

广义 *Bernoulli* 实验：假定实验有 r 种可能的结果$\mathrm{A}_1,\mathrm{A}_2,\cdots,\mathrm{A}_\mathrm{r}$，并且$\mathrm{p_i}=\mathrm{p}(\mathrm{A_i})>0,\sum\mathrm{p_i}=1$, $\mathrm{i}=1,2,..,\mathrm{r}$，重复 n 次，$\mathrm{A_i}$出现$\mathrm{k_i}$次$(\sum\mathrm{k_i}=\mathrm{n},\mathrm{i}=1,2,..,\mathrm{r})$的概率为：

$$\binom{k_1\quad\dots\quad k_r}{n}p_1^{k^1}\cdots p_r^{k^r}$$

$\mathrm{A_i}$在$\mathrm{A_j}$之前出现的概率

$$\frac{p_i}{p_i+p_j}$$

随机徘徊的吸收概率：

质点在数轴的整点上运动，无论它处在哪个点 i 上，下一时刻都以概率 p 向右移动到 i+1，以概率 q 移动到 i-1, $p, q>0$, $p+q=1$. 我们把这种运动称为(直线上的)随机徘徊，现在考虑数轴上的两个特殊的点 0 和 a (a > 0)，假定质点运动到 0 或 a 之后就永远不再移动，称这样的 0 与 a 为随机徘徊的吸收壁，现求自 i (0 < i < a)出发的质点将被 0 或 a 吸收的概率。

令 P(i)为质点从 i 出发，被 a 吸收的概率，则

$$\boldsymbol{P(i)=\begin{cases}\dfrac{i}{a} & p=q\\[2ex] \dfrac{1-\left(\frac{q}{p}\right)^i}{1-\left(\frac{q}{p}\right)^a} & p\neq q\end{cases}}$$

证明思路：

由全概率公式 P(i) = qP(i-1) + pP(i+1)，得 P(i+1) – P(i) = (q / p) (P(i) – P(i-1))

若 q / p = 1，则 {P(i)} 为等差数列；

若 $q / p \neq 1$，则 {P(i+1) – P(i)} 为等比数列。

注： **若$\boldsymbol{p}+\boldsymbol{q} \neq \mathbf{1}$**，则 P(i) = qP(i-1) + pP(i+1) + (1-p-q)P(i)，**只需设 $\boldsymbol{p'} = \boldsymbol{p}/(\boldsymbol{p}+\boldsymbol{q}), \boldsymbol{q'} = \boldsymbol{q}/(\boldsymbol{p}+\boldsymbol{q})$**，则 P(i) = q'P(i-1) + p'P(i+1)，可得到相同结论。

注：也可用 O(n)的方法求解三对角线性方程组。

分布

分布名称	概率密度	最大值点[1]	期望 E	方差 D[2]
二项分布 $B(n,p)$	$C_n^k p^k q^{n-k}$ $k = 0, 1, \ldots, n$	$(n+1)p - 1 \leq k \leq (n+1)p$	np	npq
几何分布 $G(p)$	$q^{k-1}p$ $k = 1, 2, \ldots$	$k = 1$	$\frac{1}{p}$	$\frac{q}{p^2}$
Pascal 分布[3] (负二项分布)	$C_{k-1}^{r-1} p^r q^{k-r}$ $k = r, r+1, \ldots$	$\frac{r-1}{p} \leq k \leq \frac{r-1}{p} + 1$	$\frac{r}{p}$	$\frac{rq}{p^2}$
超几何分布[4]	$\frac{C_M^k C_{N-M}^{n-k}}{C_N^n}$ $k = 0, 1, \ldots, n$	$\frac{(M+1)(N+1)}{2+N} - 1 \leq k \leq \frac{(M+1)(N+1)}{2+N}$	$\frac{nM}{N}$	$\frac{nM}{N}\left(1-\frac{M}{N}\right)\frac{N-n}{N-1}$
Poisson 分布 $P(\lambda)$	$\frac{\lambda^k}{k!}e^{-\lambda}$ $k = 0, 1, 2, \ldots$	$\lambda - 1 \leq k \leq \lambda$	λ	λ
正态分布[5] $N(\mu, \sigma^2)$	$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ $-\infty < x < +\infty$	$x = \mu$	μ	σ^2
指数分布	$\lambda e^{-\lambda x}$	$x = 0$	$\frac{1}{\lambda}$	$\frac{1}{\lambda^2}$

注：

1. 求最大值点方法$f(i) \geq f(i+1), f(i) \geq f(i-1)$
2. $D\xi = E(\xi^2) - (E\xi)^2$
3. **Pascal 分布$\{\boldsymbol{f(k;r,p)}\}$意义**：可列重 Bernoulli 试验第 r 次成功的等待时间为 k 的概率，几何分布是 Pascal 分布 r = 1 时的特例，即$g(k;p) = f(k;1,p)$
4. 超几何分布：有 N 个产品，其中 M 个次品，N-M 个合格，不放回取 n 次，恰取到 k 个次品概率。
5. 对于正态分布$\Phi(x) = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$

对于 n 个随机变量 $X_1, X_2, \ldots, X_n$，设他们的概率密度函数为 $f_i(x)$ (1<=i<=n)，则 $X_1+X_2+\ldots+X_n$ 的密度函数为$(f_1 * f_2 * \ldots * f_n)(x)$，其中*为卷积符号

定理：取值为自然数的随机变量ξ有几何分布 iff. ξ无记忆性，即$P(\xi > m+n \mid \xi > m) = P(\xi > n)$

***Poisson* 分布在随机选择下不变**：假设一块放射性物质在单位时间内放射出的α粒子数$\xi \sim$ P(λ)，每个放射出的α粒子被记录下来的概率均为 p，就是说，每个放射出的粒子有 1-p 的概率被计数器遗漏。如果各粒子是否被记录相互独立，记录下的α粒子数~P(λp)。
证明：

$$P(\eta=k)=\sum_{n=k}^{\infty}P(\xi=n)P(\eta=k\mid\xi=n)$$

$$=\sum_{n=k}^{\infty}p(n;\,\lambda)b(k;n,p)=\sum_{n=k}^{\infty}\frac{\lambda^n}{n!}e^{-\lambda}C_n^k p^k q^{n-k}=\sum_{n=k}^{\infty}\frac{(\lambda q)^{n-k}}{(n-k)!}\cdot\frac{1}{k!}e^{-\lambda}(\lambda p)^k$$

$$=\frac{1}{k!}(\lambda p)^k e^{-\lambda p}$$

***Poission* 过程：**
定理：假定于随机时间陆续到达的质点流满足以下条件：
1. 在不相交时段内到达的质点数目相互独立。
2. 在长为 t 的时段 [a, a + t)内到达 k 个质点的概率只与计时长度 t 有关，而与计时起点 a 无关。
3. 在有限的时间内只来有限个质点，且在充分短的时间内，最多只来一个质点。

则必存在常数 λ > 0，使得对一切 t > 0 有$P_k(t)=\frac{(\lambda t)^k}{k!}e^{-\lambda t}$

数学公式

Lucas 定理
对于 n, m >= 0, p – prime
$n = n_k p^k + \ldots + n_1 p + n_0$
$m = m_k p^k + \ldots + m_1 p + m_0$

$$C_n^m=\prod_{i=0}^{k}C_{n_i}^{m_i}$$

应用：
1. $\mathbf{C}_{\boldsymbol{m}}^{\boldsymbol{p}} \equiv (\boldsymbol{m}\ \mathbf{div}\ \mathbf{p})\ \mathbf{mod}\ \mathbf{p}$
2. Pascal 三角形第 n 行$C_n^0, C_n^1 \ldots, C_n^n$中 mod p != 0 的数，有$\prod_{i=0}^{n}(n_i+1)$个。

设 y ≥ 1, $\sum_{1\le n\le y}\frac{\mathbf{1}}{\boldsymbol{n}}=\mathbf{ln}\,\boldsymbol{y}+\boldsymbol{\gamma}+\Delta(y), |\Delta(y)|\le\frac{1}{y}$,欧拉常数$\gamma=0.5772$

设 x ≥ 2, $\sum_{p\le x}\frac{\mathbf{1}}{\boldsymbol{p}}=\mathbf{ln}\,\mathbf{ln}\,\boldsymbol{x}+\boldsymbol{A}+r(x)$,

其中 $A = \gamma + \sum_p \left\{\ln\left(1 - \frac{1}{p}\right) + \frac{1}{p}\right\} = 0.26149721 \ldots, |r(x)| < 2(5\ln 2 + 3)\frac{1}{\ln x}$

不超过 x 的质数的个数 $\pi(x) \approx \frac{x}{\ln(x)+B}$，其中 Legendre 常数 B = -1.08366

Wilson 定理：
设 $r_1, \ldots, r_{p-1}$ 是模 m 的既约剩余系，我们有
$$\prod_{(r,\mathrm{m})=1} (r = r_1 \ldots r_c) \equiv \begin{cases} -1 & m = 1, 2, 4, p^\alpha, 2p^\alpha, p - \text{positive odd prime} \\ 1 & \text{else} \end{cases} \text{mod m}$$
特别地
$$(p-1)! \equiv -1 \; mod \; p$$

[拓扑学]欧拉公式：对于一个凸多面体，|V| - |E| + |F| = 2

Stirling 近似公式：
$$\mathrm{n}! = \left(\frac{\mathrm{n}}{\mathrm{e}}\right)^{\mathrm{n}} \sqrt{2\pi\mathrm{n}}\left(1 + \Theta\left(\frac{1}{\mathrm{n}}\right)\right) > \left(\frac{\mathrm{n}}{\mathrm{e}}\right)^{\mathrm{n}}$$
$$\mathrm{n}! = \left(\frac{\mathrm{n}}{\mathrm{e}}\right)^{\mathrm{n}} \sqrt{2\pi\mathrm{n}}\left(1 + \frac{1}{12n} + \frac{1}{288n^2} + \Theta\left(\frac{1}{\mathrm{n}^3}\right)\right) = \Gamma(n+1)$$
因此
$$\log(\mathrm{n}!) = \Theta(\mathrm{n}\log \mathrm{n})$$

$$\mathrm{H}_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n} = ln(n) + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} + O\left(\frac{1}{n^6}\right)$$

广义快速幂取模
$$\mathrm{A}^{\mathrm{x}} \equiv \mathrm{A}^{\mathrm{x} \bmod \phi(\mathrm{C}) + \phi(\mathrm{C})} \bmod \mathrm{C} \; (\mathbf{x} \geq \boldsymbol{\varphi}(\mathbf{C}))$$

数学常数

$\pi = 3.14159\ 26535\ 89793\ 23846\ 26433\ 83279\ 50288\ 41971\ 69399\ 37510\ 58209\ 74944 \ldots$

$\gamma = \lim_{\mathrm{n}\to\infty}(\mathrm{H}_n - \ln n) = 0.57721566490153286060651209008240243104 2\ldots$

$\phi = \frac{1+\sqrt{5}}{2} = 1.61803$

$\widehat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803$

一元方程求根公式

二次方程求根公式

$ax^2+bx+c=0$	$x_1=\frac{-b-(\mathbf{sgn}(\mathbf{b})\geq\mathbf{0}\,?\,\mathbf{1}:-\mathbf{1})\sqrt{b^2-4ac}}{2a}$ $x_2=\frac{c}{ax_1}$

三次方程求根公式 - 卡尔丹公式

一元三次方程 $X^3+pX+q=0$ 通解	$X_1=\sqrt[3]{Y_1}+\sqrt[3]{Y_2}$ $X_2=\omega\sqrt[3]{Y_1}+\omega^2\sqrt[3]{Y_2}$ $X_3=\omega^2\sqrt[3]{Y_1}+\omega\sqrt[3]{Y_2}$	其中$\omega=\frac{1}{2}\left(-1+i\sqrt{3}\right)$ $Y_{1,2}=-\frac{q}{2}\pm\sqrt{\left(\frac{q}{2}\right)^2+\left(\frac{p}{3}\right)^3}$

一元三次方程$aX^3+bX^2+cX+d=0$，令$X=Y-\frac{b}{3a}$，代入原方程得即可

四次方程求根公式

$$x^4+bx^3+cx^2+dx+e=0$$
$$x^4+bx^3=-cx^2-dx-e$$

两边同时加上$\left(\frac{1}{2}bx\right)^2$，将左边配成完全平方

$$\left(x^2+\frac{1}{2}bx\right)^2=\left(\frac{1}{4}b^2-c\right)x^2-dx-e$$

两边同时加上$\left(x^2+\frac{1}{2}bx\right)y+\frac{1}{4}y^2$，将左边配方

$$\left[\left(x^2+\frac{1}{2}bx\right)+\frac{1}{2}y\right]^2=\left(\frac{1}{4}b^2-c+y\right)x^2+\left(\frac{1}{2}by-d\right)x+\frac{1}{4}y^2-e$$

上式中的 y 是一个参数，当上式中的 x 为原方程的根时，无论 y 取什么值，上式都应该成立。特别地，若所取的 y 值使上式右边关于 x 的二次多项式也能变成一个完全平方式，则对上式两边同时开方可以得到次数较低的方程。为了使上式右边关于 x 的二次多项式也能变成一个完全平方式，只需使他的判别式变成 0，即

$$\left(\frac{1}{2}by-d\right)^2-4\left(\frac{1}{4}b^2-c+y\right)\left(\frac{1}{4}y^2-e\right)=0$$

化简得

$$y^3-cy^2+(bd-4e)y-d^2-(b^2-4c)e=0$$

这是一个关于 y 的一元三次方程，可以通过塔塔利亚公式来求出 y 的值。把求出的 y 的值带入后，可以得到两个关于 x 的一元二次方程。解这两个一元二次方程，就可以得出原方程的

四个根。

代码

```
typedef complex<double> cpl;
void solve2(cpl a, cpl b, cpl c, cpl r[]){
	cpl delta = b * b - 4.0 * a * c; delta = sqrt(delta);
	r[0] = (-b + delta) / (2.0 * a); r[1] = (-b - delta) / (2.0 * a);
}

void solve3(double p, double q, cpl r[]){ //已经通过 cugb1024
	cpl delta = q * q * (1.0 / 4.0) + p * p * p * (1.0 / 27.0); delta = sqrt(delta);
	cpl y1 = -q * 0.5 + delta, y2 = -q * 0.5 - delta;
	cpl w(-0.5, sqrt(3.0) * 0.5), w2 = w * w;
	y1 = pow(y1, 1.0 / 3.0); y2 = pow(y2, 1.0 / 3.0);
	double diffmax = DBL_MAX;
	for (int i = 0; i < 3; i++, y2 *= w){
		cpl rr[3]; double diff = 0;
		rr[0] = y1 + y2; rr[1] = w * y1 + w2 * y2; rr[2] = w2 * y1 + w * y2;
		rep(k, 3) diff += abs(rr[i] * rr[i] * rr[i] + p * rr[i] + q);
		if (diff < diffmax){
			diffmax = diff; rep(k, 3) r[k] = rr[k];
		}
	}
}

void solve3(double b, double c, double d, cpl r[]){
	double p = -b * b * (1.0 / 3.0) + c, q = (2.0 / 27.0) * (b * b * b) - (1.0 / 3.0) * (b * c) + d;
	solve3(p, q, r);
	for (int i = 0; i < 3; i++) r[i] -= b * (1.0 / 3.0);
}

void solve3(double a, double b, double c, double d, complex<double> r[]){
	b /= a; c /= a; d /= a; solve3(b, c, d, r);
}

void solve4(double b, double c, double d, double e, complex<double> r[]){//已通过 hdu 某题
	double B, C, D; cpl y[3];
	B = -c; C = (b * d - 4.0 * e); D = - d * d - (b * b - 4.0 * c) * e; solve3(B, C, D, y);
	cpl p = 0, q, tp, yy;
	rep(i, 3) {
		tp = 0.25 * b * b - c + y[i];
		if (abs(tp) > abs(p)){ p = tp; yy = y[i]; }
	}
```

```
        q = (0.5 * b * yy - d) / (p * 2.0); p = sqrt(p);
        solve2(1.0, 0.5 * b + p, 0.5 * yy + p * q, r);
        solve2(1.0, 0.5 * b - p, 0.5 * yy - p * q, r + 2);
}

void solve4(double a, double b, double c, double d, double e, cpl r[]){
        solve4(b / a, c / a, d / a, e / a, r);
}
```

数学级数

$$f(x)=\sum_{n=0}^{\infty}\frac{f^n(x)}{n!}x^n$$

指数函数和对数函数

$$\mathrm{e}^x=\sum_{n=0}^{\infty}\frac{x^n}{n!}=1+\mathrm{x}+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots,\forall x$$

$$\ln(1+x)=\sum_{n=1}^{\infty}\frac{(-1)^{n+1}}{n!}x^n=\mathrm{x}-\frac{x^2}{2!}+\frac{x^3}{3!}-\frac{x^4}{4!}+\cdots,\forall x\in(-1,1]$$

$$\ln(1-x)=-\sum_{n=1}^{\infty}\frac{x^n}{n!}=-\mathrm{x}-\frac{x^2}{2!}-\frac{x^3}{3!}-\frac{x^4}{4!}+\cdots,\forall x\in[-1,1)$$

二项式级数

$$(1+x)^{\alpha}=\sum_{n=0}^{\infty}C(\alpha,\mathrm{n})x^n,\forall x：|x|<1,\forall\alpha\in\mathbb{C}$$

$\alpha=-1$时为几何级数

$$\frac{1}{1-\mathrm{x}}=\sum_{n=0}^{\infty}x^n,\forall x：|x|<1$$

$\alpha=-\frac{1}{2}$时为平方根级数

$$\sqrt{1+\mathrm{x}}=\sum_{n=0}^{\infty}\frac{(-1)^n(2n)!}{(1-2\mathrm{n})(n!)^2 4^n}x^n=1+\frac{1}{2}x-\frac{1}{8}x^2+\frac{1}{16}x^3-\frac{5}{128}x^4+\cdots,\forall|x|\le 1$$

三角函数

$$\mathrm{sinx}=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n+1)!}x^{2n+1}=\mathrm{x}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\cdots,\forall x$$

$$\mathrm{cosx}=\sum_{n=0}^{\infty}\frac{(-1)^n}{(2n)!}x^{2n}=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\cdots,\forall x$$

$$\tan x=\sum_{n=0}^{\infty}\frac{B_{2n}(-4)^n(1-4^n)}{(2n)!}x^{2n-1}=\mathrm{x}+\frac{x^3}{3}+\frac{2x^5}{15}+\cdots,\forall x:\ |x|<\frac{\pi}{2}$$

$$\sec x=\sum_{n=0}^{\infty}\frac{(-1)^nE_{2n}}{(2n)!}x^{2n},\forall x:\ |x|<\frac{\pi}{2}$$

$$\arcsin x=\frac{\pi}{2}-\arccos x=\sum_{n=0}^{\infty}\frac{(2n)!}{4^n(n!)^2(2n+1)}x^{2n+1},\forall x:|x|\le 1$$

$$\arctan x=\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}x^{2n+1}=\mathrm{x}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\cdots,\forall x:|x|<1$$

双曲函数

$$\sinh x=\sum_{n=0}^{\infty}\frac{1}{(2n+1)!}x^{2n+1}=\mathrm{x}+\frac{x^3}{3!}+\frac{x^5}{5!}+\frac{x^7}{7!}+\cdots,,\forall x$$

$$\cosh x=\sum_{n=0}^{\infty}\frac{1}{(2n)!}x^{2n}=1+\frac{x^2}{2!}+\frac{x^4}{4!}+\frac{x^6}{6!}+\cdots,\forall x$$

$$\tanh x=\sum_{n=0}^{\infty}\frac{B_{2n}4^n(4^n-1)}{(2n)!}x^{2n-1},\forall x:|x|<\frac{\pi}{2}$$

$$\operatorname{arcsinh} x=\sum_{n=0}^{\infty}\frac{(-1)^n(2n)!}{4^n(n!)^2(2n+1)}x^{2n+1},\forall x:|x|\le 1$$

$$\arctan x=\sum_{n=0}^{\infty}\frac{1}{2n+1}x^{2n+1}=\mathrm{x}+\frac{x^3}{3!}+\frac{x^5}{5!}+\frac{x^7}{7!}+\cdots,\forall x:|x|<1$$

朗伯 W 函数

$$\mathrm{W}_0(x)=\sum_{n=1}^{\infty}\frac{(-\mathrm{n})^{n-1}}{n!}x^n=\mathrm{x}-x^2+\frac{3}{2}x^3-\frac{8}{3}x^4+\frac{125}{24}x^6-\cdots,\forall x:|x|<\frac{1}{e}$$

其中$C(\alpha,\mathrm{n})$是二项式系数，B_k是伯努利数，E_{2n}是欧拉数(不是组合数学中的欧拉数)

$$E_{2n}=\mathrm{i}\sum_{k=1}^{2n+1}\sum_{j=0}^{k}\binom{k}{j}\frac{(-1)^j(k-2\mathrm{j})^{2n+1}}{2^k i^k k},\mathrm{i}^2=-1$$

矩阵乘法

注意求递推式的时候可能要用到二项式定理，如求$\sum_{i=1}^{n} i^k k^i\ \ (k\le 50)$

求 E+A+A^2+…+A^k

$$\begin{pmatrix} A_{n\times n} & E_{n\times n} \\ O_{n\times n} & E_{n\times n} \end{pmatrix}^{k+1} = \begin{pmatrix} A^{k+1} & E+A+\cdots+A^k \\ O & E \end{pmatrix}$$

Raney 引理

设整数序列 A = {a_i}, i = 1, 2, …, n, 且部分和 $S_k = a_1 + a_2 + \ldots + a_k$，序列中所有数的和 $S_n = 1$
结论：在 A 的 n 个循环表示中，有且只有一个序列 B，满足 B 的任意部分的和 S_i 均大于 0.

卷积

定义：
两个函数 f 和 g 的卷积

$$(f*g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau$$

对于定义在离散域的函数

$$(f*g)[m] = \sum_n f[n]g[m-n]$$

性质：
交换律(Commutativity): $f*g = g*f$
结合律(Associativity): $f*(g*h) = (f*g)*h$
分配率(Distributivity): $f*(g+h) = (f*g)+(f*h)$
数乘结合律(Associativity with scalar multiplication): $a(f*g) = (af)*g = f*(ag)$
微分定理(Differentiation) $\mathcal{D}(f*g) = \mathcal{D}\mathrm{f}*g = \mathrm{f}*\mathcal{D}g$, $\mathcal{D}$表示微分(或差分——离散域上)

Chebyshev 距离与 Manhattan 距离的转化

Chebyshev 距离：dist(A, B) = max{ |Ax - Bx|, |Ay - By| }
Manhattan 距离：dist(A, B) = sum{ |Ax - Bx|, |Ay - By| }
则 Chebyshev 空间上的点 P(x, y)可以一一映射到 Manhattan 空间上的点 P’((x+y)/2, (x-y)/2)
However, this equivalence between L_1 and L_∞ metrics does not generalize to higher dimensions.

数值分析

求和公式

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \mathrm{a}$$

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4} = a^2$$

$$\sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

$$\sum_{k=1}^{n} k^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12} = \frac{4a^3-a^2}{3}$$

$$\sum_{k=1}^{n} k(k+1) = \frac{n(n+1)(n+2)}{3}$$

$$\sum_{k=1}^{n} k(k+1)(k+2) = \frac{n(n+1)(n+2)(n+3)}{4}$$

$$\sum_{k=1}^{n} k(k+1)(k+2)(k+3) = \frac{n(n+1)(n+2)(n+3)(n+4)}{5}$$

牛顿插值公式

设 f(x)过 k+1 个点$(x_0, y_0), \ldots, (x_k, y_k)$, 则

$$\begin{aligned}\mathrm{f}(x) = \mathrm{N}(x) = \mathrm{d}_{0,0} + \mathrm{d}_{0,1}(x-\mathrm{x}_0) + \mathrm{d}_{0,2}(x-\mathrm{x}_0)(x-\mathrm{x}_1) + \cdots \\ + \mathrm{d}_{0,\mathrm{k}}(x-\mathrm{x}_0)(x-\mathrm{x}_1)\ldots(x-\mathrm{x}_{k-1})\end{aligned}$$

其中

$$d(i,j) = \begin{cases} f(x_i) & i = j \\ \dfrac{d(i+1,j)-d(i,j-1)}{x_j - x_i} & i \neq j \end{cases}$$

前 n 个正整数的 k 次方的和(一般 k 次多项式的和)

定理：设 k 次多项式 f(n)的差分表的第一条对角线为 $d_0, d_1, \ldots, d_k, 0, 0, \ldots, d_k != 0$, 则

$$\mathrm{f}(n) = \mathrm{d}_0 C_n^0 + \cdots + \mathrm{d}_k C_n^k$$

$$\sum_{i=1}^{n} f(i) = d_0 C_{n+1}^{1} + \cdots + d_k C_{n+1}^{k+1}$$

规范说明：对于度数<=d ($\Delta^{d+1} f(n) = 0$ 或$\Delta^{d} f(n)$是常数) 的多项式 f(n),

$$f(n) = \sum_{k=0}^{d} \Delta^k f(0) \binom{n}{k}$$

特别地，对于 f(n) = n^d 的特别情况(其中 S 表示第二类 Stirling 数)

$$\Delta^k 0^d = k!\,S(d,k)$$

注：本定理本质上是牛顿插值公式的特殊形式

代码：//O(k * k) – O(k), 其中 s 为第二类 stirling 数

```
LL calc2(int n, int d) { // sum{ i ^ d : i = 1 .. n }
	LL ans = 0;
	LL t = ++n;
	for (int k = 0; k <= d && k + 1 <= n; ++k) {
		ans = (ans + s[d][k] * inv[k+1] % module * t) % module; t = t * (n - k - 1) % module;
	}
	return ans;
}
```

龙贝格积分

```
real f(real x) {
	return exp(-x * x);
}

//O(2 ^ maxitr) function evaluations
real Romberg(real a, real b, real(*f)(real), real eps, int maxitr = 20) {
	real T[maxitr][maxitr];
	for (int i = 0; i < maxitr; ++i) {
		real h = (b - a) / (1 << i), x = a + h, pow = 4;
		T[i][0] = (f(a) + f(b)) / 2;
		for (int j = (1 << i) - 1; j >= 1; x += h, --j) T[i][0] += f(x);
		T[i][0] *= h;
		for (int j = 1; j <= i; pow *= 4, ++j)
			T[i][j] = T[i][j - 1] + (T[i][j - 1] - T[i - 1][j - 1]) / (pow - 1);
		if (i > 0 && fabs(T[i][i] - T[i - 1][i - 1]) <= eps) return T[i][i];
	}
	return T[maxitr - 1][maxitr - 1];
}
```

Impartial(公平)组合游戏

想不清了就打个 NP/SG 表进行观察

SG 函数

sg(v) = mex{ f(u) | 图中有一条从 v 到 u 的边}
其中mex{A}表示不属于A的最大整数，即$\text{mex}\{A\} = \min(\mathbb{N} - \text{A}) = \min\{k | k \notin A \wedge k \in \mathbb{N}\}$
注：若在符合拓扑原则的前提下，一个单一游戏的后继可以分为多个单一游戏，sg函数仍然适用。

sg函数的性质

1. 对于任意局面，如果它的sg值为0，那么他的任何一个后继局面的sg不为0;
2. 对于任意局面，如果它的sg值不为0，那么他一定有一个后继局面的sg值为0.
3. **所有sg值相等的局势都是等价的**。

常规 Nim 游戏

N-pos 先手必胜　$\leftrightarrow$　$a_1 \oplus a_2 \oplus \ldots \oplus a_{n-1} \oplus a_n \neq 0$
P-pos 后手必胜　$\leftrightarrow$　$a_1 \oplus a_2 \oplus \ldots \oplus a_{n-1} \oplus a_n = 0$
设从最高位开始恰好第 k 位的异或值不为 0，即 $k = \max\{ k \mid a_1^{(k)} \oplus a_2^{(k)} \oplus \ldots \oplus a_n^{(k)} \neq 0\}$，则任取 $a_i^{(k)} = 1$，则游戏者只需要在第 i 堆石子中拿走 $a_i - (a_1 \oplus a_2 \oplus \ldots \oplus a_{n-1} \oplus a_n \oplus a_i)$颗石子，即可恢复为 P 状态 m，也只有这$|\{ i : a_i^{(k)} = 1 \}|$种取法。
注：Nim 游戏对于某必胜态的必胜取法可以在 O(n)时间内得到，是在第 i 堆中取出 $a_i \oplus sum$ 个石子(若 $a_i \oplus sum < a_i$)

Misere 规则下的 Nim 游戏（SJ 定理）

如果我们**规定当所有游戏的 sg 函数均为 0 时，游戏结束**
（可以弱化为当所有游戏的 sg 函数均为 0 时，存在一个单一游戏，使得它的 sg 函数能够通过单一操作变为 1）
则先手必胜$\leftrightarrow$

1. 局面的 sg 和不为 0 且有某个游戏的 sg 函数大于 1
2. 局面的 sg 和为 0 且没有某个游戏的 sg 函数大于 1

Moore’s Nim

Moore’s Theorem states that a position (x_1, x_2, …, x_n), is a P-position in Nim_k if and only if when x_1 to x_n are expanded in base 2 and added in base k+1 without carry, the result is zero. (In other words, the number if 1’s in each column must be divisible by k + 1.) <看 x_i 的二进制表示的mod(k+1)和>

在 misère 规则下(拿走最后一个石子的人输，所有游戏 sg 均为 0 时游戏结束)，某个状态是 P 状态(先手必败态) iff.

