

## 目录

|   |    |
|---|----|
| 概率知识 .....                                | 6  |
| 数学公式 .....                                | 8  |
| 数学常数 .....                                | 9  |
| 一元方程求根公式 .....                            | 10 |
| 二次方程求根公式 .....                            | 10 |
| 三次方程求根公式 - 卡尔丹公式 .....                    | 10 |
| 四次方程求根公式 .....                            | 10 |
| 代码 .....                                  | 11 |
| 求和公式 .....                                | 12 |
| 前 n 个正整数的 k 次方的和(一般 k 次多项式的和) .....       | 12 |
| 三角公式 Trigonometric Function Formula ..... | 14 |
| Product To Sum .....                      | 14 |
| Sum To Product .....                      | 14 |
| Centers of Triangls .....                 | 14 |
| 矩阵乘法 .....                                | 14 |
| 分数的最大公约数和最小公倍数 .....                      | 14 |
| Raney 引理 .....                            | 14 |
| 组合游戏 .....                                | 15 |
| Wythoff Game .....                        | 15 |
| Fibonacci Nim .....                       | 16 |
| 楼天城 Nim 游戏 .....                          | 17 |
| 常规 Nim 游戏 .....                           | 17 |
| 阶梯 Nim 游戏 .....                           | 17 |
| SG 函数 .....                               | 18 |
| Misere 规则下的组合游戏 (SJ 定理) .....             | 18 |
| 翻硬币游戏 .....                               | 18 |
| 无向图删边游戏 .....                             | 18 |
| 树的删边游戏 .....                              | 18 |

|                               |    |
|-------------------------------|----|
| 无向图的删边游戏 .....                | 19 |
| 组合计数 .....                    | 19 |
| M 个球放入 N 个盒子 .....            | 19 |
| 正整数的无序分拆数 .....               | 20 |
| 性质(根据分拆的 Ferrers 图可以证明) ..... | 20 |
| 第二类 Stirling 数 .....          | 20 |
| $s(n, k) \bmod 2$ .....       | 21 |
| Fibonacci 数 .....             | 21 |
| 组合数 .....                     | 22 |
| Catalan 数 .....               | 23 |
| Fuss-Catalan 数 .....          | 23 |
| 拟 Catalan 数 .....             | 24 |
| 错排数 .....                     | 24 |
| 有禁止模式的排列数 .....               | 24 |
| 大 Schröder 数 .....            | 24 |
| 小 Schröder 数 .....            | 25 |
| 第一类欧拉数 .....                  | 25 |
| 第二类欧拉数 .....                  | 26 |
| 生成树计数 - Cayley 定理 .....       | 26 |
| 区域划分计数 .....                  | 26 |
| n 对夫妻圆排列计数 .....              | 27 |
| Sperner 定理 .....              | 27 |
| 圆域染色计数 .....                  | 27 |
| Pólya 计数法 .....               | 28 |
| 其他 .....                      | 28 |
| 计算几何 .....                    | 29 |
| 几何公式 .....                    | 29 |
| 费马点 .....                     | 29 |
| Bretschneider's 公式 .....      | 31 |
| 其他 .....                      | 32 |

|   |    |
|---|----|
| 基本代码 .....  | 32 |
| 绕轴旋转、三维空间点对直线垂线的垂足 .....                                  | 35 |
| 凸包[Graham] – $O(n \log n)$ .....                          | 36 |
| 最远点对 .....  | 37 |
| 最近点对问题 – 分治 – $O(n \log n)$ .....                         | 38 |
| 周长最小三角形 – 分治 – $O(n \log n)$ .....                        | 39 |
| 面积最大多边形 .....   | 42 |
| 点集的最小覆盖圆 – 期望 $O(n)$ .....                                | 45 |
| 外接圆最大三角形 – 枚举 1 点, 极角排序 - $O(n^2 \log n)$ .....           | 46 |
| 判断两个正交矩形是否有公共点 .....                                      | 47 |
| 数论 .....  | 48 |
| 欧几里德算法 .....  | 48 |
| 扩展欧几里德 .....  | 48 |
| 扩展欧几里德-递归 .....   | 49 |
| 扩展欧几里德-迭代 .....   | 49 |
| 四则运算 (乘除指对) .....   | 49 |
| 计算 $f[i] = a \wedge f[i-1] \bmod n$ (super_pow_mod) ..... | 52 |
| 模线性方程 .....   | 53 |
| 广义中国剩余定理 .....  | 53 |
| 素数 .....  | 54 |
| 素数的判断 .....   | 55 |
| 筛法求素数 $O(n)$ .....  | 57 |
| 反素数 .....   | 58 |
| 因式分解 .....  | 58 |
| 枚举约数 .....  | 59 |
| 原根定理 .....  | 59 |
| 积性函数 .....  | 60 |
| 欧拉函数 .....  | 60 |
| Jordan's totient function 约旦欧拉函数 .....                    | 63 |
| 其他积性函数 .....  | 63 |

|  |    |
|--|----|
| The Determinants of GCD matrices.....  | 64 |
| 勒让德.....   | 64 |
| 勒让德函数.....   | 64 |
| 勒让德多项式.....  | 65 |
| 勒让德多项式的性质 .....  | 65 |
| 二次剩余 .....   | 65 |
| 最小二乘法 .....  | 66 |
| 排列组合 .....   | 67 |
| 组合数 $C(m, n)$ .....  | 67 |
| 组合 .....   | 67 |
| 伽马函数、阶乘.....   | 67 |
| 莫比乌斯函数.....  | 68 |
| 莫比乌斯反演定理.....  | 68 |
| Squarefree 数.....  | 69 |
| 求 1..a 和 1..b 中互质的数(最大公约数为 k 的数)的对数——利用 squarefree 数进行容斥 .....                     | 69 |
| 连分数 (Continued Fractions of Rationals) .....                                       | 70 |
| 典型例题 .....   | 70 |
| DESCRIPTION: find $a + bn \bmod m$ , where $0 < b - a < 1 + 2a$ and n is even..... | 70 |
| Given n and k, find $\max\{ i \mid k^i \text{ divides } n! \}$ – java.....         | 70 |
| 多项式运算 .....  | 72 |
| 多项式乘法 - FFT - O(nlogn).....  | 72 |
| 多项式除法 .....  | 74 |
| 数据类型 .....   | 75 |
| 高精度正整数.....  | 75 |
| 高精度实数 .....  | 81 |
| 分数.....  | 84 |
| 大实数 .....  | 84 |
| 矩阵运算 .....   | 86 |
| 矩阵的 LU(Doolittle)分解.....   | 96 |