1. 存在某个局势的 sg 值>1，且 sg_k 和为 0
2. 所有局势的 sg 值=1，且 sg_k 和为 1

Every-SG 游戏

对于 SG 值为 0(先手必败、P-position)的点，我们需要知道最少几步能将游戏带入中止状态。对于 SG 值不为 0(先手必胜、N-position)的点，我们需要知道最多就游戏会被带入中止状态。以上两个值，我们都用 step 来表示

$$\mathrm{step}(u)=\begin{cases}0 & u\text{为中止状态}\\ \max\{\mathrm{step}(v)\}+1 & u\text{是 Npos}[\mathrm{sg(u)}>0], \mathrm{v}\in \mathrm{succ(u)}, \mathrm{v}\text{ 是 Ppos}[\mathrm{sg(v)}=0]\\ \min\{\mathrm{step}(v)\}+1 & u\text{是 Ppos}[\mathrm{sg(u)}=0]，\mathrm{v}\in \mathrm{succ(u)},\end{cases}$$

Wythoff Game

参考：http://www.gabrielnivasch.org/academic/publications#wythoff_thesis

描述：有两堆各若干个物品，两个人轮流从某一堆或两堆同时取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

我们用(a_k, b_k, k=0..n)表示两堆物品的数量，并称其为局势。我们把先手必败的局势(P-pos)成为奇异局势。实验得出，前几个奇异局势是(0, 0), (1, 2), (3, 5), (4, 7), (6, 10), (8, 13), (9, 15), (11, 18), (12, 20)

结论：

奇异局势的通项公式是

$$\mathrm{a[k]}=\lfloor \mathrm{k}\phi\rfloor，\mathrm{b[k]}=\mathrm{a[k]}+\mathrm{k}，\text{其中 }\phi=\frac{\sqrt{5}+1}{2}$$

其中(0, 0)我们也认为是奇异局势。

判断：

对于某个(a, b)，设 b-a 等于 k，计算若 a = $\lfloor k\phi\rfloor$，则局势为奇异局势(必败态)。

注：此时$\mathrm{k}=\left\lceil\frac{\mathrm{a}}{\varphi}\right\rceil=\left\lceil\frac{\mathrm{b}}{\varphi+1}\right\rceil=\mathrm{b}-\mathrm{a}$

证明：

1. Beatty 定理(证明略)

   如果存在无理数 a，b 满足

$$\frac{1}{\mathrm{a}}+\frac{1}{\mathrm{b}}=1$$

   令

$$\mathrm{A}=\{\lfloor \mathrm{ia}\rfloor\ |\mathrm{i}\in \mathrm{N}^*\},\mathrm{B}=\{\lfloor \mathrm{jb}\rfloor\ |\mathrm{j}\in \mathrm{N}^*\}$$

则 A 和B构成N^*的划分，即$A \cap B = \emptyset$，$A \cup B = N^*$

2. 每个自然数都包含且仅包含在奇异局势数列

$$a[k] = \lfloor k\varphi \rfloor，b[k] = a[k] + k，其中\ \varphi = \frac{\sqrt{5}+1}{2}$$

中一次。(证明略)

3. 任何操作只能将奇异局势变为非奇异局势

   若只改变奇异局势的一个分量，则另一个分量不可能在其他奇异局势中，故必然是非奇异局势；如果使两个分量同时较少，由于其差不变，且不可能是其他奇异局势的差，故也是非奇异局势。

4. 采用适当的方法，可以将奇异局势变为非奇异局势。

   假设面对的是局势(a, b)

   a) 若 a=b，则从两堆中同时取走 a 个物体，就变为了奇异局势(0, 0)
   b) 若 a=a[k], b>b[k],则从 b 堆取走 b-b[k]个物体，即变为了奇异局势(a[k],b[k])
   c) 若 a=a[k],a[k]<b<b[k],则同时从两堆取走 a[k] – a[b-a[k]]个物体，即变为奇异局势(a[b-a[k], a[b-a[k]] + b – a[k]])
   d) 若 a=b[k](b>a>b[k]>a[k]),则从 b 堆取走 b-a[k]个物体，即变为了奇异局势(a[k],b[k])

5. 由以上性质可知，必胜态为$(\lfloor k\varphi \rfloor, \lfloor k\varphi \rfloor + k)$

Fibonacci Nim

描述：有一堆个数为 n 的石子，游戏双方轮流取石子，满足：

1) 先手不能第一次把所有石子取完；
2) 之后每次取得石子数介于 1 和对手刚取的石子数的 2 倍之间（含端点）

最后取走石子的人获胜。

结论：先手必胜当且仅当 n 不是 Fibonacci 数.

证明：

1. Zeckendorf(齐肯多夫)定理：任何正整数都可以唯一表示成若干个不连续的斐波那契数之和。这种和式称为齐肯多夫表述法(或 Fibonacci 展开)。

   证明：

   a) 可表示性：考虑最大的 i 使得 $F(i) \le n < F(i+1)$，若 $n = F(i)$则命题成立。否则考虑正整数 $n' = n – F(i)$. 已知 $0 < n' < n$，所以可应用数学归纳法。故 n 可以表示成若干斐波那契数列之和。
   b) 不连续性：同理存在下标 j 使得 $F(j) \le n' < F(j+1)$, $n'' = n – F(i) – F(j) \ge 0$, 若 i 和 j 是相邻的整数(即 $i = j+1$), 则 $n'' = n – F(i) – F(i-1) = n – F(i+1) \ge 0$, 矛盾。
   c) 唯一性：对于 n 的两种 Fibonacci 展开，其最大项分别为 F(i)和 F(j). 若 $i = j$，由归纳假设 $n' = n – F(i)$的展开唯一，则两种展开相同；若 $i \ne j$，不妨设 $i > j$，则由不连续性，$n = F(j) + \ldots < F(j+1) \le F(i)$矛盾。

2. 设 n 的 Fibonacci 展开为 $F(n_1) + F(n_2) + \ldots + F(n_k)$,其中 $n_1 > n_2 > \ldots > n_k$, 则一个状态为必胜态 iff.先手取走 $F(n_k)$是合法的。

证明：(数学归纳法)

a) 设先手取走 $F(n_k)$是合法的，则先手可以选择取走 $F(n_k)$。若 $k = 1$，显然状态必胜。否则，由 $F(n_{k-1}) = F(n_{k-1} - 1) + F(n_{k-1} - 2) > 2F(n_{k-1} - 2) \geq 2F(n_k)$ (由 Fibonacci 展开的不连续性)，可知后手无法完全取走 $F(n_{k-1})$，后手必败，先手必胜

b) 设先手无法完全取走 $F(n_k)$. 设先手取走的石子数为 $x > 0$，考虑 $F(n_k) - x$ 的 Fibonacci 展开 $F(m_1) + F(m_2) + \ldots + F(m_t)$，只需证 $F(m_t) \leq 2x$，由归纳假设，后手将必胜，先手必败。

   下面证明 $F(m_t) \leq 2x$. 假设 $F(m_t) > 2x$, 则 $x < F(m_t) / 2 = (F(m_t - 1) + F(m_t - 2)) / 2 < F(m_t - 1)$，则 $F(n_k) < F(m_1) + F(m_2) + \ldots + F(m_t) + F(m_t - 1) < F(m_1 + 1)$ (由 Fibonacci 展开的不连续性)，而显然 $n_k > m_1$, 即 $F(n_k) \geq F(m_1 + 1)$，矛盾。

k-Nim（k 倍动态减法游戏）

描述：有一堆个数为 n 的石子，游戏双方轮流取石子，满足：

3) 先手不能第一次把所有石子取完；
4) 之后每次取得石子数介于 1 和对手刚取的石子数的 k 倍之间（含端点）

最后取走石子的人获胜。

结论：设数组$\{a_i\}$, $\{r_i\}$满足以下条件：

$$r_0 = 0,\ a_i = r_{i-1} + 1,\ r_i = a_i + r_j,\ j = \max\{j \mid ka_j < a_i,\ 1 <= j <= i - 1\}$$

先手必胜当且仅当 n 是 a 数组的某一项.

代码：已经通过 zju3599，上次使用 2012-8-27

```
//若先手必胜，则返回一种必胜的取法；否则返回-1
int solve(int n, int k){ //每次取得的石子数介于 1 和对手刚取的石子数的 k 倍之间(含端点)
	if (k == 1){
		int r = lowbit(n);
		return r == n ? -1 : r;
	}
	static const int maxn = 1000000;
	static int a[maxn], r[maxn];
	int i, j;
	for (i = 1, j = 0; ; ++i){
		a[i] = r[i-1] + 1;
		while (k * a[j+1] < a[i]) ++j;
		r[i] = a[i] + r[j];
		if (r[i] >= n) break;
	}
	n -= a[i--];
	if (n == 0) return -1;
	for (;n; n -= a[i])
		while (n < a[i]) --i;
	return a[i];
}
```

楼天城 Nim 游戏

描述：每次只能从一堆取，至少取 1 个，取过后还可以把这堆石头任意分配到其他堆上（这些堆必须有石头——其实可以没石头也不影响结论），当然也可以不分配。先取光着胜。

结论：总之，先手必败 iff.偶数堆石子且石子数($a_1, a_2, \ldots, a_{2k}$)($a_1 \le a_2 \le \ldots \le a_{2k}$)满足 $a_{2i-1} = a_{2i}$ ($1 \le i \le k$)，即($a, a, b, b, c, c, \ldots$)型分布。

证明：

1. (0, 0, 0, …)显然是必败态
2. 对于奇数堆石子($a_1, a_2, \ldots, a_{2k}, a_{2k+1}$)($a_1 \le a_2 \le \ldots \le a_{2k} \le a_{2k+1}$)，只需将第 2k+1 堆取($a_2 - a_1$)颗石子放入第 1 堆, ($a_4 - a_3$)颗石子放入第 3 堆, …, ($a_{2k-1} - a_{2k}$)颗石子放入第 2k-1 堆，其余取走即可转移到必败态($a_2, a_2, a_4, a_4, \ldots, a_{2k}, a_{2k}$)
3. 对于偶数堆石子($a_1, a_2, \ldots, a_{2k}$)($a_1 \le a_2 \le \ldots \le a_{2k}$)，只要 $a_{2i-1} = a_{2i}$ ($1 \le i \le k$)不总成立，则可以从第 2k 堆取 $a_3 - a_2$ 颗放入第 2 堆，$a_5 - a_4$ 颗放入第 4 堆，…，$a_{2k} - a_{2k-1}$ 颗放入第 2k 堆，然后取走至剩下 a1 颗，即可转移到必败态($a_1, a_2, a_2, a_3, a_3, \ldots, a_{2k-1}, a_{2k-1}, a_1$)
4. 对于偶数堆石子($a_1, a_2, \ldots, a_{2k}$)($a_1 \le a_2 \le \ldots \le a_{2k}$)，若 $a_{2i-1} = a_{2i}$ ($1 \le i \le k$)总成立，如按照 3 方法，则取走的石子数为 0(与要求不符)。若先手从第 i 取出 b_i 颗石子，并放入第 j 堆-bj 颗石子(j ≠ i)，将状态转移到($a_1 + b_1, a_2 + b_2, \ldots, a_{2k} + b_{2k}$)，可后手可以在第 i'堆取出-bj 颗石子放入第 j'堆(j ≠ i, j ≠ i')，并取走石子时石子数为 $a_i + b_i$，最后回到必败态。其中 i 为奇数时，i' = i + 1; 否则 i' = i - 1

阶梯 Nim 游戏

描述：有 n 级阶梯，每级有一些石子，每次可以在某一级台阶上拿一些石子到下一级台阶。到地上（台阶 0）后硬币自动消失。
结论：设台阶 i 有 x_i 颗石子，则状态(x1, x2, …)为 P 状态当且仅当状态(x1, x3, x5, ..)对于 Nim 游戏是 P 状态。

Euclid Game 欧几里德游戏

局势是一个正整数数对(x,y)，两人轮流操作，每次操作是将较大数 -= 若干倍的较小数，要求操作后仍为正整数，当到达局势(d,d)时(d = gcd(x,y))，游戏结束。换句话说(d,d)是必败态。
结论：

$$\mathrm{sg}(x, y) = \left\lfloor \left| \frac{x}{y} - \frac{y}{x} \right| \right\rfloor$$

证明详见 http://www.gabrielnivasch.org/academic/publications#euclid

Rim 游戏

描述：一开始平面上有 n 个点，两人轮流画圈，要求每个圈至少经过一个点，圈两两不相交，

谁不能画谁输。

结论： $sg_{Rims}(i) = i$

翻硬币游戏

n枚硬币排成一排，有的正面朝上，有的反面朝上。我们从左开始对硬币按1到n编号。

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。

谁不能翻谁输。

有这样的结论：局面的 sg 值为局面中每个正面朝上的棋子单一存在时的 SG 值的异或和。

如果每次只能翻连续的几枚，则 Ssg = ⊕(f[i] | 左边第 i 个硬币正面朝上, i=1..n)

其中

$$f(i) = \begin{cases} i & i = 2^k \\ f\left(i - 2^k\right) & i = 2^k + j \end{cases}$$

即 f[i] = (f[i] == highbit(i) ? i : f[i – highbit[i]]);

Green Hackenbush 无向图删边游戏

树的删边游戏

给出一个有N个点的树，有一个点作为树的根节点。

游戏者轮流从树中删去边，删去一条边后，不与根节点相连的部分将被移走。

谁无路可走谁输。

叶子节点的 SG 值为 0；中间节点的 SG 值为它的所有(直接)子节点的 SG 值。（根节点的 SG 值即为所求）

无向图的删边游戏

一个无向连通图，有一个点作为图的根。

游戏者轮流从图中删去边，删去一条边后，不与根节点相连的部分将被移走。

谁无路可走谁输。

Fusion Principle 定理：将图中的任意一个偶环缩成一个新点，任意一个奇环缩成一个新点加一个新边；所有连到原先环上的边全部改为与新点相连。这样的改动不会影响图的 SG 值。

证明(不存在两个环相交的时)：

1. 对于长度为奇数的环，去掉其中任意一条边后，剩下两个链长度同奇偶，异或后 SG 不可能为奇数，而它若从中间断开，SG 和恰好为 0。所以奇环的 SG 值为 1

2. 对于长度为偶数的环，去掉其中任意一条边后，剩下两个链长度就行不同，异或后 SG 不可能为偶数，所以偶环的 SG 为 0

SOS 游戏

先手胜	后手胜	平局
对于 n >= 7 奇数	对于 n >= 14 偶数	其余情况

取石子杂题

hdu 4388 Stone Game II

n 堆石子，每次操作是把其中的一堆石子(x 个)，变成 y 和 x ^ y 两堆，或者 y 和 x ^ (2*y)两堆，其中 0 < y < x, 0 < x ^ y < x, 且只能用有限次后者，以保证可以终止。两个轮流操作，无法操作者失败，问必胜必败

答：t = sum{ pairity(x_i) + 1 : 1 <= i <= n }是不变量，所以 t 为偶数<->先手必败(P)

Nim 积

Tartan 定理

若记 n 个游戏 $G_1, G_2, \ldots, G_n$ 的积 G(V, E) = $G_1G_2\ldots G_n$ 为：

(i) $\mathrm{V} = \mathrm{V}_1 \times \mathrm{V}_2 \times \ldots \times \mathrm{V}_n$(笛卡尔积)

(ii) $\mathrm{E} = \{(xy) | x_i^* \in f_i(x_i)\}$，其中$\mathrm{x} = (x_1, x_2, \ldots, x_n), y = \sum_{y_i \in \{x_i, x_i^*\}}(y_1, y_2, \ldots, y_n)$。这里求和号表示相同游戏规则下，不同状态所表示的不同游戏的和。

则，$\mathrm{SG}(x) = \mathrm{SG}(x_1) \otimes \mathrm{SG}(x_2) \otimes \ldots \otimes \mathrm{SG}(x_n)$，其中$\mathrm{x} = (x_1, x_2, \ldots, x_n) \in \mathrm{V}$

代码

```
//不懂，上次使用 2012-8-27，已经通过 hdu3404
int m[2][2] = {0, 0, 0, 1};

int NimProductPower(int x, int y) {
    if (x < 2) return m[x][y];
    int a = 0; while ( x >= (1 << (1 << a+1 )) ) ++a;
    int m = 1 << (1 << a);
    int qx = x / m, qy = y / m, ry = y % m;
    int d1 = NimProductPower(qx, qy), d2 = NimProductPower(qx, ry);
    return (m * (d1 ^ d2)) ^ NimProductPower(m / 2, d1);
}
```

```
int NimProduct(int x, int y) {
	if (x < y) swap(x, y);
	if (x < 2) return m[x][y];
	int a = 0; while ( x >= (1 << (1 << a+1 )) ) ++a;
	int m = 1 << (1 << a);
	int qx = x / m, rx = x % m, qy = y / m, ry = y % m;
	int c1 = NimProduct(qx, qy), c3 = NimProduct(rx, ry);
	int c2 = NimProduct(qx, ry) ^ NimProduct(rx, qy);
	return (m * (c1 ^ c2))    ^ c3 ^ NimProductPower(m / 2, c1);
}
```

组合计数

容斥原理

组合数学

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$$

$$|\bar{A} \cap \bar{B} \cap \bar{C}| = \Omega - |A| - |B| - |C| + |A \cap B| + |B \cap C| + |A \cap C| - |A \cap B \cap C|$$

——mobius 函数本质

$$\left|\bigcup_{i=1}^{n} A_i\right| = \sum_{k=1}^{n} (-1)^{k+1} \left(\sum_{1 \le i_1 < \cdots < i_k \le n} \left|A_{i_1} \cap A_{i_2} \cap \ldots \cap A_{i_k}\right| \right)$$

概率

$$\mathbb{P}(A \cup B \cup C) = \mathbb{P}(A) + \mathbb{P}(B) + \mathbb{P}(C) - \mathbb{P}(A \cap B) - \mathbb{P}(B \cap C) - \mathbb{P}(A \cap C) + \mathbb{P}(A \cap B \cap C)$$

$$\mathbb{P}\left(\bigcup_{i=1}^{n} A_i\right) = \sum_{k=1}^{n} (-1)^{k+1} \left(\sum_{1 \le i_1 < \cdots < i_k \le n} \mathbb{P}\left(A_{i_1} \cap A_{i_2} \cap \ldots \cap A_{i_k}\right) \right)$$

其他形式(莫比乌斯反演定理)

$$g(A) = \sum_{S:S \subseteq A} f(S) \ \Rightarrow \ f(A) = \sum_{S:S \subseteq A} (-1)^{|A|-|S|} g(S)$$

m 个球放入 n 个盒子

m 个球	n 个盒子	盒子可否空	方案数
不同	不同	可空	n^m
不同	不同	不空	n! * S(m, n) = $\sum_{i=0}^{n}(-1)^i C_n^i (n-i)^m$（容斥原理）
不同	相同	可空	$\sum_{k=1}^{\min\{n,m\}} S(m,k) = \frac{1}{n!}\sum_{k=1}^{\min\{n,m\}}\sum_{i=0}^{k}(-1)^i C_k^i (k-i)^m$ （分类讨论）
不同	相同	不空	第二类 Stirling 数 S(m, n)
相同	不同	可空	C_{n+m-1}^{n-1}（挡板） 从 m 种元素可重复地取出 n 个元素的放法数目 $x_1 + x_2 + \cdots + x_n = m, x_i \ge 0$的解的个数 若球不必用完，答案为$C_{n+m}^{n}$，即为 从 m 中元素中可重复取出 n+1 个元素的方法数 $x_1 + x_2 + \cdots + x_n \le m, x_i \ge 0$的解的个数
相同	不同	不空	C_{m-1}^{n-1}（挡板） $x_1 + x_2 + \cdots + x_n = m, x_i \ge 1$的解的个数
相同	相同	可空	g[i][j]表示将 i 个无区别的球放入 j 个无区别的盒子，盒子可以为空(已经通过 cugb1095) i ≥ j 时，g[i][j] = g[i][j-1] + g[i-j][j]; i < j 时，g[i][j] = g[i][j-1]; 其中：g[0][0..n] = 1, g[1..m][1] = 1
相同	相同	不空	N ≥ 2 时，g(m, n) – g(m, n-1) N = 1 时，1(也可以令 g[1..m][0] = 0，将两种情况合并)

- 将 m 个相同的球，放入 m+个相同的盒子，盒子或空，或球数>=k，求方法数
  设 f(i, j) : i 个相同球，放入 i+个相同盒子，非空盒子球数>=k 的方法数
  转移函数： i >= j 时，f(i, j) = f(i, j+1) + f(i-j, j)　(枚举有没有盒子恰好放 j 个球)
  i < j 时，f(i, j) = f(i, j+1)
  初值：f(0, 0..k) = 1
  目标函数：f(m, k)
- 将 n 个不同的球放入 k 个相同的盒子，盒子不可为空，并把每个盒子中的球排成一个环排列(Arrangements of an n element set into k cycles)：
  用 s(p, k)表示满足上述要求的将 p 个球放入 k 个盒子的方法数，成为第一类 Stirling 数
  递推：s(p, k) = (p-1)s(p-1, k) + s(p-1, k-1)，其中 s(p, p) = 1(p ≥ 0), s(p, 0) = 0(p ≥ 1)
- **轮状病毒**：圆周上 n 个点和圆心通过 n 条半径和 n 段圆弧连接形成 n 元环状基，求其生成树个数。
  做法是递推。生成树的结构可以看作若干段分离圆弧和圆心分别连接，而每段长度为 *l* 的圆弧和圆心连接方案数为 *l*。考虑用 *f[i]*表示 *i* 个点分裂成圆弧的方案数，转移方程是：*f[i] = sum{f[j] * (i - j), j = 0..i-1}*. 最后枚举一下分裂点，破坏为链即可： *res[i] = sum{f[j] * (i - j)*^{2}*, j = 0..i-1}*
  如果考虑翻转和旋转，可以使用 burnside 引理计算。(摘自 ftiasch)

正整数的无序分拆数

用 B(n, k)表示 n 的 k 分拆的个数(把 n 个相同球放入 k 个相同的不空的盒子)

性质(根据分拆的 Ferrers 图可以证明)

- n 的 k 分拆的个数 = n 的最大分部量为 k 的反拆数
- n 个自共轭分拆的个数 = n 的各分部量都是奇数且两两不等的分拆数
- n 个分部量两两不等的分拆个数 = n 个各分布量都是奇数的分拆的个数(?)

第一类 Stirling 数

TODO：http://en.wikipedia.org/wiki/Stirling_numbers_of_the_first_kind
Striling number of the first kind, $s(n, k) = (-1)^n c(n, k)$
signless Striling number of the first kind, c(n, k)
递推：c(n, k) = (n-1)c(n-1, k) + c(n-1, k-1), n, k >= 1 (同第二类 Striling 数)
初值：c(n, k) = 0 for n <= 0 or k <= 0, except c(0, 0) = 1
意义：1..n 的排列，恰好有 k 个环的排列数
意义'：1..n 的排列，恰好有 k 个 left-to-right maxima(对于 p[1..n]的一个前缀 p[1..i], p[i]是极大值)排列数。
意义''：设 n, k >= 0, 对于整数序列(a_1, ..., a_n) (0 <= a_i <= n - i)，恰有 k 个 a_i 的值为 0 的序列数。

第二类 Stirling 数

S(m, n)：把 m 元集合分成 n 类的方案数，称为**第二类 Stirling 数**，常记为$\left\{ {n \atop k} \right\}$
递推公式：S(i, j) = S(i-1, j-1) + j * S(i-1, j) 2 ≤ j ≤ i-1; S(0, 0) = 1, S(0, i) = S(i, 0) = 0;

通项公式：$S(i, j) = \frac{1}{j!}\sum_{k=0}^{j}(-1)^{j-k}\binom{j}{k}k^i$（容斥原理）

注：$j!\,S(i, j) = \sum_{k=0}^{j}(-1)^{j-k}\frac{j!}{k!(j-k)!}k^i$，要求 j! S(i, j)可以预处理出 1!, 2!, ... mod p 的结果

i/j	1	2	3	4	5	6	7	8
1	1							
2	1	1						
3	1	3	1					
4	1	7	6	1				
5	1	15	25	10	1			
6	1	31	90	65	15	1		
7	1	63	301	350	140	21	1	
8	1	127	966	1701	1050	266	28	1

```
for (int i = 1; i < maxk; ++i) {
```

```
        s[i][1] = s[i][i] = 1;
        for (int j = 2; j < i; ++j) s[i][j] = (s[i-1][j-1] + j * s[i-1][j]) % module;
    }
```

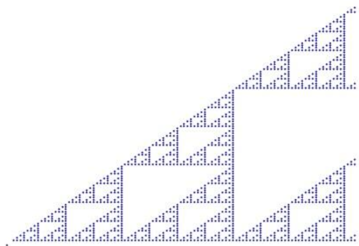
代码 – 求 j! * S(i, j) TODO 重写

```
int f_mod(int a, int b, int M){ //O(b * loga)已经通过 bupt1884
    long long r = 0, t = 1; // t = C_mod(b, i, M);
    for (int i = 0; i <= b; i++){
//        r += (i % 2 == 0 ? 1 : -1) * mul_mod(C_mod(b, i, M), pow_mod(b-i, a, M), M);
        if (i) t = mul_mod(t, b - i + 1, M), t = div_mod(t, i, M);
        r += (i % 2 == 0 ? 1 : -1) * mul_mod(t, pow_mod(b-i, a, M), M);
    }
    return mod(r, M);
}
```

s(n, k) mod 2

$$\left\{ {n \atop k} \right\} \equiv \binom{z}{w} \pmod 2, \quad z = n - \left\lceil \frac{k+1}{2} \right\rceil, \ w = \left\lfloor \frac{k-1}{2} \right\rfloor.$$

```
int c(int n, int m){
    int r = 0;
    for (int i = n; i; i /= 2) r += i / 2;
    for (int i = m; i; i /= 2) r -= i / 2;
    for (int i = n-m; i; i /= 2) r-= i / 2;
    return r ? 0 : 1;
}
```



```
int s(int n, int k){
    int z = n - ((k + 1) / 2 + (k + 1) % 2, w = (k - 1) / 2; return c(z, w);
}

int main(){ //poj1430
    int _, n, k; scanf("%d", &_);
    while (_--) scanf("%d%d", &n, &k), printf("%d\n", s(n, k));
}
```

Bell 数

将 n 个不同的球放入 n+个相同的盒子，盒子可以为空：(已经通过 hdu1028)
相当于求 n 元集合的所有划分数，记为 B(n)，成为 Bell 数
定义式：B(n) = $\sum_{k=1}^{n} S(n,k)$

递推式：$B(n+1) = \sum_{k=0}^{n} C_n^k B(k)$ ——打表求时间复杂度 $O(n^2)$
int 能存到 B_{16}, LL 能存到 B_{25}

B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
1	1	2	5	15	52	203	877	4140	21147	115975

Bernoulli 数

$$B_m(n) = n^m - \sum_{k=0}^{m-1} \binom{m}{k} \frac{B_k(n)}{m-k+1}, B_0(n) = 1$$

Fibonacci 数

long long 能存到第 91 项，int 能存到第 45 项

F_{-1}	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
0	1	1	2	3	5	8	13	21	34	55	89

$S_n = F_0 + F_1 + \cdots + F_n = F_{n+2} - 1$

$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n = \mathrm{round}\left(\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n\right) = \begin{pmatrix}1 & 1\\ 1 & 0\end{pmatrix}^n.\mathrm{data}[0][0]$$

$$F_n = C_{n-1}^0 + C_{n-2}^1 + C_{n-3}^2 + \cdots + C_{n-k}^{k-1}, \quad \left(k = \left\lfloor\frac{n+1}{2}\right\rfloor\right)$$

$F_0 + F_2 + \cdots + F_{2n} = F_{2n+1}$

$F_1 + F_3 + \cdots + F_{2n-1} = F_{2n} - 1$

$F_0^2 + F_1^2 + \cdots + F_n^2 = F_n F_{n+1}$

$F_{n+m} = F_{m-1}F_{n+1} + F_{m-2}F_n \ (m \geq 2)$

应用：S = {1, 2, …, n}的不含相邻整数的子集数为 F_{n+1}

性质：任何一个正整数 n 都可以唯一写成不同的 Fibonacci 数的和

Fi-binary 数：二进制表示没有两个连续 1 的数，第 n 个 Fi-binary 数为 n 的 Fibnacci 进制表示。

If w, x, y, z are four consecutive(连续的) Fibnacci numbers, then (wz, 2xy, yz-wx) is a Pythagorean triple

GCD(F_m, F_n) = $F_{GCD(m, n)}$

循环节长度

```
//上次使用 2012.9.17
typedef map<LL, int> factors_t;

void factorize(LL n, factors_t &factors){
	if ( isprime(n)){
```

```
            ++factors[n];
        } else {
            LL x = rho(n); factorize(x, factors); factorize(n / x, factors);
        }
}

LL fib_mod(LL i, LL n){
        if (i == 0) return 0;
        mat22 a;
        a.v00 = a.v01 = a.v10 = 1; a.v11 = 0;
        return pow_mod(a, i, n).v00;
}

LL period(LL n){ // 求 Fibnacci mod n 的循环节长度
#define foreach(iter,c) for( typeof( c.begin() ) iter=c.begin(); iter != c.end(); ++iter)
        factors_t factors;
        if (n > 1) factorize(n, factors);
        LL ret = 1;
        foreach(iter, factors){
            LL p = iter->first, k; int e = iter->second;
            if (p == 5){
                k = 20;
            } else if (p == 2){
                k = 3;
            }else {
                LL pmod5 = p % 5;
                if (pmod5 == 1 || pmod5 == 4){
                    k = p - 1;
                } else {
                    k = 2 * p + 2;
                }
                factors_t kfactors;
                factorize(k, kfactors);
                foreach(kiter, kfactors){
                    LL kp = kiter->first; int ke = kiter->second;
                    while (ke--){
                        LL newk = k / kp;
                        if ( fib_mod(newk - 1, p) == 0 && fib_mod(newk, p) == 1 ){
                            k = newk;
                        } else {
                            break;
                        }
                    }
                }
```

```
		}
		while (--e) k = k * p;
		ret = lcm(ret, k);
	}
	return ret;
}

int main(){
	cout << period(243) << endl;
	cout << period(969929135467348530LL) << endl; // 5s
}
```

Lucas 数

L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}
2	1	3	4	7	11	18	29	47	76	123

$$\mathrm{L}_n = \begin{cases} 2 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \mathrm{L}_{n-1} + \mathrm{L}_{n-2} & \text{if } n > 1 \end{cases}$$

$$\mathrm{L}_n = \varphi^n + (1-\varphi)^n = \varphi^n + (-\varphi)^{-n} = \left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$\mathrm{L}_{-n} = (-1)^n \mathrm{L}_n$$
$$\mathrm{L}_n = \mathrm{F}_{n-1} + \mathrm{F}_{n+1}$$
$$\mathrm{F}_n = \frac{1}{5}(\mathrm{L}_{n-1} + \mathrm{L}_{n+1})$$

Combinatorial Number 组合数

打表 int 能存到 n = 33, long long 能存到 n = 66

组合数$\mathrm{C}_\mathrm{n}^\mathrm{k}$为奇数当且仅当 (n&k) == n

n/k	0	1	2	3	4	5	6	7	8
0	1								
1	1								
2	1	1							
3	1	2	1						
4	1	3	3	1					
5	1	4	6	4	1				
6	1	5	10	10	5	1			
7	1	6	15	20	15	6	1		
8	1	7	21	35	35	21	7	1	
9	1	8	28	56	70	56	28	8	1

Pascal 公式

$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$

$C_n^{n_1,n_2,\dots,n_k} = C_{n-1}^{n_1-1,n_2,\dots,n_k} + C_{n-1}^{n_1,n_2-1,\dots,n_k} + \dots + C_{n-1}^{n_1,n_2,\dots,n_k-1}$

恒等式

$C_n^k = \frac{n}{k} C_{n-1}^{k-1}$

$(C_n^0)^2 + (C_n^1)^2 + \dots + (C_n^n)^2 = C_{2n}^n$

$C_k^k + C_{k+1}^k + \dots + C_{n-1}^k + C_n^k = C_{n+1}^{k+1}$

$C_{n+1}^{p+1} = \sum_i C_i^q C_{n-i}^{p-q}$, 对任意 q 成立

广义组合式定义

$C_n^{n_1,n_2,\dots,n_k} = \binom{n}{n_1 \quad n_2 \quad \dots \quad n_k} = \frac{n!}{n_1!\, n_2! \dots n_k!} (n_1 + n_2 + \dots + n_k = n)$

广义二项式定理

$(x_1 + x_2 + \dots + x_k)^n = \sum_{c_1+c_2+\dots+c_k=n} \binom{n}{n_1 \quad n_2 \quad \dots \quad n_k} {x_1}^{c_1} {x_2}^{c_2} \dots {x_k}^{c_k}$

Catalan 数

定义 1：将正 n 边形用对角线剖分为三角形的数 C_{n-2}(同构的算多次)

定义 2：n 个数的乘积 $a_1a_2\dots a_n$ 的不同结合方法数 C_{n-1}

定义 3：C_{n-1}：n 个**叶子**节点满二叉树(认为左右子树有序)的数目、n 个节点**任意**二叉树(认为左右子树有序)的数目

定义 4：在整数坐标平面的格子上，从点(0, 0)到点(n, n)只允许垂直步进和水平步进的路程，要求任何中间点(a, b)满足 a ≤ b，即必须在 y = x 上方走，路径条数 C_n；除两端点外恒保证 a < b 的路径条数是 C_{n-1}

(如果是从点(0, 0)到点(p, q) (p < q)，路径条数为$C_{p+q-1}^p - C_{p+q-1}^{p-1} = \frac{q-p}{q+p} C_{p+q}^q$)

(如果是从点(0, 0)到点(p, q) (p ≤ q)，路径条数为$C_{p+q}^p - C_{p+q}^{p-1} = \frac{q-p+1}{q+1} C_{p+q}^q$) – 已过两题

对于有些题目，可能情景还要求结果乘以 p! * q!

递推：$C_n = C_0C_{n-1} + C_1C_{n-2} + \dots\dots + C_{n-1}C_0$

递推：$C_n = \frac{4n-2}{n+1} C_{n-1}$

通项：$C_n = \frac{1}{n+1} C_{2n}^n$ **(注意 C(m, n)打表时别忘算 C(0, 0) = 1)**

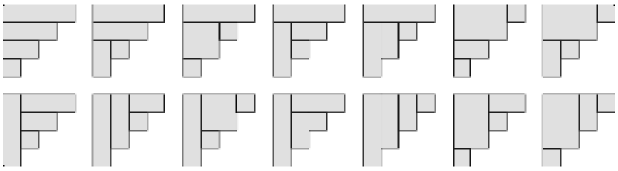
渐进函数：$C_n \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$

C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9	C_{10}	C_{11}	C_{12}	C_{13}
1	1	2	5	14	42	132	409	1430	4862	16796	58786	208012	752900

实例：

1. 一列火车 n 节车厢，依次编号为 1,2,3,⋯,n。每节车厢有两种运动方式，进栈与出栈，问 n 节车厢出栈的可能排列方式有 C_n 种。
2. 从圆上选择 2n 个点，将这些点成对连接所得的 n 条线段不相交的方法数 C_{n-1}

3. 合法的括号序列个数
4. 用 n 个长方形填充一个高度为 n 的阶梯状图形的方法个数为 Cn。下图为 n = 4 的情况:



求 Catalan 数 mod n

```
struct Zn {
	Zn(LL n, LL val) {
		setN(n); fill(cnt, cnt + pcnt, 0); v = 1; *this *= val;
	}
	void setN(LL n) {
		this->n = n; pcnt = 0;
		for (int i = 2; i * i <= n; ++i) {
			if ( n % i == 0 ) {
				p[pcnt++] = i;
				do n /= i; while ( n % i == 0 );
			}
		}
		if ( n > 1 ) p[pcnt++] = n;
	}
	Zn &operator *= (LL o) {
		for (int i = 0; i < pcnt; ++i)
			while ( o % p[i] == 0 )
				o /= p[i], ++cnt[i];
		v = v * o % n;
		return *this;
	}
	Zn &operator /= (LL o) {
		for (int i = 0; i < pcnt; ++i)
			while ( o % p[i] == 0 )
				o /= p[i], --cnt[i];
		v = v * inv(o) % n;
		return *this;
	}
	operator LL() const {
		LL ans = v;
		for (int i = 0; i < pcnt; ++i)
			for (int k = 0; k < cnt[i]; ++k)
				ans = ans * p[i] % n;
		return ans;
	}
```

```
private:
	LL n; LL p[25], pcnt;
	int cnt[25]; LL v;
	LL inv(LL v) {
		LL x, y;
		assert( exgcd(v, n, x, y) == 1 );
		x %= n; if ( x < 0 ) x += n;
		return x;
	}
};

void solve(int n, int m) { //zju3183
	Zn C(m, 1);
	LL ans = 0;
	for (int i = 1; i <= n; ++i) {
		C *= 4 * i - 2; C /= i + 1; ans += (LL)C;
	}
	cout << ans % m << endl;
}
```

Fuss-Catalan 数

定义：将正 kn+2 边形剖分为 k+2 边形的数 $C_{n,k}$

通项：$C_{n,k} = \frac{1}{n} C_{(k+1)n}^{n-1}$

拟 Catalan 数

$C_n^* = n!\,C_{n-1}$（C_n为 Catalan 数）

递推：$C_n^* = (4n-6)C_{n-1}^*$

通项：$C_n^* = (n-1)!\,C_{2n-2}^{n-1} = \frac{(2n-2)!}{(n-1)!}$

C_1^*	C_2^*	C_3^*	C_4^*	C_5^*	C_6^*
1	2	12	120	1680	30240

应用：求 a1, a2, …, an 的和有C_n^*种结合方式（加法满足交换率）

错排数

$$D_n = n!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^n \frac{1}{n!}\right)$$

有结论

$$\left|\frac{D_n}{n!}-e^{-1}\right|<\frac{1}{(n+1)!}$$

D_n是最接近$n!/e$的整数，n≥7 时e^{-1}和$D_n/n!$至少 3 位小数相同

错排概率

$$D_n/n! \approx e^{-1}$$

递推公式

$$D_n=(n-1)(D_{n-1}+D_{n-2})$$
$$D_n=nD_{n-1}+(-1)^n$$

D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
0	1	2	9	44	265	1854	14833	133496	1334961

m 组配对，恰好有 n 组错排，数：$C_m^n \cdot D_n$

有禁止模式的排列数

用 Q_n 表示{1, 2, …, n}的，不出现 12, 23, …, (n-1)n 这些模式的全排列个数，并规定 $Q_1 = 1$

则 $Q_n = D_n - D_{n-1}$

若既不能出现 i->i,又不能出现 i->i+1 (n->1)，则排列数为

$$N=\sum_{k=0}^{n}\frac{2n}{2n-k}\binom{2n-k}{k}(n-k)!\,(-1)^k$$

大 Schröder 数

- 在整数坐标平面的格子上，从点(0, 0)到点(n, n)只允许垂直步进和水平步进的路程，要求任何中间点(a, b)满足 a ≥ b，即必须在 y = x 下方走(称为下对角线格路径)，路径条数 C_{n+2}(C_n 是第 n 个 Catalan 数)
- 如果是从点(0, 0)到点(p, q)(p ≥ q)，路径条数为$\frac{p-q+1}{p+1}C_{p+q}^{q}$
- 如果还允许对角线步进，设$R(p,q:rD)$为恰好使用 r 个对角线步进 D 的从(0, 0)到(p, q) (p ≥ q)的下对角线格路径的条数则：

$$R(p,q:rD)=\frac{p-q+1}{p-r+1}C_{p+q-r}^{p-r,q-r,r}$$

设$R(p,q)$为从(0, 0)到(p, q) (p ≥ q)的可以任意使用对角线步进的下对角线格路径的条数，则

$$R(p,q)=\sum_{r=0}^{q}R(p,q:rD)=\sum_{r=0}^{q}\frac{p-q+1}{p-r+1}C_{p+q-r}^{p-r,q-r,r}$$

现在设 p=q=n，则从(0, 0)到(p, q) (p ≥ q)的可以任意使用对角线步进的下对角线格路径称为 Schröder 路径。**大 Schröder 数** R_n 是从从(0, 0)到(p, q) (p ≥ q)的 Schröder 路径条数。则

$$R(n)=R(n,n)=\sum_{r=0}^{q}\frac{1}{n-r+1}\frac{(2n-r)!}{r!\,\left((n-r)!\right)^2}$$

R(0)	R(1)	R(2)	R(3)	R(4)	R(5)	R(6)

1	2	6	22	90	394	1806

大 Schröder 数等于从(0, 0)到(2n, 0)步进为(1, 1), (1, -1)的从不会到达水平轴上方的格路径数(常被称为 **Dyck 路径**)

小 Schröder 数

对于 n≥1，小 Schröder 数 s(n)定义为符号序列 a_1, a_2, ..., a_n 的添加括号的方法数。(可以把连续两个或更多项用括号括起来)

关系式：s(1) = 1, s(n+1) = R(n) / 2 (n≥1)

递推式：(n+2)s(n+2) – 3(3n+1)s(n+1) + (n-1)s(n) = 0 (n ≥ 1)

s(1)	s(2)	s(3)	s(4)	s(5)	s(6)	s(7)
1	1	3	11	45	197	903

应用：n+1 条边的凸多边形区域的剖分数(不一定最终分成三角形)，为小 Schröder 数 s(n)

第一类欧拉数

意义：1..n 的排列，有 k 个直接升序的排列数，记为$\left\langle {n \atop k} \right\rangle$，或 A(n, k)

递推：$\left\langle {n \atop k} \right\rangle = (n-k+1)\left\langle {n-1 \atop k-1} \right\rangle + k\left\langle {n-1 \atop k} \right\rangle$

性质：$\left\langle {n \atop k} \right\rangle = \left\langle {n \atop n-1-k} \right\rangle$

n/k	0	1	2	3	4	5	6	7	8
0	1								
1	1								
2	1	1							
3	1	4	1						
4	1	11	11	1					
5	1	26	66	26	1				
6	1	57	302	302	57	1			
7	1	120	1191	2416	1191	120	1		
8	1	247	4293	15619	15619	4293	247	1	
9	1	502	14608	88234	156190	88234	14608	502	1

第二类欧拉数

意义：多重集{1, 1, 2, 2, ..., n, n}的排列，有 k 个直接升序的排列数

递推：$\left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle = (2n-k-1)\left\langle\!\left\langle {n-1 \atop k-1} \right\rangle\!\right\rangle + (k+1)\left\langle\!\left\langle {n-1 \atop k} \right\rangle\!\right\rangle$

n/k	0	1	2	3	4	5	6	7	8
0	1								
1	1								

2	1	2							
3	1	8	6						
4	1	22	58	24					
5	1	52	328	444	120				
6	1	114	1452	4400	3708	720			
7	1	240	5610	32120	58140	33984	5040		
8	1	494	19950	195800	644020	785304	341136	40320	
9	1	1004	67260	1062500	5765500	12440064	11026296	3733920	362880

生成树计数- Cayley 定理

无向图生成树计数

//已经通过 spoj HIGH，参见 41 页 gao 函数

1. Degree Matrix, D[G]: a diagonal matrix with vertex degrees on the diagonals.
2. Adjacency Matrix, A[G]
3. Laplacian Matrix / Kirchhoff Matrix, L[G] = D[G] – A[G] (这里 G 不允许有自环)

$$\{\mathrm{L[G]}\}_{\mathrm{ij}} = \begin{cases} \deg_G(\mathrm{v_i}) & \text{if } \mathrm{i} = \mathrm{j} \\ -\epsilon_{\mathrm{ij}} & \text{if } \mathrm{i} \neq \mathrm{j} \end{cases}$$

4. 注：$\epsilon_{\mathrm{ij}} = \epsilon_{\mathrm{ji}}$, G 不一定是简单图，但要去掉自环(loop)
5. Kirchhoff's Matrix-Tree Theorem: G 的所有不同的生成树的个数等于其 Laplacian 矩阵 L[G] 任何一个 n-1 阶主子式的行列式的绝对值.(如果去掉的是第 i 行第 i 列，则不需要取绝对值~~~)
6. 推论：完全图 Kn 有 n^{n-2} 棵生成树(彼此同构的子树多次计数)，有 n 个不同顶点的无根树数目为 n^{n-2}。

最小生成树计数

LL module; //已经通过 2012 金华 regional online

```
struct mat {
	int n; LL dat[105][105];
	void init(int n) {
		this->n = n; rep(i, n) rep(j, n) dat[i][j] = 0;
	}
	LL det(){
		static LL a[105][105];
		rep(i, n) rep(j, n) a[i][j] = dat[i][j] % module;
		LL det = 1;
		for (int k = 0; k < n; ++k) {
			for (int i = k + 1; i < n; ++i) {
				while ( a[i][k] != 0 ) {
```

```
                LL t = a[k][k] / a[i][k];
                for (int j = k; j < n; ++j) {
                    a[k][j] -= a[i][j] * t; a[k][j] %= module;
                    swap( a[k][j], a[i][j] );
                }
                det = -det;
            }
        }
        if ( a[k][k] == 0 ) return 0;
        det = det * a[k][k] % module;
    }
    if ( det < 0 ) det += module;
    return det;
  }
};

LL gao(const vector< pair<int, int> > &v) {
    vector<int> x;
    for (int i = 0; i < v.size(); ++i) {
        int a = v[i].first, b = v[i].second;
        x.push_back(a); x.push_back(b);
    }
    sort( x.begin(), x.end() ); x.erase( unique(x.begin(), x.end()), x.end() );
    mat mm; mm.init( x.size() );
    for (int i = 0; i < v.size(); ++i) {
        int a = lower_bound(x.begin(), x.end(), v[i].first ) - x.begin();
        int b = lower_bound(x.begin(), x.end(), v[i].second) - x.begin();
        --mm.dat[a][b]; --mm.dat[b][a];
        ++mm.dat[a][a]; ++mm.dat[b][b];
    }
    mm.n--;
    return mm.det();
}

struct edge {
    int a, b, c;
    edge(int a, int b, int c) : a(a), b(b), c(c) { }
    bool operator < (const edge &o) const { return c < o.c; }
    bool operator == (const edge &o) const { return c == o.c;}
};

struct disjointset {
    vector<int> fa;
    void init(int n){ fa.resize(n + 1); rep(i, n + 1) fa[i] = i; }
```

```
	int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
	void unionset(int x, int y) { x = find(x); y = find(y); fa[x] = y; }
} ufset, ufsetlast;

void solve(int n, int m, int p) {
	module = p;
	vector< edge > E;
	for (int i = 0; i < m; ++i) {
		int a, b, c; scanf("%d%d%d", &a, &b, &c);
		E.push_back( edge(a, b, c) );
	}
	sort( E.begin(), E.end() );
	LL ans = 1 % module;
	ufset.init(n);
	for (int i = 0, j; i < m; i = j) {
		ufsetlast = ufset;
		for (j = i + 1; j < m && E[j] == E[i]; ++j)
			;
		for (int k = i; k < j; ++k) {
			int a = ufsetlast.find( E[k].a ), b = ufsetlast.find( E[k].b );
			ufset.unionset( a, b );
		}
		map< int, vector< pair<int, int> > > mm;
		for (int k = i; k < j; ++k) {
			int a = ufsetlast.find( E[k].a ), b = ufsetlast.find( E[k].b );
			if ( a == b ) continue;
			mm[ ufset.find( a ) ].push_back( make_pair( a, b ) );
		}
		for (map<int,vector<pair<int,int> > >::iterator iter=mm.begin();iter!=mm.end();++iter)
			ans = ans * gao( iter->second ) % module;
	}
	for (int i = 2; i <= n; ++i) if ( ufset.find(i) != ufset.find(1) ) ans = 0;
	cout << ans % module << endl;
}

int main(){ for (LL n, m, p; cin >> n >> m >> p && n + m + p; solve(n, m, p) ); }
```

有向图的生成树(内向树)计数

1. (内向树)An oriented spanning tree of G rooted at $r \in V$ is a spanning subgraph T = (V, A) such that
   - Every vertex $v \neq r$ has out degree 1.
   - r has ***out*** degree 0
   - T has no oriented cycles.

2. Laplacian Matrix, L[G] = D[G] – A[G]
   D[G] : a diagonal matrix with vertex ***out*** degrees on the diagonals.
3. Matrix-Tree Theorem: Let

$$\kappa(\mathrm{G}, r) = \#\{\text{oriented spanning trees of G rooted at r}\}$$

   and L_r be the Laplacian matrix of G with the row and column corresponding to vertex r crossed out. Then

$$\kappa(G, r) = \det(\mathrm{L}_r)$$

完全图的生成森林计数

定义：

planted forest: a graph for which every connected component is a (rooted) tree.