|  |     |
|--|-----|
| 追赶法求解三对角线性方程组 - O(n) .....   | 97  |
| 拟对角线性方程组的求解 - O(n) .....   | 98  |
| 行列式取模 - det_mod .....  | 99  |
| 求 $E+A+A^2+\dots+A^k$ .....  | 99  |
| 线性规划 .....   | 100 |
| 线性规划对偶问题 .....   | 100 |
| 其他 .....   | 100 |
| 图论结论 .....   | 100 |
| 骨牌放置问题 .....   | 101 |
| 求 $s!$ 的最后非 0 位 .....  | 101 |
| 马(Knight)从(0,0)到(x,y) ( $0 \leq x \leq y$ )的最少步数 .....   | 101 |
| 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) ..... | 102 |
| Gray 码 .....   | 102 |
| Hanoi 问题 .....   | 102 |
| 最多移动次数(不能出现重复状态) .....   | 102 |
| 4 柱 hanoi 问题 .....   | 103 |
| Farey 序列的生成 .....  | 103 |
| 构造 $n$ 阶幻方 (魔方) .....  | 103 |
| 构造 $n$ 阶反幻方 .....  | 105 |
| 约瑟夫问题 - 数学解法 .....   | 105 |
| 扫雷机器人 2011(robot2011) - SHTSC2011 .....  | 106 |
| 最大不能构造数 .....  | 106 |
| 用天平称 $k$ 次最多可确定多少个坏球 .....   | 106 |
| 累计 $1..n$ 中每个数字的出现次数 .....   | 106 |
| 统计 $1..n$ 的数中含 49 的数的个数 .....  | 107 |

# 概率知识

条件 概率:  $P(A|B) = \frac{P(AB)}{P(B)}$

全概率公式:  $P(A) = \sum_{i=1}^n P(B_i)P(A|B_i)$

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

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

注:

- 求最大值点方法  $f(i) \geq f(i+1), f(i) \geq f(i-1)$
- $D\xi = E(\xi^2) - (E\xi)^2$
- 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。
- Pascal 分布**  $\{f(k; r, p)\}$  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  $g(k; p) = f(k; 1, p)$

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

**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

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

**广义 Bernoulli 实验:** 假定实验有  $r$  种可能的结果  $A_1, A_2, \dots, A_r$ , 并且  $p_i = p(A_i) > 0, \sum p_i = 1$ ,  $i = 1, 2, \dots, r$ , 重复  $n$  次,  $A_i$  出现  $k_i$  次 ( $\sum k_i = n, i = 1, 2, \dots, r$ ) 的概率为:

$$\binom{k_1 \dots k_r}{n} p_1^{k_1} \dots p_r^{k_r}$$

$A_i$  在  $A_j$  之前出现的概率

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

**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  $\alpha$  粒子数  $\xi$  服从  $P(\lambda)$  分布, 每个放射出的  $\alpha$  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  $\alpha$  粒子数服从  $P(\lambda p)$  分布。  
证明:

$$\begin{aligned} P(\eta = k) &= \sum_{n=k}^{\infty} P(\xi = n)P(\eta = k | \xi = n) \\ &= \sum_{n=k}^{\infty} p(n; \lambda) b(k; n, p) = \sum_{n=k}^{\infty} \frac{\lambda^n}{n!} e^{-\lambda} C_n^k p^k q^{n-k} = \sum_{n=k}^{\infty} \frac{(\lambda q)^{n-k}}{(n-k)!} \cdot \frac{1}{k!} e^{-\lambda} (\lambda p)^k \\ &= \frac{1}{k!} (\lambda p)^k e^{-\lambda p} \end{aligned}$$

**Poisson 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  $\lambda > 0$ , 使得对一切  $t > 0$  有  $P_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$

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

**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

$$P(i) = \begin{cases} \frac{i}{a} & p = q \\ \frac{1 - \left(\frac{q}{p}\right)^i}{1 - \left(\frac{q}{p}\right)^a} & p \neq q \end{cases}$$

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  $p + q \neq 1$ , 则  $P(i) = qP(i-1) + pP(i+1) + (1-p-q)P(i)$ , 只需令  $p' = p/(p+q), q' = q/(p+q)$ , 则  $P(i) = q'P(i-1) + p'P(i+1)$ , 可得到相同结论。

注：也可用  $O(n)$  的方法三对角线性方程组。

## 数学公式

$n$  的所有因子之和  $(1 + p_1 + \dots + p_1^{r_1})(1 + p_2 + \dots + p_2^{r_2}) \dots (1 + p_k + \dots + p_k^{r_k})$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots$$

### Lucas 定理

对于  $n, m \geq 0, p - \text{prime}$

$$n = n_k p^k + \dots + n_1 p + n_0$$

$$m = m_k p^k + \dots + m_1 p + m_0$$

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

应用： $C_m^p \equiv (m \text{ div } p) \bmod p$

设  $y \geq 1$ ,  $\sum_{1 \leq n \leq y} \frac{1}{n} = \ln y + \gamma + \Delta(y)$ ,  $|\Delta(y)| \leq \frac{1}{y}$ , 欧拉常数  $\gamma = 0.5772$

设  $x \geq 2$ ,  $\sum_{p \leq x} \frac{1}{p} = \ln \ln x + A + r(x)$ ,

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

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  $r_1, \dots, r_{p-1}$  是模  $m$  的既约剩余系, 我们有

$$\prod_{(r,m)=1} (r = r_1 \dots r_c) \equiv \begin{cases} -1 & m = 1, 2, 4, p^\alpha, 2p^\alpha, p - \text{positive odd prime} \pmod{m} \\ 1 & \text{else} \end{cases}$$