$p_k(n)$ = number of *planted forest* with k components on the vertex set [n]

$p_S(n)$ = number of *planted forest* on [n] with #S components, whose set of roots is S.

p(n) = number of plant forests on [n]

性质：

$\mathrm{p}_S(\mathrm{n}) = \mathrm{k}\,\mathrm{n}^{\mathrm{n-k-1}}$ for $\forall \mathrm{S} \subseteq \binom{[\mathrm{n}]}{k}$

Notice: #S = k, **n == k** 时$\mathbf{p}_S(\mathbf{n}) = \mathbf{1}$

$\mathrm{p}_k(\mathrm{n}) = \binom{\mathrm{n}}{k} \mathrm{k}\,\mathrm{n}^{\mathrm{n-k-1}} = \binom{\mathrm{n}-1}{k-1} \mathrm{n}^{\mathrm{n-k}}$

$\mathrm{p}(\mathrm{n}) = (\mathrm{n}+1)^{n-1}$

证明：

考虑生成森林的 prufer 序列，前 n-k-1 个空可以有 n 种填法，第 n-k 个空只有 k 种填法。

其他

n 个节点的度依次为 $d_1, d_2, \ldots, d_n$ 的无根树共有$\frac{(\mathrm{n}-2)!}{(d_1-1)!(d_2-1)!\ldots(d_n-1)!}$个，因为此时 Prufer 编码中的数字 i 恰好出现 $d_i - 1$ 次

区域划分计数(TODO：添加北大校赛那题)

欧拉公式：V(顶点数) – E(边数) + F(面数) = 2

定义$\mathrm{h}_\mathrm{n}^{(\mathrm{k})} = \mathrm{C}_\mathrm{n}^0 + \mathrm{C}_\mathrm{n}^1 + \cdots + \mathrm{C}_\mathrm{n}^\mathrm{k}$

性质

$\Delta \mathrm{h}_\mathrm{n}^{(\mathrm{k})} = \mathrm{h}_\mathrm{n}^{(\mathrm{k}-1)}$

$\mathrm{h}_\mathrm{n}^{(\mathrm{k})}$是 n 个一般位置的(k-1)维超平面分割 k-维空间所成的区域数

//三维情况已经通过 bnu4335

n 条直线将平面分成的区域数

关系式：$D_n = h_n^{(2)} = C_n^0 + C_n^1 + C_n^2 = 1 + \frac{n(n+1)}{2}$

递推：$D_n = D_{n-1} + n$, $D_1 = 2$

n 条 V 型折线将平面分成的区域数 -已经通过 hdu 递推求解专题练习
关系式：$D'_n = D_{2n} - 2n = 2n^2 - n + 1$

n 条 Z 型折线将平面分成的区域数 – 已经通过 cugb1042
关系式：$D'_n = D_{3n} - 5n$

平面 n 个互相交叠的椭圆分成的区域数 h_n
经过一个公共点的 n 个平面（但任意三个平面不过同一直线）把空间分成的块数
递推 $h_n = h_{n-1} + 2(n-1)$
通项 $h_n = n^2 - n + 2$

其他
对没有三条对角线交于一点的凸多边形，各边及对角线互不重叠的区域个数$C_n^4 - C_{n-1}^2$

n 对夫妻圆排列计数

n 对夫妻围以圆桌坐下，要求男女交错，设恰有 r 对夫妻座位相邻的方案数 2n! * M(n, r)，其中 2n!为女性的排列数，M(n, r)为男性的排列数，则

$$M(n,r) = \sum_{k=r}^{n} (-1)^{k-r} C_k^r \frac{2n}{2n-k} C_{2n-k}^k (n-k)!$$

设 $M_n = M(n, 0)$，有

M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}
-	0	1	2	13	80	579	4738	43387	439792	4890741	59216642

M 的(莱桑的)递推关系式为

$$(n-1)M_{n+1} = (n^2-1)M_n + (n+1)M_{n-1} + 4(-1)^n$$

可以证明

$$\lim_{n\to\infty} \frac{M_n}{n!} = e^{-2}$$

Sperner 定理

令 S 为 n 个元素的集合，则 S 的杂置最多包含$C_n^{n/2}$个集合。
其中杂置(clutter，又称反链)是 S 中的元素的组合的集合 C，是的 C 中没有组合包含另外一个组合。例如，若 S={a, b, c, d}，则它的一个杂置 C={{a, b}, {a, c}, {a, d}, {b, c, d}}。

圆域染色计数

一个圆被等分为 n 个相同的有区别的扇形 $D_1 \ldots D_n$，用 k 种颜色染色，要求相邻区域颜色不同，求方法数 a_n。
对于 a_n,讨论 D_1 和 D_{n-1} 是否颜色相同：
颜色相同，D_n 有 k-1 种染色方案，剩下的染色方案与 n-2 个域的染色方案一一对应；
颜色不同，D_n 有 k-2 种染色方案，剩下的染色方案与 n-1 个域的染色方案一一对应。
故：$a_i = (k-2) * a_{i-1} + (k-1) * a_{i-2}\ (i \ge 4)$, $a_1 = k$, $a_2 = k * (k-1)$, $a_3 = k * (k-1) * (k-2)$
(a_i 已经通过 cugb1249)
若旋转对称的，算作同一种方案，则由 buinside 引理，答案为

$$\frac{1}{n}\sum_{k=0}^{n-1} a_{\gcd(\mathrm{k,n})} = \frac{1}{n}\sum_{d|n} \Phi\left(\frac{n}{d}\right) a_d$$

Pòlya 计数法

Burnside 引理：设 G = {p_1, p_2, …, p_g}是目标集[1, n]上的置换群，G 将[1, n]分成 L 个等价类。设 $c_1(p_k)$是在置换 p_k 作用下不动点的个数(也就是长度为 1 的循环的个数)，则等价类的个数

$$\mathrm{L} = \frac{1}{|G|}\sum_{i=1}^{g} c_1(p_i)$$

Pòlya 定理：设 G={a_1, a_2, …, $a_{|G|}$}是 N={1,2,..,N}上的置换群，现用 m 中颜色对这 N 个点染色，则不同的染色方案数为：

$$S = (m^{c1}+m^{C2}+\ldots+m^{C|G|}) / |G|$$

常见置换的循环数
旋转：n 个点顺时针(逆时针)旋转 i 个位置的置换，循环数为 gcd(n, i)
翻转：
  n 为偶数时：
    对称轴不过顶点，循环数为 n/2
    对称轴过顶点：循环数为 n/2+1
  n 为奇数时：循环 数为(n+1)/2

立方体面用 k 种颜色涂色

$$\frac{8\mathrm{k}^2 + 12\mathrm{k}^3 + 3k^4 + k^6}{24}$$

例题

给定 n, m (n >= m)，从正 n 边形顶点中选出 m 个，能组成多少个不同的 m 边形?(旋转、翻转同构的算同一个)
解：把每种旋转、翻转视作一种置换，看做若干个互不相交的循环，他的 V 值设为置换中

取出 m 个元素并使每个循环中的元素或同被取出，或同未被取出的方案数。最终答案为所有置换 V 值得平均数。

在顺时针旋转 a 个的置换中，循环个数为(n, a)，每个循环长度为$\frac{n}{(n,a)}$，

$$\sum V = \sum_{d|(\mathrm{m},\mathrm{n})} C_{\frac{n}{d}}^{\frac{m}{d}} \varphi(d)$$

对于翻转(共 n 个置换)

	置换个数	循环个数	V
若 n 为奇数	n 个	$\left\lfloor\frac{n}{2}\right\rfloor$个循环长为 *2*，1 个循环长为 *1*	$C_{\left\lfloor\frac{n}{2}\right\rfloor}^{\left\lfloor\frac{m}{2}\right\rfloor}$
若 n 为偶数	$\frac{n}{2}$个	$\frac{n}{2}$个循环长为 *2*	$\begin{cases} C_{\frac{n}{2}}^{\frac{m}{2}} & m \text{ even} \\ 0 & m \text{ odd} \end{cases}$
	$\frac{n}{2}$个	$\frac{n}{2}-1$个循环长为 *2*， 2 个循环长为 *1*	$\begin{cases} C_{\frac{n}{2}-1}^{\frac{m}{2}} + C_{\frac{n}{2}-1}^{\frac{m}{2}-1} = C_{\frac{n}{2}}^{\frac{m}{2}} & m \text{ even} \\ 2C_{\frac{n}{2}-1}^{\frac{m}{2}-1} & m \text{ odd} \end{cases}$

杨氏图表(Young Tableau)

Young Tableau 是一个矩阵，他满足条件：

1. a[i, j] = null -> a[i+1, j] = null, a[i, j+1] = null
2. a[i, j] != null -> (a[i+1, j] = null or a[i+1, j] > a[i, j]), (a[i, j+1] = null or a[i, j+1] > a[i, j])

Y[n]: 1..n 这 n 个数组成的杨氏图表的个数。

$$\mathrm{Y}_n = \begin{cases} 1 & n = 1 \\ 2 & n = 2 \\ \mathrm{Y}_{n-1} + (n-1)\mathrm{Y}_{n-2} & n > 2 \end{cases}$$

e.g. n=3 时

1	2	3

2	
1	3

3	
1	2

3
2
1

格点几何

1. **Pick Theorem(Pick 定理)**：Let P be a simple polygon on the plane with vertices on lattices points, then its area is $A = I + B/2 - 1$, where A is area, B is the numbers of lattices on the edge. I is the number of interior lattice points of polygon. B = $\sum_{e \in E} gcd(e_x, e_y)$, where e is the vector of an edge.
2. 线段(0, 0) – (a, b)所经过的整点数 gcd(a, b) + 1,
3. 线段(0, 0) – (a, b)所覆盖的整数格子数 a + b – gcd(a, b)
4. w*h 的点阵有多少个锐角三角形

LL f(int w, int h) {

```
    LL ans = 0; int left = 1, right = h - 1;
    for (int i = 1; i <= w; ++i) {
        while ( left <= right && w * i > (h - left) * left ) ++left;
        while ( left <= right && w * i > (h - right) * right ) --right;
        int low = (w - i) * i / h + 1;
        if ( low >= right + 1 ) {
            ans += max(h-1 - low + 1, 0);
        } else {
            ans += max(h-1 - (right+1) + 1, 0);
            if ( low < left ) ans += max(left-1 - low + 1, 0);
        }
    }
    return ans;
}

LL calc(int w, int h) { return f(w, h) + f(h, w) << 1; }

LL gao(int w, int h) {
    LL ans = 0;
    for (int i = 1; i <= w; ++i) for (int j = 1; j <= h; ++j)
        ans += calc(i, j) * (w - i + 1) * (h - j + 1);
    return ans;
}

int main(){ for (int w, h; cin >> w >> h; cout << gao(w, h) << endl ); }
```

算两次技巧

问：空间 n 个点，任意三点不共线，没两点都用或红或黑的边连接，问纯色三角形有多少个？

答：总三角形个数C_n^3；非纯色三角形$\frac{1}{2}\sum_{i=1}^{n} R_i(n-1-R_i)$，Ri 为从 i 点出发的红边的条数。

number of solutions to x_1 ^ x_2 ^ ... ^ x_n = y, 0 <= x_j <= m_j

```
public class Main { // poj3968
    public static BigInteger solve(BigInteger []m, int n, BigInteger y, BigInteger any/* = 0*/) {
        // number of solutions to equation x_1 ^ x_2 ^ ... ^ x_n = y, 0 <= x_j <= m_j (index start at 1)
        // any[i] = 1 indicates that y[i] can be either 1 or 0
        int bitLength = y.bitLength(); // Notice: don't use BitCount
        for (int i = 1; i <= n; ++i) {
            if ( m[i].compareTo(BigInteger.ZERO) < 0 ) return BigInteger.ZERO;
            bitLength = Math.max(bitLength, m[i].bitLength() );
        }
```

```
        BigInteger []mask = new BigInteger[bitLength+1], c = new BigInteger[bitLength+1];

        BigInteger [][]f = new BigInteger[n+1][2];
        mask[0] = BigInteger.ZERO; c[0] = BigInteger.ONE;
        for (int i = 1; i <= bitLength; ++i) {
            mask[i] = BigInteger.ONE.shiftLeft(i).subtract(BigInteger.ONE);

            int parity = y.testBit(i-1) ? 1 : 0;
            for (int j = 1; j <= n; ++j) parity ^= m[j].testBit(i-1) ? 1 : 0;
            c[i] = any.testBit(i-1) || parity == 0 ? c[i-1] : BigInteger.ZERO;

            f[0][0] = BigInteger.ONE; f[0][1] = BigInteger.ZERO;
            for (int j = 1; j <= n; ++j) {
                for (int k = 0; k < 2; ++k) {
                    if ( m[j].testBit(i-1) ) {
f[j][k] = f[j-1][k^1].multiply( m[j].and(mask[i-1]).add(BigInteger.ONE) )
    .add( f[j-1][k].multiply( BigInteger.ONE.shiftLeft(i-1) ) ).mod(module);
                    } else {
f[j][k] = f[j-1][k].multiply( m[j].and(mask[i-1]).add(BigInteger.ONE) ).mod(module);
                    }
                }
            }

            BigInteger g = BigInteger.ONE;
            parity = y.testBit(i-1) ? 1 : 0;
            for (int j = n; j >= 1; --j) {
                if ( m[j].testBit(i-1) ) {
                    c[i] = c[i].add( f[j-1][parity].multiply(g) ).mod(module);
                    if ( any.testBit(i-1) )
                        c[i] = c[i].add( f[j-1][parity^1].multiply(g) ).mod(module);
                }
                g = g.multiply( m[j].and(mask[i-1]).add(BigInteger.ONE) ).mod(module);
                parity ^= m[j].testBit(i-1) ? 1 : 0;
            }
        }
        return c[bitLength];
    }
    final static Scanner cin = new Scanner(new BufferedInputStream(System.in));
    final static BigInteger module = BigInteger.valueOf(1000000003);
}
```

Lindstrom-Gessel-Viennot Lemma(格点多路计数)

设有带权向无环图 G = (V, E)，权值函数为 w : E->R，设 A = (a_1, a_2, …, a_n), B = (b_1, b_2, …, b_n)，其

中$a_i, b_j \in V$（任何顶点可以再任意位置出现任意次），对于路$P = (v_1, e_{v_1 v_2}, v_2, \dots, e_{v_{k-1} v_k}, v_k)$，设$w(P) = \prod_{e \in P} w_e$，$w(a, b) = \sum_{P: a \to b} w(P)$，

$$M = \begin{pmatrix} w(a_1, b_1) & \cdots & w(a_1, b_n) \\ \vdots & \ddots & \vdots \\ w(a_n, b_1) & \cdots & w(a_n, b_n) \end{pmatrix}$$

则 det(M)表示 A->B 所有的不相交 n 条路 $P = (P_1, \dots, P_n)$的权值之有向和，即

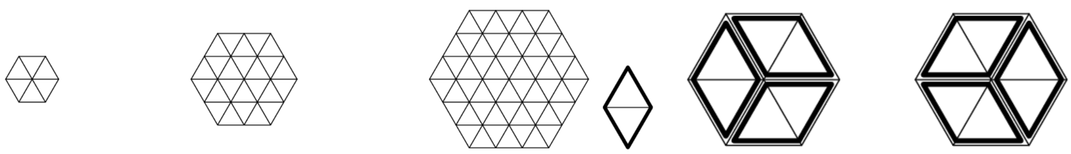
$$\det(M) = \sum_{(P_1, \dots, P_n): A \to B} \operatorname{sign}(\sigma(P)) \prod_{i=1}^{n} w(P_i)$$

事实上，我们可以设 w(i, j)为 i->j 的路径条数，每组 A->B 的 n 条不相交路都是 ai->bi，则 det(M)为 A->B 的不相交的 n 条路的数目。

例题 zoj4759：求(0, 0)->(m, n), (p, 0)->(m, q)的不相交的两条路的这样的 pair 的数目(p<m, q<n)。

$$ans = \det\begin{pmatrix} \binom{m+n}{m} & \binom{m+q}{m} \\ \binom{m-p+n}{m-p} & \binom{m-p+q}{m-p} \end{pmatrix}$$

用 指定平行四边形 覆盖 变长为 n 的蜂窝的方法数



1, 2, 20, 980, 232848, 267227532, 1478619421136, 39405996318420160, 5055160684040254910720, 3120344782196754906063540800, 926503771818193701224172728445000, 1323074488954060867061079598997993343750000

其他

n 个元素的集合的循环 r-排列的个数是：A_n^r / r

特别地，n 个元素的循环排列的个数是(n-1)!

有 m 个点排成一个环，选出其中的 k 个点，使得选出的任意两点不相邻，求方法数

$$\frac{m}{m-k}\binom{m-k}{k}$$

定理：{1, 2, …, n}的 r-组合 $a_1a_2\dots a_r$($a_1 \le a_2 \le \dots \le a_r$)的出现在所有该集合的 r-组合中的字典序号（从 1 开始编号）是：

$$C_n^r - C_{n-a_1}^r - C_{n-a_2}^{r-1} - \cdots - C_{n-a_{r-1}}^2 - C_{n-a_r}^1$$

蘑菇分裂

小 m 在宇宙中发现了一种奇怪的蘑菇，它每天都会固定分裂一次，长度为 x 的蘑菇会分裂成两个长度分别为 x - 1 和 x+1 的蘑菇，但是长度为 0 的蘑菇是不存在的，所以长度为 1 的蘑菇只能生长成一个长度为 2 的蘑菇。现在小 m 第一天有一个长度为 2 的蘑菇，他想知道第 n 天他有多少个蘑菇。
答案：C(n, n / 2)

数论

gcd & lcm

分数的最大公约数和最小公倍数

gcd(ai / bi) = gcd(ai) / lcm(bi)
lcm(ai / bi) = lcm(ai) / gcd(bi)
注意：**ai / bi 一定是最简分数**

其他

设$a > b, (a, b) = 1$，则$(a^m - b^m, a^n - b^n) = a^{(m,n)} - b^{(m,n)}$
推论：设$a > b$,则$(a^m - 1, a^n - 1) = a^{(m,n)} - 1$
推论：$(2^m - 1, 2^n - 1) = 2^{(m,n)} - 1$ i.e. $(m, n) = 1 \leftrightarrow (2^m - 1, 2^n - 1) = 1$

欧几里德算法

最坏情况：Fibnacci 数列相邻两项

```
T gcd(T a, T b){ //gcc 函数库 stl_algo.h 中有函数 T __gcd(T m, T n)
	while (b != 0){ T t = b; b = a % b; a = t; } return a;
}

T gcdfast(T a, T b){ //stein 算法，复杂度 log(n), 待测试
	T t = 0;
	while ((a & 1) == 0 && (b & 1) == 0)
		a >>= 1, b >>= 1, ++t;
	if (a < b) swap(a, b);
	while (b){
		while ((a & 1) == 0) a >>= 1;
		while ((b & 1) == 0) b >>= 1;
		a > b ? a -= b : b -= a;
		if (a < b) swap(a, b);
	}
```

```
    return a << t;
}
```

扩展欧几里德

解方程 ax + by = d

设 $ax_0 + by_0 = (a, b) = g$ (由扩展欧几里德算法)

若 d % g != 0 无解，否则

x = x0 * (d / g) – (b / g) * t

y = y0 * (d / g) + (a / g) * t

扩展欧几里德-递归

```
//求解  ax + by = gcd(a,b)
T exgcd(T a, T b, T &x, T &y){
    if (b == 0){ x = 1; y = 0; return a; }
    T d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

扩展欧几里德-迭代

```
LL exgcd(LL a, LL b, LL &x, LL &y){
    LL nx = 0, ny = 1;
    x = 1; y = 0;
    while (b != 0){
        LL q = a / b, t;
        t = a % b; a = b; b = t;             //{a, b} = {b, a % b}
        t = x - q * nx; x = nx; nx = t;      //{nextx, x} = {x - q * nextx, nextx}
        t = y - q * ny; y = ny; ny = t;      //{nexty, y} = {y - q * nexty, nexty}
    }
    return a;
}
```

取模

```
inline T mod(T x, T y){
    #define abs(x) ((x) > 0 ? (x) : -(x))     /*否则可能编译器不支持 LL 的绝对值*/
    y = abs(y);
```

```
    return x >= 0 ? x % y : x % y + y;
}
```

乘法

```
inline int mul_mod(int a, int b, int n){ //已经通过 bnu4362
    return (long long)a * b % n;
    int ret;
    __asm__ __volatile__ (
        "\tmull %%ebx\n"
        "\tdivl %%ecx\n"
        : "=d"(ret) : "a"(a), "b"(b), "c"(n)
    );
    return ret;
}

//已经通过 vijos1037
//计算两个 64 位整数的积对 n 的模，时间复杂度 O(100)，大概是直接乘积取模时间的 6、7 倍，效果比传说中的 Head 算法好
LL mul_mod(LL a, LL b, LL n) {
    LL r = 0, tmp = a % n;
    while (b) {
        if ( (b & 1) && (r += tmp) >= n) r -= n;
        if ( (b >>= 1) && (tmp <<= 1) >= n) tmp -= n;
    }
    return ret;
}
```

乘法逆元

```
inline int inv_mod(int a, int p){
    int x, y;
    assert ( exgcd(a, p, x, y) == 1 );
    return mod(x, p);
}
```

设$p = at + k$，则$at = -k \pmod p \Rightarrow a^{-1} = -k^{-1}t \pmod p$——可以 O(n) dp 求 1..n 的逆

```
inv[1] = 1;
for (int i = 2; i <= n; ++i) {
    int t = module / i, k = module % i;
    inv[i] = (module - inv[k]) * t % module;
}
```

除法

```
inline int div_mod(int a, int b, int n){
	return mul_mod(a, inv_mod(b, n), n);
}
```

指数

```
T pow_mod(T a, T b, T n){ //要求 a >= 0，已经通过 hdu2837
	T r = 1; a = a % n;
	while(b){
		if (b & 1) r = mul_mod(r, a, n);
		if (b >>= 1) a = mul_mod(a, a, n);
	}
	return ret;
}
```

离散对数(shank's Baby-Step-Giant-Step Algorithm)

```
const int maxn = 1000000;
LL mexp[maxn]; int id[maxn];

bool logcmp(const int& a, const int& b){
	return mexp[a] < mexp[b];
}

int log_mod(int a, int b, int n){
	// a ^ x = b mod n
	//时间复杂度 sqrt(n) * logn，空间复杂度 sqrt(n)
	//已经通过 poj3358, poj2417, hdu3930
	int m = (int)ceil(sqrt(n));
	int v = inv_mod(pow_mod(a, m, n), n);
	id[0] = 0; mexp[0] = 1;
	for (int i = 1; i <= m; i++){
		id[i] = i;
		mexp[i] = mul_mod(mexp[i-1], a, n);
	}
	stable_sort(id + 1, id + m + 1, logcmp);
	sort(mexp + 1, mexp + m + 1);
	for (int i = 0, j; i < m; i++){
		j = lower_bound(mexp + 1, mexp + m + 1, b) - mexp;
```

```
        if (j <= m && mexp[j] == b) return i * m + id[j];
        b = mul_mod(b, v, n);
    }
    return -1;
}
```

广义离散对数

```
struct Zn {
    const LL n;
    Zn(LL n) : n(n){
    }
    LL eval(LL a) {
        a %= n; return a >= 0 ? a : a + n;
    }
    LL mul(LL a, LL b) {
        if ( n <= 1000000000 ) return a * b % n;
        assert(0);
    }
    LL pow(LL a, LL b) {
        LL r = 1 % n;
        for ( ;b; ) {
            if ( b & 1 ) r = mul(r, a);
            if ( b >>= 1 ) a = mul(a, a);
        }
        return r;
    }
    LL inv(LL a) {
        LL x, y; assert( exgcd(a, n, x, y) == 1 );
        return eval(x);
    }
    LL log(LL a, LL b);
};

struct Zp : Zn {
    Zp(LL n) : Zn(n) {
    }
    const static int maxsqrtn = 100000 + 5;
    static int id[]; static LL mexp[];

    struct logcmp {
        bool operator()(int a, int b) { return mexp[a] < mexp[b]; }
    };
```

```
	LL log(LL a, LL b) { // a ^ x = b
		int m = (int)( ceil( sqrt(n) ) );
		LL v = inv( pow(a, m)    );
		id[0] = 0; mexp[0] = 1;
		for (int i = 1; i <= m; ++i) {
			id[i] = i; mexp[i] = mul( mexp[i-1], a );
		}
		stable_sort( id + 1, id + m + 1, logcmp() );
		sort(mexp + 1, mexp + m + 1);
		for ( int i = 0, j; i < m; ++i ) {
			j = lower_bound( mexp, mexp + m + 1, b ) - mexp;
			if ( j <= m && mexp[j] == b ) return i * m + id[j];
			b = mul(b, v);
		}
		return -1;
	}
};

LL Zn::log(LL a, LL b) { // a ^ x = b
	for (int i = 0; i <= 50; ++i) if ( pow(a, i) == b ) return i;
	LL g, d = 1, n = this->n, x = 0;
	while ( (g = gcd(a, n)) > 1 ) {
		if ( b % g ) return -1;
		b /= g; n /= g; d    = mul(d, a / g); ++x;
	}
	Zp zp(n); LL ans = zp.log( a, zp.mul(b, zp.inv(d) ) );
	return ans == -1 ? -1 : ans + x;
}

int Zp::id[ Zp::maxsqrtn ]; LL Zp::mexp[ Zp::maxsqrtn ];

int main(){
	for (LL k, p, n; cin >> k >> p >> n; ) {
		if ( n >= p ) { puts("Orz,I can't find D!"); continue; }
		k %= p; n %= p; Zn zn(p); LL ans = zn.log( k, n );
	}
}
```

开方(Tonelli-Shanks algorithm)

```
int sqrt_mod(int a, int n){ //时间复杂度  blogn + logn * logn， E(b) = O(1)
	if (n == 2) return a % n;
	if (pow_mod(a, (n-1) / 2, n) == 1){
		int x;
```

```
		if (n % 4 == 3) {
			x = pow_mod(a, (n+1)/4, n);
		} else {
			int b = 1;
			while (pow_mod(b, (n-1)/2, n) == 1) b++;
			int i = (n-1)/2, k = 0;
			do {
				i /= 2; k /= 2;
				if (mul_mod(pow_mod(a, i, n), pow_mod(b, k, n), n) == n - 1){
					k += (n-1) / 2;
				}
			} while (i % 2 == 0);
			x = mul_mod(pow_mod(a, (i+1)/2, n), pow_mod(b, k / 2, n), n);
		}
		if (x * 2 > n) x = n - x;
		return x;
	}
	return -1;
}
```

大数因式分解(rho)

最坏$O(n^{1/2})$，平均$O(n^{1/4})$，详见 http://wwwmaths.anu.edu.au/~brent/pd/rpb051i.pdf

//已经通过好多题，确保正确，java版本的rho详见第83页，上次使用2012.9.17

```
LL rho(LL n){
	LL x, y, d, c; int k, i;
	for (;;){
		c = rand() % (n - 1) + 1;
		x = y = rand() % n;
		k = 2; i = 1;
		do {
			d = gcd( ABS(x - y), n );
			if (d > 1 && d < n) return d;
			if (++i == k) y = x, k *= 2;
			x = mul_mod(x, x, n); x = (x + c) % n;
		} while (x != y);
	}
}
```

FFT

```
struct Zp {
	const LL mod; const int pri;
```

```
Zp(LL module, int primitive) : mod(module), pri(primitive){
}
Zp(LL module) : mod(module), pri( primitive() ) {
}
LL add(LL a, LL b) {
	a += b; return a >= mod ? a - mod : a;
}
LL sub(LL a, LL b) {
	a -= b; return a < 0 ? a + mod : a;
}
LL mul(LL a, LL b) {
	if ( mod <= 1000000000 ) return a * b % mod;
	LL t = (LL)( (double)a * (double)b / mod + 0.5 );
	LL r = (a * b - t * mod) % mod;
	return r >= 0 ? r : r + mod;
}
LL pow(LL a, LL b) {
	LL r = 1;
	for (;b;) {
		if ( b & 1 ) r = mul(r, a);
		if ( b >>= 1 ) a = mul(a, a);
	}
	return r;
}
LL inv(LL a) {
	return pow(a, mod - 2);
}
void fft(int n, LL root, LL a[]) {
	for (int m = n; m >= 2; m >>= 1) {
		int mh = m >> 1; LL w = 1;
		for (int i = 0; i < mh; ++i) {
			for (int j = i; j < n; j += m) {
				int k = j + mh;
				LL t = sub(a[j], a[k]);
				a[j] = add(a[j], a[k]);
				a[k] = mul(w, t);
			}
			w = mul(w, root);
		}
		root = mul(root, root);
	}
	for (int j = 1, i = 0; j < n - 1; ++j) {
		for (int k = n >> 1; k > (i ^= k); k >>= 1);
		if (j < i) swap(a[i], a[j]);
```

```
		}
	}
	void dft(const LL a[], int an, LL b[], int n) {
		LL root = pow( pri, mod / n);
		copy(a, a + an, b); fill( b + an, b + n, 0 );
		fft(n, root, b);
	}
	void nft(const LL a[], LL b[], int n) {
		LL root =inv( pow( pri, mod / n) );
		copy(a, a + n, b);
		fft(n, root, b );
		LL invn = inv(n);
		rep(i, n) b[i] = mul(b[i], invn);
	}
	int primitive() {
		LL n = mod - 1;
		LL p[25], pcnt = 0;
		for (LL i = 2; i * i <= n; ++i) {
			if ( n % i == 0 ) {
				do n /= i; while (n % i == 0);
				p[pcnt++] = i;
			}
		}
		if (n > 1) p[pcnt++] = n;
		for (int g = 2; ;++g) {
			int ok = 1; //assert( pow(g, mod-1) == 1 );
			rep(i, pcnt) if ( pow(g, (mod-1)/p[i] ) == 1 ) {
				ok = 0; break;
			}
			if (ok) return g;
		}
	}
} zp(500000000001507329LL, 3);

struct Poly { // Polynomial functions
	const static int maxn = ::maxn * 4 + 5;
	LL a[maxn]; int n;
	void init(const LL a[], int n) {
		this->n = n; copy(a, a + n, this->a);
	}
	void init(const int a[], int n) {
		this->n = n; copy(a, a + n, this->a);
	}
	LL eval(LL x) const {
```

```
        LL ans = 0;
        for (int i = n - 1; i >= 0; --i)
            ans = zp.add( zp.mul(ans, x), a[i] );
        return ans;
    }
    friend void multiply(const Poly& x, const Poly& y, Poly& r) {
        static LL xb[maxn], yb[maxn];
        int n = 1; while ( n < x.n + y.n ) n *= 2;
        LL root = zp.pow( zp.pri, zp.mod / n);
        zp.dft(x.a, x.n, xb, n); rep(i, n) assert( x.eval( zp.pow( root, i ) ) == xb[i] );
        zp.dft(y.a, y.n, yb, n); rep(i, n) assert( y.eval( zp.pow( root, i ) ) == yb[i] );
        rep(i, n) xb[i] = zp.mul(xb[i], yb[i]);
        zp.nft(xb, r.a, n);
        r.n = n;
        while (r.n > 0 && r.a[r.n-1] == 0) --r.n;
    }
};

struct BigInteger { //unsigned
    static const int digit = 4;
    static const int base = 10000;
    static const int cap = 50000 * 2 + 5; // 10 ^ 500
    static const int maxn = cap / digit + 1;
    int dat[maxn], n;

    BigInteger(const BigInteger& o) : n(o.n) {
        copy(o.dat, o.dat + n, dat);
    }

    BigInteger(LL v = 0) {
        for (n = 0; v; v /= base) dat[n++] = v % base;
    }

    void parse(char *s) {
        n = 0;
        int m = 1;
        for (int i = strlen(s) - 1, v = 0; i >= 0; --i) {
            v = v + (s[i] - '0') * m; m *= 10;
            if (m == base || i == 0) {
                dat[n++] = v; v = 0; m = 1;
            }
        }
    }
```

```
char *toString(char *s) const {
    if (n == 0) {
        strcpy(s, "0");
    } else {
        char *p = s;
        p += sprintf(p, "%d", dat[n-1]);
        for (int i = n - 2; i >= 0; --i)
            p += sprintf(p, "%0*d", digit, dat[i]);
    }
    return s;
}

char *toString() const {
    static char buf[cap+5];
    return toString(buf);
}

friend void add(const BigInteger& x, const BigInteger& y, BigInteger& r) {
    int i = 0;
    for (int t = 0; i < x.n || i < y.n || t; ++i, t /= base) {
        if (i < x.n) t += x.dat[i];
        if (i < y.n) t += y.dat[i];
        r.dat[i] = t % base;
    }
    r.n = i;
}

friend void sub(const BigInteger& x, const BigInteger& y, BigInteger& r) {
    r.n = x.n;
    for (int i = 0, t = 0; i < r.n; ++i) {
        r.dat[i] = x.dat[i] - t;
        if ( i < y.n ) r.dat[i] -= y.dat[i];
        if ( r.dat[i] < 0 ) {
            t = 1; r.dat[i] += base;
        } else {
            t = 0;
        }
    }
    while ( r.n    && r.dat[r.n - 1] == 0 ) --r.n;
}

friend void mul(const BigInteger& x, int y, BigInteger& r) {
    int i = 0;
    for (LL t = 0; i < x.n || t; ++i, t /= base) {
```

```
            if (i < x.n) t += (LL)(x.dat[i]) * y;
            r.dat[i] = t % base;
        }
        r.n = i;
    }

    friend void mul(const BigInteger& x, const BigInteger& y, BigInteger& r) {
        r.n = 0;
        for (int i = 0; i < x.n; ++i) {
            for (int j = 0, t = 0; j < y.n || t; ++j, t /= base) {
                if ( j < y.n ) t += (LL)x.dat[i] * y.dat[j];
                if ( i + j < r.n ) t += r.dat[i+j];
                r.dat[ i+j ] = t % base;
                if ( i + j == r.n ) ++r.n;
            }
        }
    }

    friend void mulfft(const BigInteger& x, const BigInteger& y, BigInteger& r) { //已经经过测试
        static Poly px, py, pr;
        px.init(x.dat, x.n);
        py.init(y.dat, y.n);
        multiply(px, py, pr);
        int i = 0;
        for (LL t = 0; i < pr.n || t; ++i, t /= base) {
            if (i < pr.n) t += pr.a[i];
            r.dat[i] = t % base;
        }
        r.n = i;
    }

    friend void div(const BigInteger& x, int y, BigInteger& q, int &r) {
        q.n = x.n; r = 0;
        for (int i = x.n - 1; i >= 0; --i, r %= y) {
            r = r * base + x.dat[i];
            q.dat[i] = r / y;
        }
        while (q.n && q.dat[ q.n-1 ] == 0) --q.n;
    }
};

BigInteger x, y, z;
char buf[1000000];
```

```
int main(){
	while ( gets(buf) ) {
		x.parse(buf);
		gets(buf); y.parse(buf);
		mulfft(x, y, z);
		puts( z.toString() );
	}
}
```

Binomial 求 C(n, i) mod P = 0…P-1 的 i 的个数%29

```
const int P = 51061;
Zp zp(50000000001507329LL, 3), zq(P, 2);

struct Poly {
	static const int maxn = P * 4;
	friend void multiply(const Poly& x, const Poly& y, Poly& r) {
		//blablabla
		r.n = n;
		while ( r.n-1 >= P - 1) {
			--r.n; r.a[ r.n - (P-1) ] += r.a[ r.n ];
		}
		for (int i = 0; i < r.n; ++i) r.a[ i ] %= 29;
	}
} poly[11];

LL powG[P], indFact[P]; int ind[P];

void init(){
	powG[0] = 1; ind[1] = 0;
	for (int i = 1; i < P - 1; ++i) ind[ powG[i] = zq.mul( powG[i-1], zq.pri ) ] = i;
	indFact[0] = indFact[1] = ind[1];
	for (int i = 2; i < P; ++i) indFact[i] = indFact[i-1] + ind[i];
}

void solve(const char s[]) {
	LL n[15] = {0};
	int tot = 0, up = 0;
	for (const char *ptr = s; *ptr; ++ptr) {
		int v = *ptr - '0';
		tot = ( tot * 10 + v ) % 29;
		for (int i = 0; i <= up; ++i) n[i] *= 10;
		n[0] += v;
		for (int i = 0; i <= up; ++i) {
```

```
                if ( n[i] >= P ) {
                    n[i+1] += n[i] / P; n[i] %= P; up = max(up, i + 1);
                }
            }
        }
        ++tot;
        for (int i = 0; i <= up; ++i) {
            static LL a[P];
            memset(a, 0, sizeof a);
            for (int j = 0; j <= n[i]; ++j) {
                LL ind = indFact[ n[i] ] - indFact[ j ] - indFact[ n[i]-j ];
                ind %= P - 1;
                if ( ind < 0 ) ind += P - 1; // of importance
                ++a[ind];
                if ( a[ind] == 29 ) a[ind] = 0;
            }
            poly[i].init(a, P - 1);
        }
        for (int i = 1; i <= up; ++i)
            multiply( poly[0], poly[i], poly[0] );
        LL ans[P], totnonezero = 0;
        for (int i = 0; i < P - 1; ++i) {
            ans[ powG[i] ] = poly[0].a[i] % 29;
            totnonezero +=   ans[ powG[i] ];
        }
        ans[0] = ( (tot - totnonezero) % 29 + 29 ) % 29;
        for (int i = 0; i < P; ++i)
            putchar( ans[i]["0123456789ABCDEFGHIJKLMNOPQRS"] );
        puts("");
}

int main(){
        char n[55]; LL p;
        for (init(); cin >> n >> p; solve(n)) assert(p == P);
}
```

计算 f[i] = a ^ f[i-1] mod n (super_pow_mod)

```
int mypow(long long a, int n, int m){
        if (n == -1) return -1; if (n == 0) return 1;
        if (a == 0) return 0; if (a == 1) return 1;
        long long r = 1;
        while (n--)
            if ((r *= a) >= m) return -1;
```

```
    return r;
}

int _super_pow_mod(int a, int n, int dep, int r[]){
    if (r[dep] != -1 || n == 1)
        return r[dep] % n;
    int phin = phi(n);
    if (r[dep-1] == -1){
        int e = _super_pow_mod(a, phin, dep - 1, r);
        return pow_mod(a, e + phin, n);
    } else {
        return pow_mod(a, r[dep-1], n);
    }
}

int super_pow_mod(int a, int n, int dep){
    static int r[maxdep];
    r[0] = 1;
    for (int i = 1; i <= dep; i++)
        r[i] = mypow(a, r[i-1], n);
    return _super_pow_mod(a, n, dep, r);
}
```

模线性方程

```
/*
     ax == b mod n
若(a, n) | b => Zn 有(a, n)个根，否则无根
<->    ax - ny = b
let    d = gcd(a, n)
if     b mod d != 0 no solution
else solve ax0 - by0 = d
     xi = ((x0 * b/d) + i * (n/d)) mod n
     min(xi) = (x0 * b/d) mod n/d
*/
int mels(int a, int b, int n, int sol[]){
    int d, x0, y0;
    d = exgcd(a, b, x0, y0);
    if (c % d != 0){ // no solution，若方程是 a mod n = b，则 c >= b 也无解
        return 0;
    }
    int e = mod(x0 * c / d, n), f = n / d;
    for (int i = 0; i < d; i++){
        //sol[i] = mod((x0 * b/d) + i * (n/d), n);
```

```
            sol[i] = e;
            if ((e +=f) >= n) e -= n;
        }
        /*minsol = mod(x0 * b / d, n / d); */
        return d;
}
```

广义中国剩余定理

```
//China Remainder Theorem – 已经通过 pku2891, hdu3579
bool CRT2(T d1, T r1, T d2, T r2, T &dd, T &rr){
        T q1, q2, g = exgcd(d1, d2, q1, q2);
        T c = r1 - r2; if (c < 0) c += d1;
        dd = d1 / g * d2;
        if (c % g != 0) { rr = -1; return false; }
        T t = d1 / g;
        q2 *= c / g; q2 = q2 % t; if (q2 <= 0) q2 += t;
        rr = q2 * d2 + r2; if (rr >= dd) rr -= dd;
        return true;
}

bool CRT(T d[], T r[], int n, T &dd, T &rr){ //d[]除数, r[]余数 0..n-1
        dd = 1, rr = 0;
        rep(i, n) if (!CRT2(dd, rr, d[i], r[i], dd, rr)) return false;
        return true;
}

int main(){
        const int maxn = 100000;
        static T d[maxn], r[maxn];
        int k;
        while (scanf("%d", &k) != EOF){
                for (int i = 0; i < k; i++)
                        scanf("%lld%lld", &d[i], &r[i]);
                T dd, rr; CRT(d, r, k, dd, rr);
                printf("%lld\n", rr);
        }
}
```

Pell 方程

```
public class Main {
        //In:   a positive integer N, where N is not a perfect square number
```

```
//Out: the minimum non-trivial solution of equation x * x − n * y * y = 1.
//Pell方程通解(可以用矩阵快速幂快速求出第n大的解)
// x[n] = x[n-1] * x[1] + d * y[n-1] * y[1]
// y[n] = x[n-1] * y[1] + y[n-1] * x[1]
static BigInteger[] Pell(BigInteger n) {
	BigInteger[] p = new BigInteger[10], q = new BigInteger[10];
	BigInteger[] g = new BigInteger[10], h = new BigInteger[10];
	BigInteger[] a = new BigInteger[10]; BigInteger a0;
	p[-1 & 3] = BigInteger.ONE; p[-2 & 3] = BigInteger.ZERO;
	q[-1 & 3] = BigInteger.ZERO; q[-2 & 3] = BigInteger.ONE;
	a0 = a[0] = Sqrt(n);
	g[-1 & 3] = BigInteger.ZERO; h[-1 & 3] = BigInteger.ONE;
	for (int i = 0;; i++) {
		g[i & 3] = a[i & 3].multiply(h[i - 1 & 3]).subtract(g[i - 1 & 3]);
		h[i & 3] = n.subtract(g[i & 3].multiply(g[i & 3]))
				.divide(h[i - 1 & 3]);
		a[i + 1 & 3] = g[i & 3].add(a0).divide(h[i & 3]);
		p[i & 3] = a[i & 3].multiply(p[i - 1 & 3]).add(p[i - 2 & 3]);
		q[i & 3] = a[i & 3].multiply(q[i - 1 & 3]).add(q[i - 2 & 3]);
		if (p[i & 3].multiply(p[i & 3])
				.subtract(n.multiply(q[i & 3]).multiply(q[i & 3]))
				.equals(BigInteger.ONE)) {
			return new BigInteger[] { p[i & 3], q[i & 3] };
		}
	}
}

static BigInteger Sqrt(BigInteger n) {
	BigInteger low = BigInteger.ZERO, high = n, ans = null;
	while (low.compareTo(high) <= 0) { // Notice: use <= rather than <
		BigInteger mid = low.add(high).shiftRight(1);
		if (mid.multiply(mid).compareTo(n) <= 0) {
			ans = mid; low = mid.add(BigInteger.ONE);
		} else {
			high = mid.subtract(BigInteger.ONE);
		}
	}
	return ans;
}

public static void main(String args[]) { //spoj Yet Another Equation
	Scanner cin = new Scanner(new BufferedInputStream(System.in));
	int t = cin.nextInt();
	while (t-- != 0) {
```

```
            BigInteger n = cin.nextBigInteger();
            BigInteger[] r = Pell(n);
            System.out.println(r[0] + " " + r[1]);
        }
    }
}
```

毕达哥拉斯三元组

定理：正整数 x,y,z 构成一个本原毕达哥拉斯三元组且 y 为偶数，当且仅当互素的正整数 m,n (m > n)，其中 m 和 n 奇偶性不同，且满足

$$\begin{cases} x = m^2 - n^2 \\ y = 2mn \\ z = m^2 + n^2 \end{cases}$$

素数

费马-欧拉素数定理：每个可表示为 4n+1 形式的素数，只能用一种两数的平方和的形式来表达。

2	3	5	7	11	13	17	19	23	29
31	37	41	43	53	59	61	67	71	73
79	81	89	97						

2 primes[0]

10007

1000003 primes[78498]

2000003 primes[148933]

10000019 = primes[664579]

素数的判断

$$\begin{aligned}
\psi_1 &= 2047 \\
\psi_2 &= 1\,373\,653 \\
\psi_3 &= 25\,326\,001 \\
\psi_4 &= 3\,215\,031\,751 \\
\psi_5 &= 2\,152\,302\,898\,747 \\
\psi_6 &= 3\,474\,749\,660\,383 \\
\psi_7 &= 341\,550\,071\,728\,321 \\
\psi_8 &= 341\,550\,071\,728\,321 \\
\psi_9 &= 3\,825\,123\,056\,546\,413\,051 \\
\psi_{10} &= 3\,825\,123\,056\,546\,413\,051
\end{aligned}$$

ψ_{11}	=	3 825 123 056 546 413 051
ψ_{12}	=	318 665 857 834 031 151 167 461
ψ_{13}	=	3 317 044 064 679 887 385 961 981
ψ_{14}	=	6 003 094 289 670 105 800 312 596 501
ψ_{15}	=	59 276 361 075 595 573 263 446 330 101
ψ_{16}	=	564 132 928 021 909 221 014 087 501 701
ψ_{17}	=	564 132 928 021 909 221 014 087 501 701
ψ_{18}	=	1 543 267 864 443 420 616 877 677 640 751 301
ψ_{19}	=	1 543 267 864 443 420 616 877 677 640 751 301
ψ_{20}	>	10^{36}.

n < 25,000,000,000 && n != 3,215,031,751(prime)	2, 3, 5, 7
n < 75,792,980,677	2, 379215, 457083754
n < 21,652,684,502,221	2, 1215, 34862, 574237825

```
bool witness (LL a, LL n){ //适用范围：奇数 n >= 5
	int k = 0;
	LL m = n - 1;
	do {m /= 2; k++; } while (m % 2 == 0);
	LL x = pow_mod(a, m, n);
	if (x == 1) return true;
	for (int i = 0; i < k; x = mul_mod(x, x, n), ++i){
		if (x == n - 1) return true;
	}
	return false;
}
```

//已经通过 4 题，上次使用 2012.9.17

```
bool miller_rabin(LL n, int time = 50){
	if (n == 2 || n == 3 || n ==5 || n == 7) return 1;
	if (n == 1 || n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n % 7 == 0) return 0;
	while (time--){
		LL r = rand() % (n-2) + 2;
		if ( gcd(r, n) != 1 || !witness (r % n, n) ) return 0;
	}
	return 1;
}
```

高效版

上次使用 2012.8

```
int strong_psudo_prime_test(LL n, int a){
	LL m = n - 1, res;
```

```
	int s = 0;
	while (m % 2 == 0) m >>= 1, s++;
	res = pow_mod(a, m, n);
	if (res == 1 || res == n-1) return 1;
	while (--s >= 0){
		res = mul_mod(res, res, n);
		if (res == n-1) return 1;
	}
	return 0;
}

int isprime(LL n){
	if (n < 2) return 0;
	if (n < 4) return 1;
	if (!strong_psudo_prime_test(n, 2)) return 0;
	if (!strong_psudo_prime_test(n, 3)) return 0;
	if (n < 1373653LL) return 1;
	if (!strong_psudo_prime_test(n, 5)) return 0;
	if (n < 25326001LL) return 1;
	if (!strong_psudo_prime_test(n, 7)) return 0;
	if (n == 3215031751LL) return 0;
	if (n < 25000000000LL) return 1;
	if (!strong_psudo_prime_test(n, 11)) return 0;
	if (n < 2152302898747LL) return 1;
	if (!strong_psudo_prime_test(n, 13)) return 0;
	if (n < 3474749660383LL) return 1;
	if (!strong_psudo_prime_test(n, 17)) return 0;
	if (n < 341550071728321LL) return 1;
	if (!strong_psudo_prime_test(n, 19)) return 0;
	if (!strong_psudo_prime_test(n, 23)) return 0;
	if (!strong_psudo_prime_test(n, 29)) return 0;
	if (!strong_psudo_prime_test(n, 31)) return 0;
	if (!strong_psudo_prime_test(n, 37)) return 0;
	return 1;
}
```

筛法求素数 O(n)

利用了每个合数必有一个最小素因子。每个合数仅被它的最小素因子筛去正好一次。所以为线性时间。

经检验，线性时间筛法求素数的的时间，至多是普通筛法的 1/2

```
const int maxn = 10000000;
int factorcnt[maxn]; //factorcnt = 2 indicates prime，已经通过 bnu1571
char minfactorcnt[maxn];
```

```
int primes[maxn], pcnt = 0;

void init(int n = maxn - 1){
	for (int i = 2; i <= n; i++){
		factorcnt[i] = 2;
	}
	for (int i = 2; i <= n; i++){
		if (factorcnt[i] == 2){
			primes[pcnt++] = i, minfactorcnt[i] = 1;
		}
		for (int j = 0, x; j < pcnt && (x = i * primes[j]) <= n; j++){
			if (i % primes[j] == 0){
				//primes[j] isn't the unique factor of x
				minfactorcnt[x] = minfactorcnt[i] + 1;
				factorcnt[x] = factorcnt[i] / minfactorcnt[x] * (minfactorcnt[x] + 1);
				break; //关键一句，防止重复标记
			}
			else {
				minfactorcnt[x] = 1;
				factorcnt[x] = factorcnt[i] * 2;
			}
		}
	}
}
```

反素数

反素数：d(n) > d(k)对于 n > k 恒成立的 n，其中 d(n): n 的约数个数

1..n 中最大的反素数 <-> 约数最多的数(如果有多个满足要求，得到最小的一个)

代码

```
//求解 <= x 的最大的反素数(约数最多的数——如果有多个满足要求，得到最小的一个)
int primes[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, …};
T x, res = 1, factcnt = 1;
void dfs(int dep, T product, int last, T fc){ /* x 很大(高精度)时, fc 可能需要用高精度*/
	//前 dep 个素数，积为 product，最大质因数次数 last，因子数为 fc
	if (product > x) return;
	if (fc > factcnt || fc == factcnt && product < res)
		factcnt = fc, res = product;
	for (int j = 1; j <= last && (product *= primes[dep]) <= x; j++)
		dfs(dep+1, product, j, fc * (j+1));
}

int main(){ //zju 2562
	while (cin >> x){
```

```
        res = factcnt = 1; //记得初始化！！
        dfs(0, 1, 9999999, 1);
        cout << res << endl;
    }
}
```

因式分解

```
//已经通过 PKU2480
void factorize(T n, T p[], int r[], int &c){
    //对 n 进行因式分解，要求质数表至少打到 floor(sqrt(n)), O(sqrt(n) / log(n) + log(n))
    c = 0;
    for (int i = 0; i < pcnt && primes[i] * primes[i] <= n; i++){
        if (n % primes[i] == 0){
            p[c] = primes[i]; r[c] = 1; n /= p[i];
            while (n % p[i] == 0) r[c]++, n /= p[i];
            c++;
        }
    }
    if (n > 1){
        p[c] = n; r[c] = 1; c++;
    }
}
```

原根

原根定理：n > 1 时，使得 Z_n*为循环群的 n 只有 2，4，p^e 或者 $2p^e$，其中 p 为任一奇素数，e 为任意正整数。

Gauss proved that for any prime number p (with the sole exception of p = 3), the product of its primitive roots is congruent to 1 modulo p.

He also proved that for any prime number p, the sum of its primitive roots is congruent to μ(p-1) mod p where μ is the Mobius function.

The number of primitive roots modulo n, if there are any, is equal to ϕ(ϕ(n)).
Zn 如果有原根，则个数为 ϕ(ϕ(n))

Order of magnitude of primitive roots:
The least primitive root modulo p is generally small.
Shoup(1990, 1992) proved, assuming the generalized Riemann hypothesis, the $g_p = O(\log^6 p)$