特别地

$$(p-1)! \equiv -1 \pmod{p}$$

**Pick 定理:** 在一个平面直角坐标系内, 如果一个多边形的顶点全都在格点上, 那么这个图形的面积恰好就等于边界上经过的格点数的一半加上内部所含格点数再减一.

**Pick Theorem:** 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.

对任意正整数  $m, n$ ,  $m \times n$  的棋盘的对角线恰好穿过  $m+n-\gcd(m, n)$  个格子。

[拓扑学] 欧拉公式:  $|V| - |E| + |F| = 2$

**Stirling 近似公式:**

$$\begin{aligned} n! &= \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(1 + \Theta\left(\frac{1}{n}\right)\right) > \left(\frac{n}{e}\right)^n \\ n! &= \left(\frac{n}{e}\right)^n \sqrt{2\pi n} \left(1 + \frac{1}{12n} + \frac{1}{288n^2} + \Theta\left(\frac{1}{n^3}\right)\right) \end{aligned}$$

因此

$$\log(n!) = \Theta(n \log n)$$

$$H_n = 1 + \frac{1}{2} + \dots + \frac{1}{n} = \ln(n) + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} + O\left(\frac{1}{n^6}\right)$$

广义快速幂取模

$$A^x \equiv A^{x \bmod \phi(C) + \phi(C)} \pmod{C} \quad (x \geq \varphi(C))$$

## 数学常数

$$\gamma = \lim_{n \rightarrow \infty} (H_n - \ln n) = 0.577215664901532860606512090082402431042 \dots$$

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

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

## 一元方程求根公式

## 二次方程求根公式

$$ax^2 + bx + c = 0$$

$$x_1 = \frac{-b - \operatorname{sgn}(b)\sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{c}{ax_1}$$

## 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：

$$X_1 = \sqrt[3]{Y_1} + \sqrt[3]{Y_2}$$

$$X_2 = \omega \sqrt[3]{Y_1} + \omega^2 \sqrt[3]{Y_2}$$

$$X_3 = \omega^2 \sqrt[3]{Y_1} + \omega \sqrt[3]{Y_2}$$

$$\text{其中 } \omega = \frac{1}{2}(-1 + i\sqrt{3}), \quad Y_{1,2} = -\frac{q}{2} \pm \sqrt{\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3}$$

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

将方程两边同时除以最高次项的系数得

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

移项可得

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

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

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

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

```
void solve2(complex<double> a, complex<double> b, complex<double> c, complex<double> r[]){
    complex<double> 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, complex<double> r[]){ //已经通过 cugb1024
    complex<double> delta = q * q * (1.0 / 4.0) + p * p * p * (1.0 / 27.0); delta = sqrt(delta);
    complex<double> y1 = -q * 0.5 + delta, y2 = -q * 0.5 - delta;
    complex<double> 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){
        complex<double> rr[3]; double diff = 0;
        rr[0] = y1 + y2; rr[1] = w * y1 + w2 * y2; rr[2] = w2 * y1 + w * y2;
        for (int k = 0; k < 3; k++)
            diff += abs(rr[i] * rr[i] * rr[i] + p * rr[i] + q);
        if (diff < diffmax){
            diffmax = diff;
            for (int k = 0; k < 3; k++) r[k] = rr[k];
        }
    }
}

void solve3(double b, double c, double d, complex<double> r[]){
    double p = -b * b * (1.0 / 3.0) + c;
    double 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; complex<double> y[3];
```

```

B = -c; C = (b * d - 4.0 * e); D = -d * d - (b * b - 4.0 * c) * e; solve3(B, C, D, y);
complex<double> p = 0, q, tp, yy;
for (int i = 0; i < 3; i++){
    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, complex<double> r[]){
    solve4(b / a, c / a, d / a, e / a, r);
}

```

## 求和公式

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

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

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

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

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

## 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  $d_0, d_1, \dots, d_k, 0, 0, \dots, d_k \neq 0$ , 则

$$f(n) = d_0 C_n^0 + \dots + d_k C_n^k$$

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

**代码:** //O(k \* k)

```
const int maxk = 2000;
```

```

void get_Cnk_tbl_mod(int n, int k, int Cn[], int m){
    static int cnt[maxk];
    for (int j = 2; j <= k; j++)
        cnt[j] = 0;
    int r = 1;
    Cn[0] = 1;
    for (int i = 1; i <= k; i++){
        int x = n--;
        for (int j = 2; j <= k && j * j <= x; j++)
            if (x % j == 0) do x /= j, cnt[j]++;
            while (x % j == 0);
        if (x > 1)
            if (x <= k) cnt[x]++;
            else r = mul_mod(r, x, m);
        x = i;
        for (int j = 2; j <= k && j * j <= x; j++)
            if (x % j == 0) do x /= j, cnt[j]--;
            while (x % j == 0);
        if (x > 1) cnt[x]--;
        /* x <= i <= k */
        Cn[i] = r;
        for (int j = 2; j <= k; j++)
            if (cnt[j]) Cn[i] = mul_mod(Cn[i], pow_mod(j, cnt[j], m), m);
    }
}

int calc(int n, int k, int m){ //时间复杂度 O(k * k)
    static int f[maxk][maxk], Cn[maxk];
    f[0][0] = 0;
    for (int j = 1; j <= k; j++)
        f[0][j] = pow_mod(j, k, m);
    for (int i = 1; i <= k; i++)
        for (int j = 0; j <= k - i; j++)
            f[i][j] = (f[i-1][j+1] - f[i-1][j] + m) % m;
    get_Cnk_tbl_mod(n + 1, k + 1, Cn, m);
    LL ret = 0;
    for (int i = 0; i <= k; i++)
        ret += mul_mod(f[i][0], Cn[i+1], m);
    return ret % m;
}

int main(){//bnu4362
    int n, k;
    while (cin >> n >> k && n != -1 && k != -1)
        printf("%09d\n", calc(n, k, 1000000000));
}

```

# 三角公式 Trigonometric Function Formula

## Product To Sum

$$\begin{aligned}\sin A \cos B &= \frac{\sin(A+B) + \sin(A-B)}{2} \\ \cos A \sin B &= \frac{\sin(A+B) - \sin(A-B)}{2} \\ \sin A \sin B &= -\frac{\cos(A+B) - \cos(A-B)}{2} \\ \cos A \cos B &= \frac{\cos(A+B) + \cos(A-B)}{2}\end{aligned}$$

## Sum To Product

$$\begin{aligned}\sin A + \sin B &= 2 \sin \frac{A+B}{2} \cos \frac{A-B}{2} \\ \sin A - \sin B &= 2 \cos \frac{A+B}{2} \sin \frac{A-B}{2} \\ \cos A + \cos B &= 2 \cos \frac{A+B}{2} \cos \frac{A-B}{2} \\ \cos A - \cos B &= -2 \sin \frac{A+B}{2} \sin \frac{A-B}{2}\end{aligned}$$

## Centers of Triangls

### 内心 Incenter

$$I = \frac{|\vec{a}| \cdot A + |\vec{b}| \cdot B + |\vec{c}| \cdot C}{|\vec{a}| + |\vec{b}| + |\vec{c}|}$$

### 重心 Median

$$M = \frac{A + B + C}{2}$$

### 外心 Circumcenter – 详见点集的最小外接圆 垂心

$$\begin{aligned}H &= \frac{\alpha A + \beta B + \gamma C}{\alpha + \beta + \gamma} \\ \alpha &= (\vec{a} \cdot \vec{b})(\vec{a} \cdot \vec{c}) \\ \beta &= (\vec{b} \cdot \vec{c})(\vec{b} \cdot \vec{a}) \\ \gamma &= (\vec{c} \cdot \vec{a})(\vec{c} \cdot \vec{b})\end{aligned}$$

## 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  $\sum_{i=1}^n t^k k^i$  ( $k \leq 50$ )

## 分数的最大公约数和最小公倍数

$$\gcd(a_i / b_i) = \gcd(a_i) / \text{lcm}(b_i)$$

$$\text{lcm}(a_i / b_i) = \text{lcm}(a_i) / \gcd(b_i)$$

注意： $a_i / b_i$  一定是最简分数

## Raney 引理

设整数序列  $A = \{a_i\}$ ,  $i = 1, 2, \dots, n$ , 且部分和  $S_k = a_1 + a_2 + \dots + a_k$ , 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.

# 组合游戏

## Wythoff Game

**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

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

其中(0, 0)我们认为是奇异局势。

**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  $k = \left\lceil \frac{a}{\phi} \right\rceil = \left\lceil \frac{b}{\phi+1} \right\rceil = b - a$

**证明:**

1. Beatty 定理

如果存在无理数  $a, b$  满足

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

令

$$A = \{\lfloor ia \rfloor \mid i \in \mathbb{N}^*\}$$

$$B = \{\lfloor jb \rfloor \mid j \in \mathbb{N}^*\}$$

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

证明：(略)

2. 每个自然数都包含且仅包含在奇异局势数列

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

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  $F(i) \leq n < F(i+1)$ , 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  $F(j) \leq n' < F(j+1)$ ,  $n'' = n - F(i) - F(j) \geq 0$ , 若  $i$  和  $j$  是相邻的整数(即  $i = j+1$ ), 则  $n'' = n - F(i) - F(i-1) = n - F(i+1) \geq 0$ , 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  $i \neq j$ , 不妨设  $i > j$ , 则由不连续性,  $n = F(j) + \dots < F(j+1) \leq F(i)$  矛盾。

2. 设  $n$  的 Fibonacci 展开为  $F(n_1) + F(n_2) + \dots + F(n_k)$ , 其中  $n_1 > n_2 > \dots > n_k$ , 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  $F(m_1) + F(m_2) + \dots + F(m_t)$ , 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  $F(n_k) < F(m_1) + F(m_2) + \dots + F(m_t) + F(m_t - 1) < F(m_1 + 1)$  (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。

## 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 $(a_1, a_2, \dots, a_{2k})(a_1 \leq a_2 \leq \dots \leq a_{2k})$ 满足  $a_{2i-1} = a_{2i}$  ( $1 \leq i \leq k$ ), 即 $(a, a, b, b, c, c, \dots)$ 型分布。

**证明:**

1.  $(0, 0, 0, \dots)$  显然是必败态
2. 对于奇数堆石子 $(a_1, a_2, \dots, a_{2k}, a_{2k+1})(a_1 \leq a_2 \leq \dots \leq a_{2k} \leq a_{2k+1})$ , 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 $(a_2, a_2, a_4, a_4, \dots, a_{2k}, a_{2k})$
3. 对于偶数堆石子 $(a_1, a_2, \dots, a_{2k})(a_1 \leq a_2 \leq \dots \leq a_{2k})$ , 只要  $a_{2i-1} = a_{2i}$  ( $1 \leq i \leq k$ ) 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  $a_1$  颗, 即可转移到必败态 $(a_1, a_2, a_2, a_3, a_3, \dots, a_{2k-1}, a_{2k-1}, a_1)$
4. 对于偶数堆石子 $(a_1, a_2, \dots, a_{2k})(a_1 \leq a_2 \leq \dots \leq a_{2k})$ , 若  $a_{2i-1} = a_{2i}$  ( $1 \leq i \leq k$ ) 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 $-b_j$  颗石子( $j \neq i$ ), 将状态转移到 $(a_1 + b_1, a_2 + b_2, \dots, a_{2k} + b_{2k})$ , 可后手可以在第  $i'$  堆取出 $-b_{i'}$  颗石子放入第  $j'$  堆( $j \neq i, j \neq i'$ ), 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  $i' = i - 1$

## 常规 Nim 游戏

N-pos 先手必胜  $\leftrightarrow a_1 \oplus a_2 \oplus \dots \oplus a_{n-1} \oplus a_n \neq 0$

P-pos 后手必胜  $\leftrightarrow a_1 \oplus a_2 \oplus \dots \oplus a_{n-1} \oplus a_n = 0$

设从最高位开始恰好第  $k$  位的异或值不为 0, 即  $k = \max\{k \mid a_1^{(k)} \oplus a_2^{(k)} \oplus \dots \oplus a_n^{(k)} \neq 0\}$ , 则任取  $a_i^{(k)} = 1$ , 则游戏者只需要在第  $i$  堆石子中拿走  $a_i - (a_1 \oplus a_2 \oplus \dots \oplus a_{n-1} \oplus a_n \oplus a_i)$  颗石子, 即可恢复为 P 状态。

**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  $a_i \oplus \text{sum}$  个石子(若  $a_i \oplus \text{sum} < a_i$ )

## 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 $(x_1, x_2, \dots)$  为 P 状态当且仅当状态 $(x_1, x_3, x_5, \dots)$  对于 Nim 游戏是 P 状态。

## SG 函数

$SG(v) = \text{mex}\{ f(u) \mid \text{图中有一条从 } v \text{ 到 } u \text{ 的边}\}$

其中  $\text{mex}\{A\}$  表示不属于  $A$  的最大整数，即  $\text{mex}\{A\} = \min\{k \mid k \notin A \wedge k \in \mathbb{N}\}$

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.