```
int primitive_root(LL n){ //已经通过 hdu3930
```

```
	//要求 n 满足原根定理
	LL phin = n - 1;//calc_phi(n);
	LL t = phin;
	LL p[20]; int pcnt = 0;
	for (LL i = 2; i * i <= t; i++){
		if (t % i == 0){
			p[pcnt++] = i; while (t % i == 0) t /= i;
		}
	}
	if (t > 1)
		p[pcnt++] = t;
	for (int r = 2; ; r++){
		if ( __gcd(n, (LL)r) != 1 ) continue;
		int flag = true;
		for (int k = 0; !flag && k < pcnt; k++)
			if ( pow_mod(r, phin / p[k], n ) == 1 )
				flag = false;
		if (flag) return r;
	}
}
```

积性函数

欧拉函数

预备知识

定理(消去律)：如果 gcd(c,p) ＝ 1，则 ac ≡ bc mod p 可以推出 a ≡ b mod p
证明：因为 ac ≡ bc mod p，所以 ac = bc + kp，也就是 c(a-b) = kp，因为 c 和 p 没有除 1 以外的公因子，因此上式要成立必须满足下面两个条件中的一个
1) c 能整除 kp
2) a = b
如果 a ≠ b 不成立，则 c|kp。因为 c 和 p 没有公因子，因此显然 c|k，所以 k = ck'，因此 c(a-b) = kp 可以表示为 c(a-b) =ck'p，因此 a-b = k'p，得出 a ≡ b mod p
如果 a = b，则 a ≡ b mod p 显然成立
得证

定义

欧拉函数是指：对于一个正整数 n，<=n 且和 n 互质的正整数(包括 1)的个数，记作 φ(n) 。(也有材料定义 φ(n)为模 n 的所有既约剩余类的个数)

定义小于 n 且和 n 互质的数构成的集合为 Z_n。 显然 $|Z_n| = \phi(n)$

性质

1. 对于 p^k，$\phi(n) = p^k - p^{k-1}$
2. 若 p、q 互质，p * q 的欧拉函数 $\phi(p * q) = \phi(p) * \phi(q)$
   证明：令 n = p * q, gcd(p,q) = 1，则由中国剩余定理，有 Zn 和 Zp×Zq 之间的一一映射(a ∈Zp, b∈Zq ⇔ (a * q + b * p) mod n∈Zn)。所以 $\phi(p * q) = |Zn| = |Zp \times Zq| = \phi(p) * \phi(q)$
3. 任意正整数的欧拉函数：任意一个整数 n 都可以表示为其质因数的乘积为：
   $$n = \prod_{i=1}^{n} p_i^{k_i}$$
   根据前面的结论很容易得到
   $$\varphi(n) = \prod_{i=1}^{n} p_i^{k_i-1}(p_i - 1) = n\prod_{i=1}^{n}\frac{p_i - 1}{p_i}$$
4. 欧拉定理：对于互质的正整数 a 和 n，有 $a^{\phi(n)} \equiv 1 \bmod n$
   证明：
   (1)引理：令 $Zn = \{x_1, x_2, ..., x_{\phi(n)}\}$，$S = \{a * x_1 \bmod n, a * x_2 \bmod n, ... , a * x_{\phi(n)} \bmod n\}$，则 Zn = S。
   ① 因为 a 与 n 互质，x_i $(1 \le i \le \phi(n))$ 与 n 互质，所以 $a * x_i$ 与 n 互质，所以 $a * x_i \bmod n \in Zn$
   ② 若 $i \ne j$，那么 $x_i \ne x_j$，且由 a, n 互质可得 $a * x_i \bmod n \ne a * x_j \bmod n$(消去律)。
   (2) $a^{\phi(n)} * x_1 * x_2 *... * x_{\phi(n)} \bmod n$
   $\equiv (a * x_1) * (a * x_2) * ... * (a * x_{\phi(n)}) \bmod n$
   $\equiv (a * x_1 \bmod n) * (a * x_2 \bmod n) * ... * (a * x_{\phi(n)} \bmod n) \bmod n$
   $\equiv x_1 * x_2 * ... * x_{\phi(n)} \bmod n$
   对比等式的左右两端，因为 x_i $(1 \le i \le \phi(n))$ 与 n 互质，所以 $a^{\phi(n)} \equiv 1 \bmod n$(消去律)。
   (3) 推论：$a^{-1} \equiv a^{\phi(n)-1} \bmod p$
5. 费马小定理：若正整数 a 与素数 p 互质，则有 $a^{p-1} \equiv 1 \bmod p$
6. **将 i =1..n 按照(i, n) >= d 分成不同的等价类，则等价类{x | (x, n) >= d} = {d, 2d, …, kd, …, (n / d)d | k 属于 $Z_{n/d}$}**
   设 G_k = { gcd(i,n) = k, 0 <= i < n }
   若 k|n，G_k = { gcd(i/k, n/k) = 1, 0 <= i/k < n/k } –> **$|G_k| = \Phi(n/k)$**
7. **1..n 中与 n 互质的数的和为 n*Φ(n)/2**
   证明提示：if gcd(n,i)=1 then gcd(n,n-i)=1 (1<=i<=n)
8. **1..n 与 n 的最大公约数是 d 的数的个数为 Φ(n/d)，和为(n/d)*Φ(n/d)/2**
9. 奇素数 p 的原根的个数为 $\phi(p-1)$
   显然 p 一定有原根，设 p 的一个原根为 x，则$\{x, x^2, ..., x^{p-1}\} = \{1, 2, ..., n\}$，我们考虑 x^i 是不是原根。
   考虑集合$\{x^i, x^{2i}, ..., x^{(p-1)i}\}$，假设存在 $x^{ai} \equiv x^{bi}$ $(1\le a < b\le p-1)$，则 $ai \equiv bi \bmod p - 1$, $(b-a)\, i \ne 0 \bmod p-1$.

若(i, p-1) = 1，则 b - a ≡ 0 mod p – 1，a = b，矛盾，故 x^i 是原根

若(i, p-1) ≠ 1，无此结论. 事实上 $x^{(p-1)/(i,p-1)} = 1$，即 $ord(x^i) \le (p-1)/(i, p-1) < p-1$，故 x^i 不是原根。

综上 xi 是原根当且仅当(i, p – 1) = 0

10. 设 n 为一个正整数，则

$$\sum_{d|n} \phi(d) = n$$

求法

筛法

```
//时间复杂度 O(n)
//已经通过 pku2478, pku2480, zju1759, hdu2837, pku2992
const int maxn = 100000 + 5;
int phi[maxn]; //phi = 0 表示还没有被比他小的数筛掉,是质数
int primes[maxn], pcnt = 0;

void init(int n = maxn - 1){
	phi[1] = 1;
	for (int i = 2; i <= n; i++){
		if (phi[i] == 0){
			primes[pcnt++] = i; phi[i] = i - 1;
		}
		for (int j = 0, x; j < pcnt && (x = i * primes[j]) <= n; j++){
			if (i % primes[j] == 0){
				phi[x] = phi[i] * primes[j]; break;
			} else{
				phi[x] = phi[i] * (primes[j] - 1);
			}
		}
	}
}
```

单个

```
T calcphi(T x){ //已经通过 fzu1759, hdu2837
	if (x < maxn-1) return phi[x];
	T ret = 1;
	for (int i = 0; i < pcnt; i++){
		int p = primes[i];
		if (x % p == 0){
```

```
			ret *= p-1; x /= p;
			while (x % p == 0){ ret *= p; x /= p; }
			if (x < maxn-1) return ret * phi[x];
		}
	}
	return ret * (x - 1); // sqr(maxn) > x
}

T calcphi(T p[], int r[], int c){ //已经通过 pku2480
	T ret = 1;
	rep(i, c) if (r[i] > 0) ret *= (p[i] - 1) * pow(p[i], r[i] - 1);
	return ret;
}
```

莫比乌斯函数

$$\mu(n) = \begin{cases} 1 & \text{if n} = 1 \\ 0 & \text{if p}^2 | \text{ n for some prime p} \\ (-1)^r & \text{if n} = \text{p}_1 \dots \text{p}_r\text{, where the p}_\text{i} \text{ are distinct primes} \end{cases}$$

性质

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & \text{if n} = 1 \\ 0 & \text{if n} > 1 \end{cases}$$

莫比乌斯函数的应用

1..n 中与 k 互质的数的个数

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

莫比乌斯反演定理

设 *f(n)*和 *g(n)*是定义在自然数集 N 上的两个函数，若对任意自然数 n，有

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

则可将 g 表示成 f 的函数

$$g(n) = \sum_{d|n} \mu(d) g\left(\frac{n}{d}\right)$$

反之亦然

Jordan's totient function 约旦欧拉函数

DEFINATION 定义

$J_k(n)$ = the number of all vectors $(a_1, \ldots, a_k)$ belonging to Z_+^k with properties $a_i <= n$, i = 1, .., k and $\gcd(a_1, \ldots, a_k, n) = 1$.

PROPERTIES 性质

1. The function J_k is multiplicative, i.e. for any positive integers m, n with gcd(m, n) = 1 the relation $J_k(mn) = J_k(m)J_k(n)$ holds.
2. If the unique prime decomposition of n is $n = p_1^{\alpha_1} \ldots p_m^{\alpha_m}$ then
$$J_k(n) = n^k\left(1-\frac{1}{p_1^k}\right)\ldots\left(1-\frac{1}{p_m^k}\right) = \prod_i p_i^{k(\alpha_i-1)}(p_i^k-1)$$
3. (Gauss' type formula) The following formular holds
$$n^k = \sum_{d|n} J_k(d)$$

其他积性函数

n 的所有因子(约数)经过 f 运算之后的和$\sum_{d|n} f(d)$, e.g.
n 的因子(约数)个数 $d(n)=(1+r_1)(1+r_2)\ldots(1+r_k)$
n 的所有因子(约数)之和$= (1 + p_1 + \cdots + p_1^{r_1})(1 + p_2 + \cdots + p_2^{r_2}) \ldots (1 + p_k + \cdots + p_k^{r_k})$

高斯质数(Gaussian Prime)

A Gaussian integer a+bi is a Gaussian prime if and only if either:
1. One of a, b is zero and other is a prime number of the form 4n+3 (with n a nonnegative integer) or its negative –(4n+3), or
2. Both are nonzero and a^2+b^2 is a prime number. (which will not be of the form 4n+3)

The Determinants of GCD matrices

DEFINITION Let $S = \{x_1, \ldots, x_n\}$ be a finite ordered set of distinct positive integers. The greatest common divisor matrix (or GCD matrix) defined on S is gived by
$$\begin{bmatrix} (x_1,x_1) & \cdots & (x_1,x_n) \\ \vdots & \ddots & \vdots \\ (x_n,x_1) & \cdots & (x_n,x_n) \end{bmatrix}$$
and is denoted by [S]. In other words, for $S = \{x_1, .., x_n\}$, $[S] = (s_{ij})_{n\times n}$, where $s_{ij} = (x_i, x_j) = \gcd(x_i, x_j)$

A set V of positive integers is said to be factor closed (FC) iff all positive factors of any element of V belong to V

THEOREM A Let S = $\{x_1, \ldots, x_n\}$ be an ordered set of distinct positive integers, and S' = $\{x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+s}\}$ the minimal FC ordered set containing S, where $x_{n+1} < \ldots < x_{n+s}$. Define the $n\times(n+s)$ matrix A = (a_{ij}) by

$$a_{ij} = \begin{cases} \sqrt{\varphi(x_j)} & if\, x_j | x_i \\ 0 & otherwise \end{cases}$$

where ϕ is Euler's totient function. The [S] = A AA^T.

COROLLARY A. Let S = $\{x_1, \ldots, x_n\}$ be as above. If S is FC (ordered is unnecessary), then

$$\mathrm{Det}[S] = \phi(x_1)\ldots\phi(x_n).$$

THEOREM B Let S = $\{x_1, \ldots, x_n\}$ be an ordered set of distinct positive integers, and [S] the GCD matrix defined on S. Then

$$\mathrm{Det}[S] >= \phi(x_1)\ldots\phi(x_n).$$

And equality holds iff S is FC.

二次剩余

定义 1：设素数 p > 2, d 是整数, p ∤ d, 若同余方程$x^2 \equiv d\ (mod\ p)$有解，则称 d 是模 p 的二次剩余；若无解，则称 d 是模 p 的二次非剩余

定理 2：在模 p 的一个既约剩余系中，恰有(p - 1) / 2 个模 p 的二次剩余，(p - 1) / 2 个模 p 的二次非剩余。此外，若 d 是模 p 的二次剩余，则同余方程$x^2 \equiv d\ (mod\ p)$的解数为 2

定理 3：设素数$p > 2$, p ∤ d, 那么 d 是模 p 的二次剩余 *iff.* $d^{(p-1)/2} \equiv -1 (mod\ p)$

定义 4：设素数$p > 2$, 定义整变数 d 的函数

$$\left(\frac{d}{p}\right) = \begin{cases} 1 & d\text{是模}p\text{的二次剩余} \\ -1 & d\text{是模}p\text{的二次非剩余} \\ 0 & p\mid d \end{cases}$$

我们把$\left(\frac{d}{p}\right)$称为模 p 的 Legendre 符号，则

$$\left(\frac{d}{p}\right) \equiv d^{\frac{p-1}{2}} (mod\ p)$$

定义 5：设奇数$P > 1$, $P = p_1 \cdots p_s$, $p_j\ (1 \le j \le s)$是素数，定义

$$\left(\frac{d}{P}\right) = \left(\frac{d}{p_1}\right) \cdots \left(\frac{d}{p_s}\right)$$

这里$\left(\frac{d}{p_j}\right)\ (1 \le j \le s)$是$p_j$的 Legendre 符号，我们把$\left(\frac{d}{P}\right)$称为是 Jacobi 符号。

性质 6：Jacobi 符号有以下性质

- $\left(\frac{1}{P}\right) = 1$, $\left(\frac{-1}{P}\right) = (-1)^{\frac{P-1}{2}}$, $\left(\frac{2}{P}\right) = (-1)^{\frac{P^2-1}{8}}$
- $\left(\frac{d}{P}\right) = \begin{cases} 0 & (d, P) > 1 \\ \pm 1 & (d, P) = 1 \end{cases}$

- $\left(\frac{d}{P}\right) = \left(\frac{d+P}{P}\right)$
- $\left(\frac{cd}{P}\right) = \left(\frac{c}{P}\right)\left(\frac{d}{P}\right)$
- $\left(\frac{d}{P_1P_2}\right) = \left(\frac{d}{P_1}\right)\left(\frac{d}{P_2}\right)$
- $(d, P) = 1$ 时，$\left(\frac{d^2}{P}\right) = \left(\frac{d}{P^2}\right) = 1$

定理 7：(二次互反率)设奇数$\mathrm{P} > 1$，奇数$\mathrm{Q} > 1$，$(P, \mathrm{Q}) = 1$，我们有

$$\left(\frac{P}{Q}\right)\left(\frac{Q}{\mathrm{P}}\right) = (-1)^{\frac{P-1}{2}\cdot\frac{Q-1}{2}}$$

注意：Jacobi 符号$\left(\frac{d}{p}\right) = 1$，绝不表示二次同余方程$x^2 \equiv d\ (mod\ P)$一定有解

$$x^2 \equiv d\ (mod\ P)\text{有解 iff. 方程组}x^2 \equiv d\ \left(mod\ p_j\right)\text{有解 iff. }\left(\frac{d}{p_j}\right) = 1$$

其中$\mathrm{P} = p_1 \cdots p_s$，$p_j\ (1 \le j \le s)$是素数.

定义 8：设素数 p > 2, d 是整数, p ∤ d，若同余方程$x^n \equiv d\ (mod\ p)$有解，则称 d 是模 p 的 n 次剩余；若无解，则称 d 是模 p 的 n 次非剩余。

定理 9：同余方程$x^n \equiv d\ (mod\ p)$有解的充要条件是

$$\mathrm{d}^{\frac{p-1}{k}} \equiv 1\ (mod\ p)$$

且有解时解数为 k，其中 k = (n, p-1)

定理 10：在模 p 的一个既约剩余系中，模 p 的 n 次剩余的元素个数是(p-1) / (n, p-1)

最小二乘法

用最小二乘法拟合为函数$\varphi(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$，根据 m 组数据$(x_i, y_i)$ $(1 \le i \le m, m > n)$代入公式

$$\begin{bmatrix} m & \sum x_i & \cdots & \sum x_i^n \\ \sum x_i & \sum x_i^2 & \cdots & \sum x_i^{n+1} \\ \cdots & \cdots & \cdots & \cdots \\ \sum x_i^n & \sum x_i^{n+1} & \cdots & \sum x_i^{2n} \end{bmatrix}\begin{bmatrix} a_0 \\ a_1 \\ \cdots \\ a_n \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \\ \cdots \\ \sum x_i^n y_i \end{bmatrix}$$

即可求出各项系数a_i，从而得到的拟合函数$\varphi(x)$的表达式

特别地，当 n=1 时

$$\begin{bmatrix} m & \Sigma x_i \\ \Sigma x_i & \Sigma x_i^2 \end{bmatrix}\begin{bmatrix} \mathrm{a}_0 \\ \mathrm{a}_1 \end{bmatrix} = \begin{bmatrix} \Sigma y_i \\ \Sigma x_i y_i \end{bmatrix}$$

排列组合

组合

```
void combine(int arr[], int m, int n, int pos, int tmp[], int k){
	if (k != n)
		for (int i = pos; i < m; ++i){
			tmp[k++] = arr[i];
			combine(arr, m, n, i+1, tmp, k--);
		}
	else { // k == n
		for (int i = 0; i < n; ++i)
			printf("%d ", tmp[i]);
		printf("\n");
	}
}
```

伽马函数、阶乘

定义：$\Gamma(\mathrm{x}) = \int_0^\infty \mathrm{t}^{\mathrm{x}-1}\mathrm{e}^{-\mathrm{t}}\mathrm{dt}\ \ (\mathrm{x} \geq 0)$

```
cout << "10! = " << tgamma(10 + 1) << " = " << exp( lgamma(10 + 1) ) << endl;
```

Squarefree 数

// n 是 squarefree 数 iff $\mu(n) \neq 0$.

小于 n 的 squarefree 数的个数 $\mathrm{Q}(n) = \frac{6n}{\pi^2} + \mathrm{O}(\sqrt{n})$

求 1..a 和 1..b 中互质的数(最大公约数为 k 的数)的对数

把(i,j)按照最大公约数划分等价类，则

$$\mathrm{cnt}(a, \mathrm{b}) = \sum_{i=1}^{\min(\mathrm{a,b})} \left\lfloor \frac{a}{i} \right\rfloor \left\lfloor \frac{b}{i} \right\rfloor \mu(i)$$

注：无序对的数目为(设 a<=b)

$$\mathrm{cnt}(a, \mathrm{b}) - \frac{\mathrm{cnt}(a, a) - 1}{2}$$

用 squarefree 数曾经已通过 tju3317, cugb1213, hdu1695

连分数（Continued Fractions of Rationals）

```
template<typename INT>
void contFract(INT m, INT n, vector<INT> &ans){
	while (n){
		ans.push_back( m / n );
		m %= n; swap(m, n);
	}
}
```

求<=n 的素数的个数（π(n)）

```
#include <ext/hash_map> //n <= 1000000000，已经通过 spoj pcount
using namespace __gnu_cxx;

const int maxn = 1000000000 + 5, maxtbl = 1000000 + 5;
const double eps = 1e-8;
int primes[maxtbl], pcnt = 0, pi_tbl[maxtbl], isprime[maxtbl];

void init(int n = maxtbl - 1){
	isprime[0] = isprime[1] = 0;
	for (int i = 2; i <= n; i++)
		isprime[i] = 1;
	pi_tbl[0] = pi_tbl[1] = 0;
	for (int i = 2; i <= n; i++){
		if ( isprime[i] ){
			primes[ pcnt++ ] = i;
			pi_tbl[i] = pi_tbl[i-1] + 1;
		} else {
			pi_tbl[i] = pi_tbl[i-1];
		}
		for (int j = 0, x; (x = i * primes[j]) <= n; j++ ){
			isprime[x] = 0;
			if (primes[j] % i == 0) break;
		}
	}
}

int p2(int x, int y){
	int sqrtx = (int)floor( sqrt(x) + eps );
	int ret = 0;
	for (int p = y + 1; p <= sqrtx; p++){
		if ( !isprime[p] ) continue;
```

```
        ret += pi_tbl[x / p] - pi_tbl[p] + 1;
    }
    return ret;
}

typedef long long LL;
struct hashLL { size_t operator()(LL x) const{ return (size_t) ( (x >> 32) ^ x ); } };
hash_map< LL, int, hashLL > mm;

int phi(int x, int b){
    if (b == -1) return x;
    if (x == 0) return 0;
    LL h = (((LL)x) << 32) + b;

    hash_map< LL, int, hashLL >::iterator iter = mm.find( h );
    if ( iter != mm.end() )
        return iter->second;
    return mm[ h ] = phi(x, b - 1) - phi(x / primes[b], b - 1);
}

int pi(int x){
    int y = (int)ceil(pow(x, 1.0 / 3.0) - eps), a = pi_tbl[y];
    return phi(x, a-1) + a - 1 - p2(x, y);
}

int main(){
    int n;
    while ( cin >> n ){
        init( (int)ceil( pow(n, 2.0 / 3.0) + eps ) );
        cout << pi(n) << endl;
    }
}
```

典型例题

DESCRIPTION: find $\left\lfloor\left(\sqrt{a}+\sqrt{b}\right)^{n}\right\rfloor \bmod m$, where $0 < b - a < 1 + 2\sqrt{a}$ and n is even.

SOLUTION:

$$\text{ans} = \begin{cases} \qquad\qquad\quad - \;\; 1 & n = 4k+2 \\ 2\left(-(a-b)^2\right)^{\frac{n}{4}} \;\; - \;\; 1 & n = 4k \end{cases} \bmod \text{m}$$

PROOF:

题目保证了$0<\sqrt{b}-\sqrt{a}<1$，考虑$\mathrm{x}_n=\left(\sqrt{b}+\sqrt{a}\right)^{2n}=\left(a+b+2\sqrt{ab}\right)^n$，$\mathrm{y}_n=\left(\sqrt{b}-\sqrt{a}\right)^{2n}=\left(a+b-2\sqrt{ab}\right)^n$和$\mathrm{z}_n=\mathrm{x}_n+\mathrm{y}_n$. 那么$\mathrm{x}_1$和$\mathrm{y}_1$是方程$\mathrm{u}^2-2(a+b)u+(a-b)^2=0$的根，于是有$\mathbf{z}_{n+2}=\mathbf{2}(\boldsymbol{a}+\boldsymbol{b})\boldsymbol{z}_{n+1}-(\boldsymbol{a}-\boldsymbol{b})^2\boldsymbol{z}_n$. 又显然$\mathrm{y}_n<1$, 所以$\lfloor\mathbf{x}_n\rfloor=\boldsymbol{z}_n-\mathbf{1}$，于是由$\mathbf{z}_0=\mathbf{2},\mathbf{z}_1=\mathbf{2}(\boldsymbol{a}+\boldsymbol{b})$，利用矩阵乘法就可以求出任意$\lfloor\mathrm{x}_n\rfloor \bmod m$了，时间复杂度$\mathrm{O}(\log n)$

Sum{ gcd(i, j) : 1 <= i <= x, 1 <= j <= y }

$$\sum_{i=1}^{x}\sum_{j=1}^{y}gcd(i,j)=\sum_{i=1}^{x}\sum_{j=1}^{y}\sum_{d|\gcd(i,j)}\varphi(d)=\sum_{i=1}^{x}\sum_{j=1}^{y}\sum_{d|i\wedge d|j}\varphi(d)=\sum_{d}\varphi(d)\sum_{1\le i\le x\wedge d|i}\ \sum_{1\le j\le y\wedge d|j}1$$
$$=\sum_{d}\varphi(d)\left(\sum_{1\le i\le x\wedge d|i}1\right)\left(\sum_{1\le j\le y\wedge d|j}1\right)==\sum_{d}\varphi(d)\left\lfloor\frac{x}{d}\right\rfloor\left\lfloor\frac{y}{d}\right\rfloor$$

复杂度$\mathrm{O}(\min(x,y))$
继续优化：

可以看出$\left\lfloor\frac{x}{d}\right\rfloor$的取值只有$2\lfloor\sqrt{\mathrm{x}}\rfloor$种，同理$\left\lfloor\frac{y}{d}\right\rfloor$的取值只有$2\lfloor\sqrt{\mathrm{y}}\rfloor$种，并且相同取值对应的 d 是一个连续的区间，因此$\left\lfloor\frac{x}{d}\right\rfloor$和$\left\lfloor\frac{y}{d}\right\rfloor$都相同的区间，最多只有$2\lfloor\sqrt{\mathrm{x}}\rfloor+2\lfloor\sqrt{\mathrm{y}}\rfloor$个，这样 d 的枚举量就缩小为$\mathrm{O}\left(\sqrt{\mathrm{x}}+\sqrt{\mathrm{y}}\right)$了，注意需要预处理函数$\varphi$的部分和。

Sum{ lcm(i, j) : 1 <= i <= x, 1 <= j <= y }

$$\sum_{i=1}^{x}\sum_{j=1}^{y}\mathrm{lcm}(i,j)=\sum_{d}\sum_{i=1}^{x}\sum_{1\le j\le y\le\wedge\gcd(i,j)=d}\frac{ij}{d}=\sum_{d}\sum_{i=1}^{x}\sum_{j=1}^{y}[gcd(i,j)=d]\frac{ij}{d}$$
$$=\sum_{d}d\sum_{i=1}^{\left\lfloor\frac{x}{d}\right\rfloor}\sum_{j=1}^{\left\lfloor\frac{y}{d}\right\rfloor}[gcd(i,j)=1]ij$$

令

$$\mathrm{f}(x,y)=\sum_{i=1}^{x}\sum_{j=1}^{y}[gcd(i,j)=1]ij$$

则

$$f(x,y)=\sum_{i=1}^{x}\sum_{j=1}^{y} ij \sum_{d|\gcd(i,j)} \mu(d)=\sum_{d}\mu(d)\sum_{1\le i\le x\wedge d|x}\ \sum_{1\le j\le y\wedge d|y} ij$$

$$=\sum_{d}\mu(d)\left(\sum_{1\le i\le x\wedge d|x} i\right)\left(\sum_{1\le j\le y\wedge d|y} j\right)=\sum_{d}\mu(d)\left(\sum_{1\le i\le\left\lfloor\frac{x}{d}\right\rfloor} id\right)\left(\sum_{1\le j\le\left\lfloor\frac{y}{d}\right\rfloor} j\,d\right)$$

$$=\sum_{d}\mu(d)\frac{d\left\lfloor\frac{x}{d}\right\rfloor\left(\left\lfloor\frac{x}{d}\right\rfloor+1\right)}{2}\cdot\frac{d\left\lfloor\frac{y}{d}\right\rfloor\left(\left\lfloor\frac{y}{d}\right\rfloor+1\right)}{2}$$

$$=\frac{1}{4}\sum_{d}\mu(d)d^2\left\lfloor\frac{x}{d}\right\rfloor\left(\left\lfloor\frac{x}{d}\right\rfloor+1\right)\left\lfloor\frac{y}{d}\right\rfloor\left(\left\lfloor\frac{y}{d}\right\rfloor+1\right)$$

带回原式得

$$\sum_{i=1}^{x}\sum_{j=1}^{y}\operatorname{lcm}(i,j)=\sum_{d} d\cdot f\left(\left\lfloor\frac{x}{d}\right\rfloor,\left\lfloor\frac{y}{d}\right\rfloor\right)$$

$$=\frac{1}{4}\sum_{d} d\sum_{d'}\mu(d')d'^2\left\lfloor\frac{\left\lfloor\frac{x}{d}\right\rfloor}{d'}\right\rfloor\left(\left\lfloor\frac{\left\lfloor\frac{x}{d}\right\rfloor}{d'}\right\rfloor+1\right)\left\lfloor\frac{\left\lfloor\frac{y}{d}\right\rfloor}{d'}\right\rfloor\left(\left\lfloor\frac{\left\lfloor\frac{y}{d}\right\rfloor}{d'}\right\rfloor+1\right)$$

$$=\frac{1}{4}\sum_{d} d\sum_{d'}\mu(d')d'^2\left\lfloor\frac{x}{dd'}\right\rfloor\left(\left\lfloor\frac{x}{dd'}\right\rfloor+1\right)\left\lfloor\frac{y}{dd'}\right\rfloor\left(\left\lfloor\frac{y}{dd'}\right\rfloor+1\right)$$

$$=\frac{1}{4}\sum_{d} d\left\lfloor\frac{x}{d}\right\rfloor\left(\left\lfloor\frac{x}{d}\right\rfloor+1\right)\left\lfloor\frac{y}{d}\right\rfloor\left(\left\lfloor\frac{y}{d}\right\rfloor+1\right)\sum_{d'|d}\mu(d')d'$$

令

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

显然 g 满足积性，可以通过线性筛法预处理

Given n and k, find max{ i : k^i divides n! } – 水 – java