## Misere 规则下的组合游戏 (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

## 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  $SG = \oplus(f[i] \mid \text{左边第 } i \text{ 个硬币正面朝上}, i=1..n)$

其中

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

即  $f[i] = f[i] == \text{highbit}(i) ? i : f[i - \text{highbit}(i)];$

## 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

一个无相联通图，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)：

- 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
- 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

## 组合计数

### 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    | $\sum_{k=1}^{\min\{N,M\}} S(M, k) = \frac{1}{N!} \sum_{k=1}^{\min\{N,M\}} \sum_{i=0}^k (-1)^i C_k^i (k-i)^M$<br>(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)  |
| 相同   | 不同    | 不空    | $C_{M-1}^{N-1}$ (挡板)  |
| 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)<br>$i \geq j$ 时， $g[i][j] = g[i][j-1] + g[i-j][j]$ ;<br>$i < j$ 时， $g[i][j] = g[i][j-1]$ ;<br>其中： $g[0][0..N] = 1, g[1..M][1] = 1$ |
| 相同   | 相同    | 不空    | $N \geq 2$ 时， $g(M, N) - g(M, N-1)$<br>$N = 1$ 时，1(也可以令 $g[1..M][0] = 0$ ，将两种情况合并)  |

- 将 M 个不同的球放入 M+个相同的盒子，盒子可以为空：(已经通过 hdu1028)

相当于求  $M$  元集合的所有划分数，记为  $B(M)$ ，成为 Bell 数

定义式:  $B(M) = \sum_{k=1}^M S(M, k)$

递推式:  $B(M+1) = \sum_{k=0}^M C_M^k B(k)$  —— 表示时间复杂度  $O(n^2)$

int 能存到  $B_{16}$ , long long 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   |

- 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  $s(p, p) = 1(p \geq 0)$ ,  $s(p, 0) = 0(p \geq 1)$

- 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

递推公式:  $S(i, j) = S(i-1, j-1) + j * S(i-1, j) \quad 2 \leq j \leq i-1; S(i, 1) = S(i, i) = 1$

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

| $i/j$ | 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 |  |  |  |  |  |  | 1 |
|---|---|--|--|--|--|--|--|---|