```java
public class Main {
	public static BigInteger rho(BigInteger n){
		BigInteger x, y, d, c;
		int k, i;
		for (;;){
			c = BigInteger.valueOf( rand.nextLong() )
				.mod( n.subtract(BigInteger.ONE) )
				.add(BigInteger.ONE);
			x = y = BigInteger.valueOf( rand.nextLong() ).mod( n );
			k = 2; i = 1;
			do {
				i++;
				d = x.subtract(y).abs().gcd(n);
```

```
                if (d.compareTo(BigInteger.ONE) > 0 && d.compareTo(n) < 0)
                    return d;
                if ( i == k ){
                    y = x; k *= 2;
                }
                x = x.multiply(x).add(c).mod(n);
            } while ( !x.equals(y) ); //千万不要写成 x != y
        }
    }
    public static void factorize(BigInteger n){
        if ( n.equals(BigInteger.ONE) ) return;
        if ( n.isProbablePrime(10) ){ // do sth.
            Integer v = Table.get(n); Table.put(n, v == null ? 1 : ++v);;
        } else {
            BigInteger d = rho(n); factorize(d); factorize(n.divide(d));
        }
    }
    public static void main(String args[]){ //hdu3988
        int re = cin.nextInt();
        for (int ri = 1; ri <= re; ri++){
            BigInteger n = cin.nextBigInteger(), k = cin.nextBigInteger();
            Table = new HashMap<BigInteger, Integer>(); factorize(k);
            Iterator<BigInteger> iter = Table.keySet().iterator();
            BigInteger r = inf;
            while ( iter.hasNext() ){
                BigInteger p = iter.next();
                BigInteger rr = BigInteger.ZERO, BigInteger nn = n.divide(p);
                while ( !nn.equals(BigInteger.ZERO) ){
                    rr = rr.add( nn ); nn = nn.divide(p);
                }
                rr = rr.divide( BigInteger.valueOf( Table.get(p) ) ); r = r.min(rr);
            }
            System.out.printf("Case %d: ", ri);
            System.out.println(r.equals(inf) ? "inf" : r);
        }
    }

    static HashMap<BigInteger, Integer> Table;
    static BigInteger inf = new BigInteger("9223372036854775808" );
    static Random rand = new Random();
    static Scanner cin = new Scanner( new BufferedInputStream(System.in) );
}
```

数据类型

分数

注意：若$\frac{a}{b}$和$\frac{c}{d}$都是最简分数，则$\frac{\frac{ad}{(b,d)}+\frac{bc}{(b,d)}}{[b,d]}$不一定是最简分数

```
int sign(int x){ return x > 0 ? 1 : x == 0 ? 0 : -1;}

struct Fraction{
	int a, b;
	Fraction(int x = 0, int y = 1){
		int m = gcd(abs(x), abs(y));
		a = x / m * sign(y);
		if (a == 0) b = 1; else b = abs(y / m);
	}
	Fraction operator+(const Fraction &f){
		int m = gcd(b, f.b);
		return Fraction(f.b/m*a+b/m*f.a, b/m&f.b);
	}
	Fraction operator-(const Fraction &f){
		int m = gcd(b, f.b);
		return Fraction(f.b/m*a - b/m*f.a, b/m * f.b);
	}
	Fraction operator* (const Fraction &f){
		int m1 = gcd(abs(a), f.b);
		int m2 = gcd(abs(b), f.b);
		return Fraction((a/m1)*(f.a/m2), (b/m1)*(f.b/m2));
	}
	Fraction operator/ (const Fraction &f){
		return (*this) * Fraction(f.b, f.a);
	}
	friend ostream &operator<<(ostream &cout , const Fraction &f){
		cout << f.a; if (f.b != 1) cout << "/" << f.b; return cout;
	}
};
```

大实数

```
const double Ten = 10;
const long double eps = 1e-15;
#define abs(x) (x < 0 ? (-x) : (x))
```

```
struct MyReal{
	typedef MyReal self;
	long double a; int e;

	MyReal(long double ia = 0.0, int ie = 0){
		a = ia, e = ie;
		if (abs(a) < eps){ a = 0.0; e = 0; }
		else{
			int d = (int)log10(abs(a));
			e += d; a*= pow(Ten, -d);
		}
	}
	void adjust(int d){
		d -= e; e += d; a *= pow(Ten, -d);
	}
	friend self operator- (const self& x){
		return MyReal(-x.a, x.e);
	}
	friend self operator+ (const self& x, const self& y){
		if (x.e > y.e) {
			self t = y; t.adjust(x.e); t.a += x.a; return t;
		} else {
			self t = x; t.adjust(y.e); t.a += y.a; return t;
		}
	}
	friend self operator- (const self& x, const self& y){
		if (x.e > y.e) {
			self t = y; t.adjust(x.e); t.a = x.a - t.a; return t;
		} else {
			self t = x; t.adjust(y.e); t.a = t.a - y.a; return t;
		}
	}
	friend ostream &operator<< (ostream& cout, const self& x){
		cout << x.a << "e" << x.e;
	}
	friend int cmp(const self& x, const self& y){
		if (x.e != y.e) return x.e - y.e;
		return x.a > y.a + eps ? 1 : x.a < y.a - eps ? -1 : 0;
	}
};

int main(){
	MyReal a;
```

```
	cout << MyReal(1) + MyReal(10) << endl;
	cout << MyReal(10) << endl;
	cout << MyReal(99) - MyReal(1) << endl;
}
```

矩阵运算

线性方程组有解 iff. r = Rank{系数矩阵 A} = Rank{增广矩阵[A b]}
当 r = n(未知数个数)时，线性方程组有唯一解
当 r < n(未知数个数)时，线性方程组有多解，自由变量个数为 n-r

```
//#define COMPLEX_ELE
const int maxn = 100;

#ifdef COMPLEX_ELE
	typedef complex<double> Tvalue;
#else
	typedef double Tvalue;
#endif
typedef double Tabs;

struct mat{
	int m, n;
	Tvalue data[maxn][maxn];
	Tvalue *operator[] (int i){return data[i];}
	const Tvalue *operator[] (int i) const{return data[i];}
	void swap_row(int i1, int i2){for (int j = 0; j < n; j++) swap(data[i1][j], data[i2][j]);}
	void swap_col(int j1, int j2){for (int i = 0; i < m; i++) swap(data[i][j1], data[i][j2]);}
	Tabs max_element(int k, int &is, int &js){
		//寻找 is，js 使得 |data[is][js]| = max{|data[k..m-1][k..n-1]|}
		Tabs pivot = 0, tmp;
		for (int i = k; i < m; i++){
			for (int j = k; j < n; j++){
				tmp = abs(data[i][j]);
				if (tmp > pivot){pivot = tmp; is = i; js = j;}
			}
		}
		return pivot;
	}
};

mat operator* (const mat& x, const mat& y){
	//时间复杂度 O(n ^ 3)
	mat z; assert(x.c == y.r);
	z.r = x.r; z.c = y.c;
```

```
	rep(i, z.r) rep(j, z.c) z.data[i][j] = 0;
	rep(i, z.r) rep(k, x.c) if ( a.data[i][k] ) //超强剪枝
		rep(j, z.c) c[i][j] += a[i][k] * b[k][j];
	return c;
}

int rank(mat a){ //求矩阵的秩，时间复杂度 O(n ^ 3)
	int p = min(a.m, a.n), rank = 0, is, js;
	Tvalue tmp;
	for (int k = 0; k < p; k++){
		Tabs pivot = a.max_element(k, is, js);
		if (pivot < eps) return rank;
		++rank;
		if (k != is) a.swap_row(k, is);
		if (k != js) a.swap_col(k, js);
		for (int i = k+1; i < a.m; i++){
			tmp = a[i][k] / a[k][k];
			//a.data[i][k] = 0;
			for (int j = k+1; j < a.n; j++)
				a[i][j] -= tmp * a[k][j];
		}
	}
	return rank;
}

Tvalue det(mat a){ //时间复杂度 O(n^3),对于 int 版本,参考 P97 行列式取模(det_mod)[金斌论文]
	assert(a.m == a.n);
	int n = a.n;
	Tvalue det = 1.0, tmp;
	int flag = 1, is, js;
	for (int k = 0; k < n-1; k++){ //最多进行 n-1 次消去
		Tabs pivot = a.max_element(k, is, js);
		if (pivot < eps) return 0.0;
		if (k != is) {a.swap_row(is, k); flag = -flag;}
		if (k != js) {a.swap_col(js, k); flag = -flag;}
		for (int i = k+1; i < n; i++){
			tmp = a[i][k] / a[k][k];
			//a.data[i][k] = 0
			for (int j = k+1; j < n; j++)
				a[i][j] -= tmp * a[k][j];
		}
		det *= a[k][k];
	}
	return (Tvalue)flag * det * a[n-1][n-1];
```

```
}

bool inv(mat &a){ //时间复杂度 O(n ^ 3)
	static int is[maxn], js[maxn];
	if (a.m != a.n) return false;
	int &n = a.n;
	for (int k = 0; k < n; k++){
		Tabs pivot = a.max_element(k, is[k], js[k]);
		if (pivot < eps) return false;
		if (k != is[k]) a.swap_row(k, is[k]);
		if (k != js[k]) a.swap_col(k, js[k]);
		a.data[k][k] = 1.0 / a[k][k];
		for (int j = 0; j < n; j++){
			if (j == k) continue;
			a[k][j] *= a[k][k];
		}
		for (int i = 0; i < n; i++) if (i != k)
			for (int j = 0; j < n; j++) if (j != k)
				a[i][j] -= a[k][j] * a[i][k];
		for (int i = 0; i < n; i++) if (i != k)
			a[i][k] *= -a[k][k];
	}
	for (int k = n-1; k >= 0; k--){
		if (k != js[k]) a.swap_col(k, js[k]);
		if (k != is[k]) a.swap_row(k, is[k]);
	}
	return true;
}

bool gs(mat a, Tvalue b[]){
	//高斯消去法解线性方程组，已经通过 poj1538
	//a 系数矩阵，b[]常数向量，将解向量存到 b[]中
	if (a.m != a.n) return false;
	int &n = a.n;
	static int is, js[maxn];
	for (int k = 0; k < n; k++){
		Tabs pivot = a.max_element(k, is, js[k]);
		if (pivot < eps) return false;
		if (k != is){
			a.swap_row(k, is);
			swap(b[k], b[is]);
		}
		if (k != js[k]) a.swap_col(k, js[k]);
		a[k][k] = 1.0 / a[k][k];
```

```
            for (int j = k+1; j < n; j++)
                a[k][j] *= a[k][k];
            b[k] *= a[k][k];
            //a[k][k] = 1.0;
            for (int i = k+1; i < n; i++){
                for (int j = k+1; j < n; j++)
                    a[i][j] -= a[i][k] * a[k][j];
                b[i] -= a[i][k] * b[k];
                //a[i][k] = 0;
            }
        }
        for (int i = n-2; i >= 0; i--){
            Tvalue t = 0.0;
            for (int j = i+1; j < n; j++)
                t += b[j] * a[i][j];
            b[i] -= t;
        }
        for (int k = n-1; k >= 0; k--){
            if (js[k] != k) swap(b[k], b[js[k]]);
        }
        return true;
}

bool gsjd(mat a, mat &b){
        //用高斯-约当消去法解线性方程组
        //a 系数矩阵，b 常数矩阵，将解矩阵存到 b 中
        if (a.m != a.n) return false; if (a.m != b.m) return false;
        int &n = a.n;
        static int is, js[maxn];
        for (int k = 0; k < n; k++){
            Tabs pivot = a.max_element(k, is, js[k]);
            if (pivot < eps) return false;
            if (k != is)
                a.swap_row(k, is), b.swap_row(k, is);
            if (k != js[k]) a.swap_col(k, js[k]);
            a[k][k] = 1.0 / a[k][k];
            for (int j = k+1; j < n; j++)
                a[k][j] *= a[k][k];
            for (int j = 0; j < b.n; j++)
                b[k][j] *= a[k][k];
            //a[k][k] = 1.0
            for (int i = 0; i < n; i++) if (i != k){
                for (int j = k+1; j < n; j++)
                    a[i][j] -= a[i][k] * a[k][j];
```

```
			for (int j = 0; j < b.n; j++)
				b[i][j] -= a[i][k] * b[k][j];
			//a[i][k] = 0.0;
		}

		for (int i = 0; i < n; i++) if (i != k){
			for (int j = k+1; j < n; j++)
				a[i][j] -= a[i][k] * a[k][j];
			for (int j = 0; j < b.n; j++)
				b[i][j] -= a[i][k] * b[k][j];
			//a[i][k] = 0.0;
		}
	}
	for (int k = n-1; k >= 0; k--){
		if (js[k] != k) b.swap_row(k, js[k]);
	}
	return true;
}

mat unit(int n){ //返回 n 阶单位阵
	//omitted
}

#ifndef COMPLEX_ELE
bool mhdqr(mat &a, complex<Tvalue> z[], int itmax = 60){
	//求实上 H 矩阵 a 的特征值，存到 z[]中，最大迭代次数 itmax
	//成功：返回 1，失败：返回 0
	assert(a.m == a.n);
	const int &n = a.n;
	typedef complex<Tvalue> cpl;
	if (itmax <= 0) return 0;
	if (n == 1){
		z[0] = cpl(a[0][0], 0);
		return 1;
	}
	if (n == 2){
		Tvalue b = a[0][0] + a[1][1];
		Tvalue c = a[0][0]*a[1][1] - a[0][1]*a[1][0];
		Tvalue s = sqr(b) - 4.0 * c;
		Tvalue sqrts = sqrt(fabs(s));
		if (s > 0.0){
			if (b > 0.0) z[0] = cpl((b + sqrts) * 0.5, 0); else z[0] = cpl((b - sqrts) * 0.5, 0);
			z[1] = b - z[0];
		}
```

```
        else{
            z[0] = cpl(b * 0.5, sqrts * 0.5);
            z[1] = conj(z[0]);//x[1]为 x[0]的共轭
        }
        return 1;
    }
    int is1 = 0, is2 = 0, n1;
    while (is2 < n-1){
        is2++;
        if (abs(a[is2][is2-1]) < eps * (abs(a[is2-1][is2-1]) + abs(a[is2][is2]))){
            n1 = is2 - is1;
            mat *p = new mat;
            p->m = p->n = n1;
            for (int i = 0; i < n1; i++)
                for (int j = 0; j < n1; j++)
                    p->data[i][j] = a[i+is1][j+is1];
            mhdqr(*p, z + is1, itmax);
            delete p;
            is1 = is2;
        }
    }
    if (is1 > 0 && is1 < n){
        n1 = n - is1;
        mat *p = new mat;
        p->m = p->n = n1;
        for (int i = 0; i < n1; i++)
            for (int j = 0; j < n1; j++)
                p->data[i][j] = a[i+is1][j+is1];
        mhdqr(*p, z + is1, itmax);
        delete p;
        return 1;
    }
    else if(is1 == n){
        return 1;
    }

    Tvalue x, y, p, q, r, s;
    Tvalue q00, q01, q02, q11, q12, q22;
    for (int k = 0; k < n-1; k++){
        if (k == 0){
            x = a[n-2][n-2] + a[n-1][n-1];
            y = a[n-2][n-2]*a[n-1][n-1] - a[n-2][n-1]*a[n-1][n-2];
            p = a[0][0] * (a[0][0]-x) + a[0][1] * a[1][0] + y;
            q = a[1][0] * (a[0][0] + a[1][1] - x);
```

```
        r = a[1][0] * a[2][1];
    }
    else{
        p = a[k][k-1];
        q = a[k+1][k-1];
        if (k != n-2) r = a[k+2][k-1];
        else r = 0.0;
    }
    if ((abs(q) + abs(q) + abs(r)) > eps){
        if (p < 0.0) s = -sqrt(p*p+q*q+r*r);
        else s = sqrt(p*p+q*q+r*r);

        if (k != 0) a[k][k-1] = -s;
        q00 = -p / s;
        q01 = -q / s;
        q02 = -r / s;
        q11 = -q00 - q02 * r / (p+s);
        q12 = q01 * r / (p+s);
        q22 = -q00 - q01 * q / (p+s);
        int j;
        for (j = k; j < n; j++){
            p = q00 * a[k][j] + q01 * a[k+1][j];
            q = q01 * a[k][j] + q11 * a[k+1][j];
            r = q02 * a[k][j] + q12 * a[k+1][j];
            if (k != n-2){
                p += q02 * a[k+2][j];
                q += q12 * a[k+2][j];
                r += q22 * a[k+2][j];
                a[k+2][j] = r;
            }
            a[k][j] = p;
            a[k+1][j] = q;
        }
        j = k + 3;
        if (j >= n-1) j = n-1;

        for (int i = 0; i <= j; i++){
            p = q00 * a[i][k] + q01 * a[i][k+1];
            q = q01 * a[i][k] + q11 * a[i][k+1];
            r = q02 * a[i][k] + q12 * a[i][k+1];
            if (k != n-2){
                p += q02 * a[i][k+2];
                q += q12 * a[i][k+2];
                r += q22 * a[i][k+2];
```

```
                    a[i][k+2] = r;
                }
                a[i][k] = p;
                a[i][k+1] = q;
            }
        }
        if (k > 0){
            a[k+1][k-1] = 0.0;
            if (k != n-2) a[k+2][k-1] = 0.0;
        }
    }
    return mhdqr(a, z, itmax-1);
}

bool qrroot(Tvalue a[], int n, complex<Tvalue> z[], int itmax = 60){
    //QR 方法求实系数多项式方程全部复根
    //多项式方程 a0 + a1x + a2x2 + ... + an-1xn-1
    assert(a[n-1] > eps);
    mat b;
    --n;
    b.m = b.n = n;
    for (int j = 0; j < n; j++)
        b[0][j] = -1.0 * a[n-1-j] / a[n];
    for (int i = 1; i < n; i++){
        for (int j = 0; j < n; j++)
            b[i][j] = 0.0;
        b[i][i-1] = 1.0;
    }
    return mhdqr(b, z, itmax);
}
#endif
```

矩阵的 LU(Doolittle)分解

$$\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & & a_{1,n-1} \\ \vdots & & \ddots & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix} \rightarrow \begin{bmatrix} u_{0,0} & u_{0,1} & \cdots & u_{0,n-1} \\ l_{1,0} & u_{1,1} & & u_{1,n-1} \\ \vdots & & \ddots & \vdots \\ l_{n-1,0} & \cdots & l_{n-1,n-2} & l_{n-1,n-1} \end{bmatrix}$$

```
bool Doolittle(double a[maxn][maxn], int n, double b[maxn]){ //时间复杂度 O(n ^ 3)
    //做分解 QA = LU，LU 记录在矩阵 A 中，Q 用数组 M 记录
    static int M[maxn];
    double tmp;
    for (int k = 0; k < n; k++){
```

```
		static double s[maxn];
		for (int i = k; i < n; i++){
			s[i] = a[i][k];
			for (int t = 0; t < k; t++)
				s[i] -= a[i][t] * a[t][k];
		}
		M[k] = k;
		for (int i = k + 1; i < n; i++)
			if (abs(s[i]) > abs(s[ M[k] ]))
				M[k] = i;
		if (abs( s[ M[k] ]) < eps) return 0;

		if (M[k] != k){
			for (int t = 0; t < n; t++)
				swap(a[k][t], a[ M[k] ][t]);
			swap(s[k], s[ M[k] ]);
		}
		a[k][k] = s[k];
		for (int j = k + 1; j < n; j++)
			for (int t = 0; t < k; t++)
				a[k][j] -= a[k][t] * a[t][j];
		tmp = 1.0 / s[k];
		for (int i = k + 1; i < n; i++)
			a[i][k] = s[i] * tmp; //s[i] / s[k];
	}

	//求 Qb
	for (int k = 0; k < n; k++)
		swap( b[k], b[ M[k] ] );

	//求解 Ly = Qb  和  Ux = y, y 和 x 都存在 b 中
	for (int i = 0; i < n; i++)
		for (int t = 0; t < i; t++)
			b[i] -= a[i][t] * b[t];

	for (int i = n-1; i >= 0; i--){
		for (int t = i+1; t < n; t++)
			b[i] -= a[i][t] * b[t];
		b[i] /= a[i][i];
	}
	return 1;
}
```

追赶法求解三对角线性方程组 - O(n)

已经通过 cugb1016

$$\begin{bmatrix} a_0 & c_0 & & & & \\ d_1 & a_1 & c_1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & d_{n-2} & a_{n-2} & c_{n-2} \\ & & & & d_{n-1} & a_{n-1} \end{bmatrix}$$

```
bool tridiagonal_linear_equation(double a[], double c[], double d[], int n, double b[]){
	static double p[maxn], q[maxn], p_inv[maxn];
	p[0] = a[0];
	if (abs(p[0]) < eps) return 0;
	p_inv[0] = 1.0 / p[0];
	for (int i = 0; i < n - 1; i++){
		q[i] = c[i] * p_inv[i];
		p[i+1] = a[i+1] - d[i+1] * q[i];
		if (abs(p[i+1]) < eps) return 0;
		p_inv[i+1] = 1.0 / p[i+1];
	}
	b[0] = b[0] / p[0];
	for (int i = 1; i < n; i++)
		b[i] = (b[i] - d[i] * b[i-1]) * p_inv[i];
	for (int i = n-2; i >= 0; i--)
		b[i] = b[i] - q[i] * b[i+1];
	return 1;
}
```

拟对角线性方程组的求解 – O(n)

$$\begin{bmatrix} a_0 & c_0 & & & & d_0 \\ d_1 & a_1 & c_1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & d_{n-2} & a_{n-2} & c_{n-2} \\ c_{n-1} & & & & d_{n-1} & a_{n-1} \end{bmatrix}$$

```
bool ext_tridiagonal_linear_equation(double a[], double c[], double d[], int n, double b[]){
	static double p[maxn], q[maxn], r[maxn], s[maxn];
	p[0] = a[0];
	for (int i = 0; i < n-1; i++){
		q[i] = c[i] / p[i];
```

```
        p[i+1] = a[i+1] - d[i+1] * q[i];
    }

    s[0] = d[0] / p[0];
    for (int i = 1; i < n-2; i++)
        s[i] = -d[i] * s[i-1] / p[i];
    s[n-2] = (c[n-2] - d[n-2] * s[n-3]) / p[n-2];

    r[0] = c[n-1];
    for (int j = 1; j < n-2; j++)
        r[j] = -r[j-1] * q[j-1];
    r[n-2] = d[n-1] - r[n-3] * q[n-3];
    r[n-1] = a[n-1];
    for (int j = 0; j < n-1; j++)
        r[n-1] -= r[j] * s[j];

    b[0] = b[0] / p[0];
    for (int i = 1; i < n-1; i++)
        b[i] = (b[i] - d[i] * b[i-1]) / p[i];
    for (int j = 0; j < n-1; j++)
        b[n-1] -= r[j] * b[j];
    b[n-1] /= r[n-1];

    b[n-2] = b[n-2] - s[n-2] * b[n-1];
    for (int i = n-3; i >= 0; i--)
        b[i] = b[i] - q[i] * b[i+1] - s[i] * b[n-1];
}
```

行列式取模 - det_mod (Zm 下的 det)

```
LL det_mod(mat a, LL m){ //O(n ^ 3 log(m)), 已经通过 spoj DETER3, HIGH
    for (int i = 0; i < a.n; i++)
        for (int j = 0; j < a.n; j++)
            a[i][j] %= m;

    LL det = 1;
    for (int k = 0; k < a.n; k++){
        for (int i = k+1; i < a.n; i++){
            while (a[i][k] != 0){
                LL t = a[k][k] / a[i][k];
                for (int j = k; j < a.n; j++){
                    a[k][j] -= a[i][j] * t; a[k][j] %= m;
                    swap(a[k][j], a[i][j]);
                }
```

```
                det = -det;
            } /*end while */
        } /*end for*/
        if (a[k][k] == 0) return 0;
        det = det * a[k][k] % m;
    } /*end for*/
    if (det < 0) det += m;
    return det;
} /*end function det_mod*/
```

解 01 矩阵方程

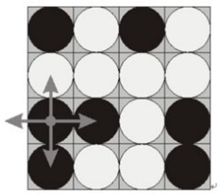
```
//已经通过 cf141-div2-E，上次使用 2012.9.30
int mat[105][105];

int solve(int r, int c){
    int I = 0, J = 0;
    for (;I < r && J < c;) {
        int is = -1;
        for (int i = I; i < r; ++i) {
            if ( mat[i][J] == 1 ) {
                is = i; break;
            }
        }
        if ( is == -1 ) {
            // x[J]是自由变量
            ++J; continue;
        }
        for (int j = J; j <= c; ++j) {
            swap( mat[I][j], mat[is][j] );
        }
        for (int i = I + 1; i < r; ++i) {
            if ( mat[i][J] == 0 ) {
                continue;
            }
            for (int j = J; j <= c; ++j) {
                mat[i][j] ^= mat[I][j];
            }
        }
        ++I; ++J;
    }
    if (I < r) {
        for (int i = I; i < r; ++i) {
            if ( mat[i][c] == 1 ) {
```

```
                    return 0;
                }
            }
        }
        static int ans[105];
        for (int i = r - 1; i >= 0; --i) {
            int first = -1;
            int val = mat[i][c];
            for (int j = 0; j < c; ++j) {
                if ( mat[i][j] ) {
                    if ( first == -1 ) {
                        first = j;
                    } else {
                        val ^= ans[j];
                    }
                }
            }
            if ( first != -1 )
                ans[first] = val;
        }
        return 1;
}
```

Flip Game

要求得到唯一的翻转方案/问有多少种翻转方案...

1. 复杂度 (2 ^ r) * c ——枚举第一行
2. 复杂度 (r * c) ^ 3 ——Z2 下的线性方程组
3. 复杂度 (r * r * c) + (r ^ 3) ——枚举第一行 + Z2 下的线性方程组