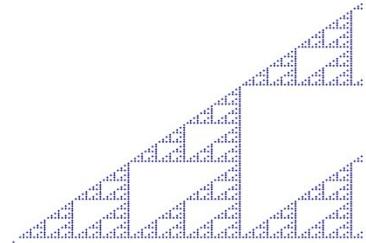
代码 - 求  $j! * S(i, j)$

```
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

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

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

## 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       |

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

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

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

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

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

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

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

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 Fibonacci 进制表示。

## 组合数

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

| 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 |

广义组合式定义

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

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

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

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

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

## 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  $a_1 a_2 \dots a_n$  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  $a \leq b$ ，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  $(p, q)$  ( $p \leq q$ )，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 hdu 某题

对于有些题目，可能情景还要求结果乘以  $p! * q!$

递推： $C_n = C_0 C_{n-1} + C_1 C_{n-2} + \dots + C_{n-1} C_0$

递推： $C_n = \frac{4n-2}{n+1} C_{n-1}$

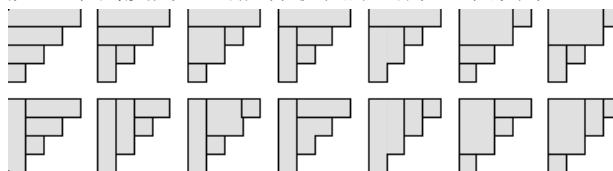
通项： $C_n = \frac{1}{n+1} C_{2n}^n$  (注意  $C(m, n)$  打表时别忘算  $C(0, 0) = 1$ )

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  $1, 2, 3, \dots, n$ 。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  $C_n$ 。下图为  $n = 4$  的情况：



## 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

$$C_n^* = n! C_{n-1} \quad (C_n \text{ 为 Catalan 数})$$

$$\text{递推: } C_n^* = (4n - 6)C_{n-1}^*$$

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

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

应用: 求  $a_1, a_2, \dots, a_n$  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,  $n \geq 7$  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  $\{1, 2, \dots, n\}$  的, 不出现  $12, 23, \dots, (n-1)n$  这些模式的全排列个数, 并规定  $Q_1 = 1$

则  $Q_n = D_n - D_{n-1}$

## 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  $a \geq b$ , 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  $(p, q)$  ( $p \geq q$ ), 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  $(p, q)$  ( $p$

$\geq q$ )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  $(p, q)$  ( $p \geq q$ ) 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  $(p, q)$  ( $p \geq q$ ) 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  $(p, q)$  ( $p \geq q$ ) 的 Schröder 路径条数。则

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

| $R(0)$ | $R(1)$ | $R(2)$ | $R(3)$ | $R(4)$ | $R(5)$ | $R(6)$ |
|--------|--------|--------|--------|--------|--------|--------|
| 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  $n \geq 1$ , 小 Schröder 数  $s(n)$  定义为符号序列  $a_1, a_2, \dots, a_n$  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:  $s(1) = 1, s(n+1) = R(n) / 2 (n \geq 1)$

递推式:  $(n+2)s(n+2) - 3(3n+1)s(n+1) + (n-1)s(n) = 0 (n \geq 1)$

| $s(1)$ | $s(2)$ | $s(3)$ | $s(4)$ | $s(5)$ | $s(6)$ | $s(7)$ |
|--------|--------|--------|--------|--------|--------|--------|
| 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  $\langle \binom{n}{k} \rangle$ , 或  $A(n, k)$

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

性质:  $\langle \binom{n}{k} \rangle = \langle \binom{n}{n-1-k} \rangle$

| $n/k$ | 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

$$\text{递推: } \langle\langle \frac{n}{k} \rangle\rangle = (2n - k - 1) \langle\langle \frac{n-1}{k-1} \rangle\rangle + (k+1) \langle\langle \frac{n-1}{k} \rangle\rangle$$

| 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

完全图  $K_n$  的生成树个数是  $n^{n-2}$ (彼此同构的子树多次计数);

有 n 个不同顶点的无根树的数目  $n^{n-2}$

## 区域划分计数

欧拉公式:  $V(\text{顶点数}) - E(\text{边数}) + F(\text{面数}) = 2$

定义  $h_n^{(k)} = C_n^0 + C_n^1 + \dots + C_n^k$

性质

$\Delta h_n^{(k)} = h_n^{(k-1)}$

$h_n^{(k)}$  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 M(n, r)，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

| $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ | $M_6$ | $M_7$ | $M_8$ | $M_9$ | $M_{10}$ |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|
| -     | 0     | 1     | 2     | 13    | 80    | 579   | 4738  | 43387 | 439792   |

可以证明

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

## 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 个有区别的扇形  $D_1 \dots D_n$ , 用 k 种颜色染色, 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:  $a_i = (k-2) * a_{i-1} + (k-1) * a_{i-2} (i \geq 4)$ ,  $a_1 = k$ ,  $a_2 = k * (k-1)$ ,  $a_3 = k * (k-1) * (k-2)$

( $a_i$  已经通过 cugb1249)

若圆形旋转  $\frac{2\pi}{t}$  对称, 则方法数  $b_n = \frac{a_n}{t}$

## Pòlya 计数法

**Pòlya 定理:** 设  $G = \{a_1, a_2, \dots, a_{|G|}\}$  是  $N = \{1, 2, \dots, N\}$  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:

$$S = (m^{c_1} + m^{c_2} + \dots + m^{c_{|G|}}) / |G|$$

### 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

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

## 其他

线段  $(0, 0) - (a, b)$  所经过的整点数  $\gcd(a, b) + 1$ ,

线段  $(0, 0) - (a, b)$  所覆盖的整数格子数  $a + b - \gcd(a, b)$

$n$  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  $(n-1)!$

定理:  $\{1, 2, \dots, n\}$  的  $r$ -组合  $a_1a_2\dots a_r (a_1 \leq a_2 \leq \dots \leq a_r)$  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:

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

### 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:  $C(n, n/2)$

# 计算几何

## 几何公式

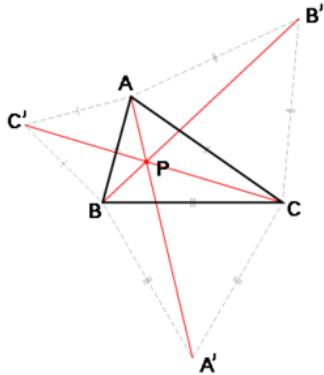
### 费马点

#### 定义

平面上有三个不在同一条直线上的点  $A, B, C$ , 对平面上的另一个点  $P$ , 考虑这一点到原来的三个点的距离之和:  $PA + PB + PC$

费马点是这样一个点  $P_0$ , 使得它到点  $A, B, C$  的距离之和  $P_0A + P_0B + P_0C$  比任何其它的  $PA + PB + PC$  都要小。

#### 作法



三角形的费马点的作法:

当有一个内角  $\geq 120^\circ$  时, 费马点为此角对应顶点

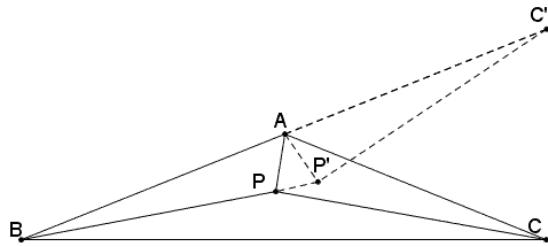
当三角形的内角都  $< 120^\circ$  时

以三角形的每一边为底边, 向外做三个正三角形  $\triangle ABC'$ ,  $\triangle BCA'$ ,  $\triangle CAB'$

连接  $CC'$ ,  $BB'$ ,  $AA'$ , 则三条线段的交点就是所求的点

#### 几何证明

当有一个内角大于等于  $120$  度时候



对三角形内任一点 P

延长 BA 至  $C'$  使得  $AC=AC'$ , 做  $\angle C'AP'=\angle CAP$ , 并且使得  $AP'=AP$ ,  $PC'=PC$ , (其实这就是把三角形 APC 以 A 为中心做了个旋转)

则  $\triangle APC \cong \triangle AP'C'$

$\therefore \angle BAC \geq 120^\circ$

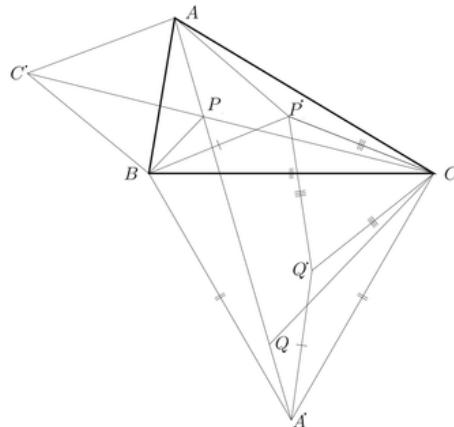
$\therefore \angle PAP' = 180^\circ - \angle BAP - \angle C'AP' = 180^\circ - \angle BAP - \angle CAP = 180^\circ - \angle BAC \leq 60^\circ$

$\therefore$  等腰三角形  $PAP'$  中,  $AP \geq PP'$  (大边对大角)

$\therefore PA + PB + PC \geq PP' + PB + PC' > BC' = AB + AC$

所以 A 是费马点

三角形的内角都小于  $120^\circ$  的情况:



首先证明  $CC'$ ,  $BB'$ ,  $AA'$  三条线交于一点。设  $P$  为线段  $CC'$  和  $BB'$  的交点。注意到  $\triangle C'AC$  和  $\triangle BAB'$  是全等的,  $\triangle C'AC$  可以看做是  $\triangle BAB'$  以  $A$  点为轴心顺时针旋转  $60^\circ$  得到的, 所以  $\angle C'PB = 60^\circ = \angle C'AB$ 。因此,  $C'ABP$  四点共圆。同样地, 可以证明  $B'ACP$  四点共圆。于是:

$$\angle APB = \angle APC = 120^\circ$$

从而

$$\angle CPB = 120^\circ$$

于是可以得出:  $A'BCP$  四点共圆。

$$\angle A'PB = \angle A'CB = 60^\circ$$

$$\angle APA' = \angle APB + \angle A'PB = 120^\circ + 60^\circ$$

$AA'$  三点共线。也就是说  $CC'$ ,  $BB'$ ,  $AA'$  三条线交于一点。

接下来证明交点  $P$  就是到三个顶点距离之和最小的点。

在线段  $AA'$  上选择一点  $Q$ , 使得  $QP = PC$ 。由于  $\angle QPC = 60^\circ$ ,  $\triangle PQC$  是正三角形。于是  $\angle PCB = \angle QCA'$ 。同时  $QC = PC$ ,  $BC = A'C$ , 于是可以得出  $\triangle BPC$  和  $\triangle A'QC$  全等。所以  $QA' = PB$ 。综上可得出:

$$PA + PB + PC = AA'$$

对于平面上另外一点  $P'$ , 以  $P'C$  为底边, 向下作正  $\triangle P'Q'C$ 。运用类似以上的推理可以证明  $\triangle BP'C$  和  $\triangle A'Q'C$  是全等三角形。因此也有:

$$P'A + P'B + P'C = AP' + P'Q' + Q'A$$

平面上两点之间以直线长度最短。因此

$$P'A + P'B + P'C = AP' + P'Q' + Q'A >= AA' = PA + PB + PC$$

也就是说, 点  $P$  是平面上到点  $A$ 、 $B$ 、 $C$  距离的和最短的一点。

最后证明唯一性。如果有另外一点  $P'$  使得

$$P'A + P'B + P'C = PA + PB + PC$$

那么

$$AA' = AP' + P'Q' + Q'A$$

因此点  $P'$  和  $Q'$  也在线段  $AA'$  之上。依照  $P'$  和  $Q'$  的定义, 可以推出

$$\angle AP'C = 180^\circ - \angle A'P'C = 120^\circ$$

$$\angle A'P'C = \angle A'BC = 60^\circ \rightarrow A'P'BC \text{ 四点共圆} \rightarrow \angle BP'C = 180^\circ - \angle BA'C = 120^\circ$$

因此  $P'$  也是  $CC'$ 、 $BB'$ 、 $AA'$  三条线的交点。因此  $P'$  点也就是  $P$  点。因此点  $P$  是唯一的。

## 推广

费马点的定义可以推广到更多点的情况。设平面上有  $m$  个点:  $P_1, P_2, \dots, P_m$ , 又有正实数:  $\lambda_1, \lambda_2, \dots, \lambda_m$ 。费马问题可以推广为: 寻找一个点  $X$ , 使得它到这  $m$  个点的距离之和:

$$\lambda_1 \cdot XP_1 + \lambda_2 \cdot XP_2 + \dots + \lambda_m \cdot XP_m$$

是最小的。

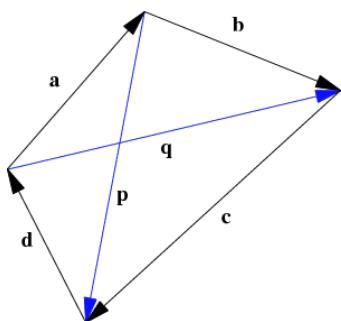
## 平面四边形费马点

1. 在凸四边形  $ABCD$  中, 费马点  $P$  为两对角线  $AC$ 、 $BD$  交点。

2. 在凹四边形  $ABCD$  中, 费马点  $P$  为凹顶点。

四边形以上的多边形没有公式求费马点, 因此可以使用随机化变步长贪心法

## Bretschneider's 公式



边长为  $a, b, c, d$  的面积为

$$K = \frac{1}{4} \sqrt{4p^2q^2 - (b^2 + d^2 - a^2 - c^2)^2} = \sqrt{(s-a)(s-b)(s-c)(s-d) - \frac{1}{4}(ac + bd + pq)(ac + bd - pq)}$$

$$K = \sqrt{(s-a)(s-b)(s-c)(s-d) - abcd \cos^2 \left[ \frac{P+Q}{2} \right]}$$

其中  $p, q$  为对角线长度,  $s$  为半周长,  $P$  和  $Q$  为四边形的一组对角。

ref <http://mathworld.wolfram.com/BretschneidersFormula.html>

## 其他

四维球的超体积公式:  $\frac{1}{2}\pi^2 r^4$

四维球的表体积公式:  $2\pi^2 r^3$

A、B、C 三点共线 iff.  $|\overrightarrow{AB} \times \overrightarrow{AC}| < \varepsilon$

三角形

中线长度:  $Ma = \frac{1}{2}\sqrt{2(b^2 + c^2) - a^2}$

内角平分线长度:  $Ta =$

## 基本代码

```
inline int sign(double a){ return a < -eps ? -1 : a > eps; }
inline int sign(int a){ return a < 0 ? -1 : a > 0; }
template <typename T> inline T sqr(T a){ return a * a; }
```

```
bool isinteger(double x){ return fabs(x - floor(x + eps)) < eps; }
```

//三维凸包

//表面积已经通过 pku3528, 但是只能用 C++ 编译器通过, G++ 编译器运行时错误

```
const int maxn = 1000;           //顶点数
```

```
const int maxf = maxn * 10; //面数
```

```
int face[maxf][3];
```

```
Point3D p[maxn];
```

```
int n, fc;
```

```
void init(){
```

```
    cin >> n;
```

```
    for (int i = 0; i < n; i++) read(p[i]);
```

```

        sort(p, p + n);
        n = unique(p, p + n) - p;//of importance
        fc = 0;
    }

void addface(int a, int b, int c){
    face[fc][0] = a; face[fc][1] = b; face[fc][2] = c;
    ++fc;
}

void delface(int i){
    --fc;
    face[i][0] = face[fc][0]; face[i][1] = face[fc][1]; face[i][2] = face[fc][2];
}

bool findTetrahedron(){
    //找到不共面的四个点 P0,P1,P2,P, 时间复杂度 O(n)
    for (int i = 2; i < n; i++){
        if (!sameline(p[0], p[1], p[i])){
            swap(p[2], p[i]);
            for (int j = i+1; j < n; j++){
                if (!sameplane(p[0], p[1], p[2], p[j])){
                    swap(p[3], p[j]);
                    addface(0, 1, 2); addface(0, 2, 1);
                    return 1;
                }
            }
            return 0;
        }
    }
    return 0;
}