```
const int maxn = 25; // http://acm.uestc.edu.cn/problem.php?pid=1555
void solve(int r = 20, int c = 20) {
        static int a[maxn][maxn], b[maxn][maxn], f[maxn][maxn], x[maxn];
        for (int i = 1; i <= r; ++i) for (int j = 1; j <= c; ++j) scanf("%d", &a[i][j]);
        for (int i = 0; i <= r + 1; ++i) for (int j = 0; j <= c + 1; ++j) f[i][j] = b[i][j] = 0;
        for (int j = 1; j <= c; ++j) f[1][j] = (1 << j);
        for (int i = 1; i <= r; ++i)    for (int j = 1; j <= c; ++j)
            f[i+1][j] = f[i-1][j] ^ f[i][j-1] ^ f[i][j] ^ f[i][j+1] ^ a[i][j];
        for (int j = 1; j <= c; ++j) x[j] = f[r+1][j];
        for (int k = 1; k <= c; ++k) {
            int s = -1;
            for (int i = k; i <= c; ++i) if ( testbit(x[i], k) ) {
                s = i; break;
            }
```

```
        if (s == -1) TLE;
        swap(x[s], x[k]);
        for (int i = k + 1; i <= c; ++i) if ( testbit(x[i], k) ) x[i] ^= x[k];
    }
    for (int k = c; k >= 1; --k) {
        int v = testbit(x[k], 0);
        for (int i = k + 1; i <= c; ++i) if ( testbit(x[k], i) ) v ^= b[1][i];
        b[1][k] = v;
    }
    for (int i = 1; i <= r; ++i) for (int j = 1; j <= c; ++j)
        b[i+1][j] = b[i-1][j] ^ b[i][j-1] ^ b[i][j] ^ b[i][j+1] ^ a[i][j];
    for (int i = 1; i <= r; ++i) for (int j = 1; j <= c; ++j) {
        printf("%d%c", b[i][j], " \n"[j == c]);
    }
}
```

对于 Z2 下的 01 矩阵，每次可以选择一个点，将这个点所在行列上的一共(r+c-1)个点的值改变(xor 1)。

1. 给定一个初始状态，问是否可以通过若干次操作变为 0 矩阵?
   a) 若 r 和 c 均为偶数，任何一个位置为 1 的情况，我们可以改变一次这个位置所在行列的所有点(r+c-1 个)，结果是这个位置变为 0，其他不变；直接 YES
   b) 若 r 和 c 不均为偶数，则分成偶数行+偶数列的一个矩阵，和一行(一列)，那个偶数阶矩阵用上面的方法完成后，如果剩下的部分全 0 或者全 1 –> YES，否则-> NO
2. 问有多少种初始状态通过若干次操作可以变为 0 矩阵?
   a) 若 r 和 c 均为偶数 –> 1 << rc
   b) 若 r 和 c 不均为偶数 -> 1 << (r – odd(r))(c – odd(c)) + 1

证明详见 http://blog.renren.com/blog/60525895/798216990

给你一个向量组 x[]，反复判断一个向量 v 是否属于这个向量组的闭包(可以有这个向量组线性表出)

方法：【搞基】求出这个向量组的一组基底(化成上三角)，然后看 v 是否可以由这组基底线性表出(从上到下每个向量是否使用时唯一确定的)；复杂度 O(r * c * Rank) – O(c * Rank)，其中 Rank < min(r, c)

```
//上次使用 2012.9.24
int readLine(){
    static char s[35]; scanf("%s", s);
    int v = 0, n = strlen(s);
    for (int j = 0; j < n; ++j) v = v << 1 | (s[j] - '0');
```

```
	return v;
}

void writeLine(int v, int c) {
	for (int j = c - 1; j >= 0; --j) putchar( testbit(v, j) + '0' );
	putchar('\n');
}

struct mat {
	int x[maxk], r, c;
	void closure() {
		for (int k = 0; k < r; ++k) {
			int maxi = k;
			for (int i = k + 1; i < r; ++i) if ( x[i] > x[maxi] ) maxi = i;
			swap( x[k], x[maxi] );
			if ( x[k] == 0 ) {
				r = k; break;
			}
			for ( int i = k + 1; i < r; ++i)
				if ( (x[i] ^ x[k]) < x[i] ) x[i] ^= x[k];
		}
	}
	void read(int r, int c){
		this->r = r; this->c = c;
		for (int i = 0; i < r; ++i) x[i] = readLine();
	}
	bool query(int v) const {
		for (int i = 0; i < r; ++i)
			if ( (v ^ x[i]) < v ) v ^= x[i];
		return v == 0;
	}
} m;

void solve(int c, int r, int q) {
	m.read(r, c);
	m.closure();
	while (q--) {
		int v = readLine(), ans = -1;
		if ( m.query(v) ) ans = v;
		if (ans != -1) {
			writeLine(ans, c); continue;
		}
		for (int i = 0; i < c; ++i) {
			flipbit(v, i);
```

```
            if ( m.query(v) ) ans = ans == -1 ? v : min(ans, v);
            flipbit(v, i);
        }
        if (ans != -1) {
            writeLine(ans, c); continue;
        }
        for (int i = 0; i < c; ++i) {
            flipbit(v, i);
            for (int j = i + 1; j < c; ++j) {
                flipbit(v, j);
                if ( m.query(v) ) ans = ans == -1 ? v : min(ans, v);
                flipbit(v, j);
            }
            flipbit(v, i);
        }
        if (ans != -1) {
            writeLine(ans, c); continue;
        }
        for (int i = 0; i < c; ++i) {
            flipbit(v, i);
            for (int j = i + 1; j < c; ++j) {
                flipbit(v, j);
                for (int k = j + 1; k < c; ++k) {
                    flipbit(v, k);
                    if ( m.query(v) ) ans = ans == -1 ? v : min(ans, v);
                    flipbit(v, k);
                }
                flipbit(v, j);
            }
            flipbit(v, i);
        }
        if (ans != -1) {
            writeLine(ans, c); continue;
        }
        puts("NA");
    }
}

int main(){ for (int n, k, m; cin >> n >> k >> m; solve(n, k, m) ); } //zoj 3636 Decode
```

线性规划

线性规划对偶问题

$$\min f = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le (=,\ge) b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le (=,\ge) b_2 \\ \quad \vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \le (=,\ge) b_m \\ x_j \ge 0 (\le 0, \text{或符号不限})\ j = 1 \sim n \end{cases}$$

$$\max z = b_1y_1 + b_2y_2 + \cdots + b_my_m$$

$$\begin{cases} a_{11}y_1 + a_{21}y_2 + \cdots + a_{m1}y_m \le (\ge,=) c_1 \\ a_{12}y_1 + a_{22}y_2 + \cdots + a_{m2}y_m \le (\ge,=) c_2 \\ \quad \vdots \qquad\quad \vdots \qquad\qquad\quad \vdots \\ a_{1n}y_1 + a_{2n}y_2 + \cdots + a_{mn}y_m \le (\ge,=) c_n \\ y_j \le 0 (\text{符号不限}, \text{或} \ge 0)\ i = 1 \sim m \end{cases}$$

对偶问题对应表

原问题（对偶问题）	对偶问题（原问题）
目标函数min	目标函数max
约束条件：m个 第i个约束类型为"≥" 第i个约束类型为"≤" 第i个约束类型为"＝"	变量数：m个 第i个变量≥0 第i个变量≤0 第i个变量是自由变量
变量数：n个 第j个变量≤0 第j个变量≥0 第j个变量是自由变量	约束条件：n个 第j个约束类型为"≥" 第j个约束类型为"≤" 第j个约束类型为"＝"

(LP) \ (DP)	有最优解	无界解	无可行解
有最优解	√	×	×
无界解	×	×	√
无可行解	×	√	√

其他

图论结论

设 G 是一个连通一般图，则 G 中存在闭欧拉迹，当且仅当 G 中每个顶点的度都是偶数；G 中村来一条开欧拉链，当且仅当 G 中恰好有两个奇度顶点 u 和 v（此外，G 中任何一条开欧拉迹连接 u 和 v）。

设 G 是一个连通一般图，并设 G 中奇度顶点的个数 m>0（显然 m 一定是偶数），则 G 的边可以被划分为 m/2 个开迹，但是不能被划分为少于 m/2 个开迹。

在一个 n 阶简单图中，若每对不邻接顶点的度数之和至少是 n-1，则图中存在 Hamilton 路径；若每对不邻接顶点的度数之和至少为 n，则图中存在 Hamilton 圈

骨牌放置问题

一张 m 行 n 列棋盘有一个 b-牌的完美覆盖，当且仅当 b|m 或 b|n。
m×n(m, n ≥ 2)矩形可用 L 骨牌完全覆盖，当且仅当 mn 可以被 8 整除。
如果一个 m×n(m, n ≥ 2)矩形不能被 L 骨牌完全覆盖，最少空格数为 STEP：

1. 若 mn 能被 4 整除，但是不能被 8 整除，STEP = 4
2. 若 mn 不能被 4 整除，STEP = mn % 4

求 s!的最后非 0 位

```
int lastdigit(char s[], int n){
	//求 s!的最后非 0 位，已经通过 zju1222
	//时间复杂度 nlogn, n = strlen(s)
	static int hash0[] = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8}; //one digit hash
	static int hash [] = {6, 6, 2, 6, 4, 4, 4, 8, 4, 6}; //multi-digit hash
	if (n == 1) return hash0[s[0] - '0'];
	int ret = hash[s[n-1] - '0'], t = 0, cnt5 = 0;
	for (int i = 0; i < n; i++){
		t = t * 10 + s[i] - '0';
		s[i] = t / 5 + '0'; t %= 5;
		cnt5 = cnt5 * 10 + s[i] - '0';
	}
	if (s[0] == '0') s++, n--;
	ret = ret * lastdigit(s, n) % 10;
	for (cnt5 &= 3; cnt5--; ) ret = ret * 8 % 10; //注意一定要用 &= 3， 不要用 %= 4
	return ret;
}
```

马(Knight)从(0,0)到(x,y) (0 <= x <= y)的最少步数

$$f = \begin{cases} 3 & x = 0, y = 1 \\ 4 & x = y = 2 \\ max\left\{\left\lfloor\frac{y+1}{2}\right\rfloor, \left\lfloor\frac{x+y+2}{3}\right\rfloor\right\} + t & other \end{cases}$$

$t \in \{0, 1\}$用来确保 f 和 $x + y$ 的奇偶性相同
代码

```
int calc(int x, int y){
	x = abs(x); y = abs(y); if (x > y) swap(x, y); //ensure 0 <= x <= y
	if (x == 0 && y == 1) return 3;
	if (x == 2 && y == 2) return 4;
	int r = max((y + 1) / 2, (x + y + 2) / 3);
	if ((r - x - y) & 1) r++;
```

```
    return r;
}
```

将一块蛋糕或者平均分给 x_1 人，或者平均分给 x_2 人，…，或者平均分给 x_n 人（来多少个人不确定），问最少需要切成几块(每块大小可以不同)

res = $x_1 + x_2 + \cdots + x_n -(x_1, x_2) - ([x_1, x_2], x_3) - ([x_1, x_2, x_3], x_4) - \cdots - ([x_1, x_2, \cdots, x_{n-1}], x_n)$

其中$([x_1, x_2, \cdots, x_{k-1}], x_k) = ([([x_1, x_2, \cdots, x_{k-2}], x_k), (x_{k-1}, x_k)], x_k)$

一个简单的切法：不妨设蛋糕为矩形，沿着矩形的一边，先平均切成 x_1 份，再沿着同一边平均切成 x_2 份，…，以此类推。如果某位置已经切过就不再切了。也就是说，在坐标为 $k*(L / x_i)$的位置下刀。

Gray 码

二进制码 B = $b_{n-1}b_{n-2}\ldots b_1b_0$

Gray 码 G=$g_{n-1}g_{n-2}\ldots g_1g_0$

$$b_i = g_{n-1} \oplus g_{n-2} \oplus \ldots \oplus g_{i+1} \oplus g_i = \begin{cases} b_{i+1} \oplus g_i & i \neq n-1 \\ g_i & i = n-1 \end{cases}$$

$$g_i = \begin{cases} b_{i+1} \oplus b_i & i \neq n-1 \\ b_i & i = n-1 \end{cases}$$

G = B | (B >> 1)

n 阶 Gray 码相当于在 n 维立方体上的 Hamilton 回路

Gray 码和 Hanoi 塔问题等价，Gray 码改变的是第几个数，Hanoi 塔就该移动第几个盘子。

Hanoi 问题

只能移动 A->B，B->C，C->A

设 i 个盘子，要 A->B，最少移动次数 F1(i); 要 A->C 最少移动次数 F2(i)，则

$F2(i) = 2F_2(i-1) + F_1(i-1) + 2$[i]

$F1(i) = 2F_2(i-1) + 1$[i]

最多移动次数(不能出现重复状态)

```
void Move(int n, int s, int m, int t){
    //T[n] = 3T[n-1] + 2; T[1] = 2 -> T[n] = 3 ^ n - 1
    if (n == 1){
```

```
        //    1 from s to m
        //    1 from m to t
    } else {
        Move(n-1, s, m, t);
        //    n from s to m
        Move(n-1, t, m, s);
        //    n from m to t
        Move(n-1, s, m, t);
    }
}
```

4 柱 hanoi 问题

$$\mathrm{F}^4(n,A,B,C,D)=\begin{cases}\mathrm{F}^4(n-R(n),A,C,D,B)\\ \mathrm{F}^3(R(n),A,C,D)\\ \mathrm{F}^4(n-R(n),B,A,C,D)\end{cases}$$

其中

$$R(n)=\left\lfloor\frac{\sqrt{8n+1}-1}{2}\right\rfloor$$

最少的步数

$$\mathrm{F}^4(n)=\left[n-\frac{R^2(n)-R(n)+2}{2}\right]\cdot 2^{R(n)}+1$$

算法摘自论文《A Non-recursive Algorithm for 4-Peg Hanoi Tower》(实际大数据不保证正确性)

```
long long hanoi4(int n){ //已经通过 hdu Gardon-DYGG Contest 2  某题
    int r = (int)floor((sqrt(8.0 * n + 1) - 1) * 0.5 + 1e-10);
    return (n - (r * r - r + 2) / 2) * (1LL << r) + 1;
}
```

Farey 序列的生成

Fn: 分母<=n 的 Farey 数，|Fn| 约等于 0.304*N^2

```
int n;
void make_farey(int x1, int y1, int x2, int y2){
    if (x1 + x2 <= n && y1 + y2 <= n){
        make_farey(x1, y1, x1 + x2, y1 + y2);
        printf("%d/%d\n", x1 + x2, y1 + y2);
        make_farey(x1 + x2, y1 + y2, x2, y2);
    }
}

int main(){
    while ( cin >> n ) make_farey(0, 1, 1, 1);
```

}

构造 n 阶幻方（魔方）

除二阶魔方不存在外，任何 n>=1 都可以构造一个 n 阶魔方

1. n = 2k + 1

(1) 1 放在第一行中间位置上

(2) 下一个数放在当前位置的“右上角”(循环移动)

(3) 若下一个数要放的位置上已经有了数字，则下一个数放在当前位置的“下面”(循环移动)

三阶幻方：

8	1	6
3	5	7
4	9	2

2. n = 4k + 2

(1) 把方阵按照左上、右上、左下、右下分为 A, B, C, D 四个区域，这样每个区域肯定是奇数阶

A	B
C	D

(2) 用楼梯法，依次在 A，D, B, C 区域按奇数阶幻方的填法填数

17	24	1	8	15	67	74	51	58	65
23	5	7	14	16	73	55	57	64	66
4	6	13	20	22	54	56	63	70	72
10	12	19	21	3	60	62	69	71	53
11	18	25	2	9	61	68	75	52	59
92	99	76	83	90	42	49	26	33	40
98	80	82	89	91	48	30	32	39	41
79	81	88	95	97	29	31	38	45	47
85	87	94	96	78	35	37	44	46	28
86	93	100	77	84	36	43	50	27	34

(3) 在 A 区域的中间行，中间列开始，从左到右，标出 k 格；A 区域其他行则标出最左边的 k 格。将标记的格子，和 C 区域相应的位置的数互换

<92>	<99>	1	8	15	67	74	51	58	65
<98>	<80>	7	14	16	73	55	57	64	66
4	6	<88>	<95>	22	54	56	63	70	72
<85>	<87>	19	21	3	60	62	69	71	53
<86>	<93>	25	2	9	61	68	75	52	59
<17>	<24>	76	83	90	42	49	26	33	40
<23>	<5>	82	89	91	48	30	32	39	41

79	81	<13>	<20>	97	29	31	38	45	47
<10>	<12>	94	96	78	35	37	44	46	28
<11>	<18>	100	77	84	36	43	50	27	34

(4) 在 B 区域中间列开始，从右向左标出 k-1 列。将标记的格子，和 D 区域相应的位置的数互换

<92>	<99>	1	8	15	67	74	<26>	58	65
<98>	<80>	7	14	16	73	55	<32>	64	66
4	6	<88>	<95>	22	54	56	<38>	70	72
<85>	<87>	19	21	3	60	62	<44>	71	53
<86>	<93>	25	2	9	61	68	<50>	52	59
<17>	<24>	76	83	90	42	49	<51>	33	40
<23>	<5>	82	89	91	48	30	<57>	39	41
79	81	<13>	<20>	97	29	31	<63>	45	47
<10>	<12>	94	96	78	35	37	<69>	46	28
<11>	<18>	100	77	84	36	43	<75>	27	34

3. n = 4k

(1) 把幻方划分为 k*k 个区域，先把数字按照顺序填写

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	57	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

(2) 把每个区域对角线上的数字换成其互补数字，即 x -> n * n + 1 – x

<64>	2	3	<61>	<60>	6	7	<57>
9	<55>	<54>	12	13	<51>	<50>	16
17	<47>	<46>	20	21	<43>	<42>	24
<40>	26	27	<37>	<36>	30	31	<33>
<32>	34	35	<29>	<28>	38	39	<25>
41	<23>	<22>	44	45	<19>	<18>	48
49	<15>	<14>	52	53	<11>	<10>	56
<8>	58	59	<5>	<4>	62	63	<1>

构造 n 阶反幻方

定理：若 n (n >= 3)阶方阵为 A = [aij]，则

$$a_{ij} = \begin{cases} (i-1)(n-1)+j & i = 1..n, j = 1..n-1 \\ n(n-1)+i & i = 1..n, j = n \end{cases}$$

是一个 n 阶反幻方。

Joseph 约瑟夫问题 – 数学解法

令 f[n]表示“n 个人玩游戏报 m 退出”的胜利者的编号(下标从 0 开始)，则：
f[1] = 0; f[n] = (f[n-1] + m) % n; //已经通过 cugb1056
注：当 m<<n 的时候，可以用等差数列的性质，加速运算.
当 m = 2 时，f[n] = {n 循环左移 1 位}

扫雷机器人 2011(robot2011) - SHTSC2011

地雷排成一排，输入每个地雷在数轴上的坐标、爆炸范围。问随机引爆的前提下，引爆完所需要的引爆次数的期望。
标准解答：引爆总次数的期望，即为每个炸弹被引爆的次数的期望的和，而每个炸弹要么被引爆（次数为 1），要么不被引爆（次数为 0），所以引爆次数的期望等于每个炸弹被引爆的概率之和。假设出了 i 之外有 p[i]个炸弹可以直接或者间接的引爆 i，那么 i 被引爆的概率就是 1/(p[i]+1)。这个可以在平方时间内朴素计算完成。不过，标程的解法是 O(n)的。(TODO：怎么做到的!!!!!!)

最大不能构造数

题意：给出 n 个正整数，求这 n 个数任意相加(可重复)所不构造的最大的数(设 $\gcd(a_1, \ldots, a_n) = 1$)。
解法：设 $a_1 \le a_2 \le \ldots \le a_n$，所有由 $a_1, \ldots, a_n$ 构成的元素集合为 Ω，即 $\Omega = \{c_1a_1 + \ldots + c_na_n \mid c_i \ge 0\}$，将这些数按照 mod a_1 划分等价类 $R_0 \ldots R_{a1-1}$，每个等价类 R_j 求出一个最小的代表元 r_j

$$r_j = min\{r \in R_j \mid (r' \ge r \wedge r' \in R_j) \rightarrow r \in \Omega\}$$

则最终答案为 $\max\{r_j\} - a_1$
对于 r_j 的求法，可以以每个 R_j 做为顶点，$a_2 \% a_1, \ldots, a_n \% a_1$ 为带权边建图（$|V| = a_1$, $|E| = n|V|$），求 R_0 到所有点的最短路即可。

用天平称 k 次最多可确定多少个坏球

	k=0	k>=1
不用确定是轻是重	1	$(3^k - 1) / 2$
需要确定是轻是重	0	$(3^k - 3) / 2$

罗马数字&阿拉伯数字

```
int toArab(char *roman){
    int dict[300] = {0};
    dict['I'] = 1; dict['V'] = 5; dict['X'] = 10; dict['L'] = 50;
```

```
    dict['C'] = 100; dict['D'] = 500; dict['M'] = 1000; dict[0] = 0;

    int res = 0;
    for (int i = 0; roman[i]; i++){
        if (dict[ roman[i] ] < dict[ roman[i+1] ])
            res -= dict[ roman[i] ];
        else
            res += dict[ roman[i] ];
    }
    return res;
}

void printRoman(int n){
    static const char *tbl[4][10] = {
        {"", "I", "II", "III", "IV", "V", "VI", "VII", "VIII", "IX"},
        {"", "X", "XX", "XXX", "XL", "L", "LX", "LXX", "LXXX", "XC"},
        {"", "C", "CC", "CCC", "CD", "D", "DC", "DCC", "DCCC", "CM"},
        {"", "M", "MM", "MMM", "MMMM"},
    };
    printf("%s", tbl[3][n/1000]); n%=1000;
    printf("%s", tbl[2][n/100]);  n%=100;
    printf("%s", tbl[1][n/10]);   n%=10;
    printf("%s", tbl[0][n]);
}
```

4 个数算 24 点

```
//上次使用 2012.7.8
bool get24(vector<double> a) {
    static map< vector<double>, bool > tbl;

    sort( a.begin(), a.end() );
    if ( tbl.find(a) != tbl.end() ) return tbl[a];
    if ( a.size() == 1 ) {
        return tbl[a] = fabs( a[0] - 24 ) < 1e-8;
    } else {
        for (int i = 0; i < a.size(); ++i) {
            for (int j = i + 1; j < a.size(); ++j) {
                #define go(x, op, y) { \
                    vector<double> b; b.push_back(a[x] op a[y]);\
                    for (int k = 0; k < a.size(); ++k) if (k != i && k != j) b.push_back(a[k]); \
                    if ( get24(b) ) return tbl[a] = true; \
                }
                go(i, +, j); go(i, -, j); go(j, -, i);
```

```
                go(i, *, j); go(i, /, j); go(j, /, i);
            }
        }
        return tbl[a] = false;
    }
}
```

[1, 10^n]中,字符串 M 出现多少次(n <= 15, m <= 10^6)

```
const int maxn = 1000000 + 5;
const int maxm = 18;
const int lim = 5; // <= lim 位直接算, >lim 位公式算
LL powten[maxm];
LL len[maxm]; // len[k]: 长度<=k 的数有多长
char s[maxn]; int lenlim;
char tar[maxn]; int tlen;

void toString(int i, char *buf, int &n){
    for (n = 0;i; i /= 10) buf[n++] = i % 10;
    reverse(buf, buf + n);
}

void init(){
    powten[0] = 1;
    for (int i = 1; i < maxm; i++) powten[i] = powten[i-1] * 10;
    for (int i = 1; i < maxm; i++) len[i] = powten[i-1] * i * 9;
    for (int i = 2; i < maxm; i++) len[i] += len[i-1];
    lenlim = 0;
    for (int i = 1, j; i < powten[lim]; i++){
        toString(i, s + lenlim, j); lenlim += j;
    }
    s[lenlim] = 0;
}


LL h(int len, int m){
    LL ans = 0;
    for (int i = 0, v = 0; i < len; i++){
        if (i < tlen){
            v = v * 10 + s[i];
            if (i == tlen - 1 && v == m) ++ans;
        } else {
            v -= (s[i-tlen]) * powten[tlen-1];
            v = v * 10 + s[i];
```

```
			if (v == m) ++ans;
		}
	}
	return ans;
}

LL g(int n, int p){ //[0..p],[p+1..tlen-1]
	if (tar[p+1] == 0) return 0;
	LL ret = 0;
	for (int i = max(lim+1, tlen); i <= n; i++ )
		ret += powten[i-tlen];
	return ret;
}

LL f(int n){ //长度为 n 的数[10^(n-1), 10^n)中含有 tar 的个数
	LL ret = 0;
	for (int i = 0; i <= n-tlen; i++)
		ret += (i ? powten[i]-powten[i-1] : 1) * powten[n-tlen-i];
	return ret;
}

LL solve(int n, int m){
	LL ans = 0;
	if (0 == m){
		LL g[maxm] = {0, 1};
		for (int i = 2; i <= n; i++) g[i] = g[i-1] * 10 + powten[i-1];
		for (int i = 2; i <= n; i++) ans += g[i-1] * 9;
		return ans;
	}
	toString(m, tar, tlen);
	if (n <= lim){
		return h( len[n], m );
	} else {
		ans += h( lenlim, m );
		for (int i = 0; i < tlen-1; i++) // < tlen-1
			ans += g(n, i);
		for (int i = lim+1; i <= n; i++)
			ans += f(i);
		return ans;
	}
}

int main(){
	init();
```

```
    for (int n, m; cin >> n >> m && n + m; ){
        cout << solve(n, m) << endl;
    }
}
```

累计 1..n 中每个数字的出现次数

```
void cntdigit(int n, int cnt[], int t = 1){ //已经通过 PKU2282
    //累计 1..n 中每个数字的出现次数，并加到 cnt 数组中
    if (n <= 0) return;

    int a = n / 10, b = n % 10;
    n = a;

    for (int i = 0; i <= b; i++) cnt[i] += t;
    for (;a; a /= 10) cnt[a % 10] += (b + 1) * t;

    for(int i = 0; i < 10; i++) cnt[i] += n * t;
    cnt[0] -= t;

    cntdigit(n-1, cnt, t * 10);
}
```