void addpoint(int v){
    static int mark[maxn][maxn], cnt = 0;
    ++cnt;
    if (cnt == 0){
        memset(mark, 0, sizeof(mark)); //mark 该初始化了.....
        ++cnt;
    }

    int a, b, c;
    bool flag = 0;
    for (int i = 0; i < fc;){

```

```

    a = face[i][0]; b = face[i][1]; c = face[i][2];
    if (vol6(p[v], p[a], p[b], p[c]) < -eps){
        flag = 1;
        mark[a][b] = mark[b][a] = mark[a][c] = mark[c][a] = mark[b][c] = mark[c][b] = cnt;
        delface(i);
    }
    else
        i++;
}
if (!flag) return;

int _fc = fc;
for (int i = 0; i < _fc; i++){
    a = face[i][0]; b = face[i][1]; c = face[i][2];
    if (mark[a][b] == cnt) addface(b, a, v);
    if (mark[b][c] == cnt) addface(c, b, v);
    if (mark[c][a] == cnt) addface(a, c, v);
}
}

bool convexHull(){
    int ans = 0, a, b, c;
    random_shuffle(p, p + n);
    if (!findTetrahedron()) return 0;

    for (int i = 3; i < n; i++)
        addpoint(i);
    return 1;
}

int countface(){ //求凸包上有几个面
    static Point3D normal[maxf];
    int a, b, c;
    for (int i = 0; i < fc; i++){
        a = face[i][0]; b = face[i][1]; c = face[i][2];
        normal[i] = unit(cross(p[b]-p[a], p[c]-p[a]));
    }
    sort(normal, normal + fc);
    return unique(normal, normal + fc) - normal;
}

double surface(){ //凸包表面积
    double ans = 0;
    for (int i = 0; i < fc; i++)

```

```

        ans += area2(p[face[i][0]], p[face[i][1]], p[face[i][2]]);
    ans /= 2;
    return ans;
}

double volume(){ //凸包体积
    double ans = 0;
    for (int i = 0; i < fc; i++)
        ans += vol6(p[0], p[face[i][0]], p[face[i][1]], p[face[i][2]]);
    return ans /= 6;
}

int main(){ //pku 3528
    srand(847536234);
    init();
    convexHull();
    printf("%.3f\n", surface());
}

```

## 绕轴旋转、三维空间点对直线垂线的垂足

```

//已经通过 hdu2014, 1006, brbin regional online 2010
#define sqr(x) ((x)*(x))
struct Point{
    double x, y, z;
    Point(double a, double b, double c){ x = a; y = b; z = c; }
    Point() {}
};

Point rotate(double angle, Point a, Point b, Point p){ //点 P 绕轴 $\overrightarrow{AB}$ 旋转(右手系)
    double ab2 = sqr(a.x-b.x) + sqr(a.y-b.y) + sqr(a.z-b.z);
    double lamda = ((b.x-a.x)*(b.x-p.x) + (b.y-a.y)*(b.y-p.y) + (b.z-a.z)*(b.z-p.z)) / ab2;
    Point q = Point(
        a.x * lamda + b.x * (1-lamda),
        a.y * lamda + b.y * (1-lamda),
        a.z * lamda + b.z * (1-lamda)
    ); //q 为 p 在 ab 上的垂足
    double sinadivab = sin(angle) / sqrt(ab2);
    double cosa = cos(angle);
    return Point(
        (p.x-q.x)*cosa + q.x + ((b.y-a.y)*(p.z-q.z)-(b.z-a.z)*(p.y-q.y)) * sinadivab,
        (p.y-q.y)*cosa + q.y + ((b.z-a.z)*(p.x-q.x)-(b.x-a.x)*(p.z-q.z)) * sinadivab,
        (p.z-q.z)*cosa + q.z + ((b.x-a.x)*(p.y-q.y)-(b.y-a.y)*(p.x-q.x)) * sinadivab
    );
}

```

```
}
```

```
Point foot(Point a, Point b, Point p){ //返回 p 在 ab 上的垂足
    double ab2 = sqr(a.x-b.x) + sqr(a.y-b.y) + sqr(a.z-b.z);
    double lamda = ((b.x-a.x)*(b.x-p.x) + (b.y-a.y)*(b.y-p.y) + (b.z-a.z)*(b.z-p.z)) / ab2;
    return Point(
        a.x * lamda + b.x * (1-lamda),
        a.y * lamda + b.y * (1-lamda),
        a.z * lamda + b.z * (1-lamda)
    );
}
```

## 凸包[Graham] - O(nlogn)

```
//已经通过 2 题
const double eps = 1e-8;
inline double sqr(double x){ return x * x;}
struct Point{ double x, y;};
struct lessx{
    bool operator()(const Point& a, const Point &b){
        if (fabs(a.x - b.x) < eps) return a.y < b.y; return a.x < b.x;
    }
};

double cross(const Point& a, const Point &b, const Point &c){ //ab x ac
    return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}

//水平序, 分左右链, 返回凸包点数 k, 凸包按逆时针顺序存在 q[] 中, q[k] = q[0]
//要包含共线点的话, 一定要提前 unique 一下
int graham(Point p[], int n, Point q[]){
    int i, kk, k;
    if (1 == n){ q[0] = q[1] = p[0]; return 1;}
    sort(p, p + n, lessx());
    for (k = 0, i = 0; i < n; i++){
        while (k >= 2 && cross(q[k-2], q[k-1], p[i]) < eps) //要包含共线点则 < -eps
            k--;
        q[k++] = p[i];
    }
    for (kk = k, i = n-2; i >= 0; i--){
        while (k > kk && cross(q[k-2], q[k-1], p[i]) < eps) //要包含共线点则 < -eps
            k--;
        q[k++] = p[i];
    }
}
```

```

        return k-1;
    }

const int maxn = 2000;
Point p[maxn], q[maxn];

int main(){ // bianchengla1737
    int n;
    while (cin >> n, n){
        for (int i = 0; i < n; i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);
        int k = graham(p, n, q);
        double dist = 0;
        for (int i = 0; i < k; i++)
            dist += sqrt(sqrt(q[i].x - q[i+1].x) + sqrt(q[i].y - q[i+1].y));
        printf("%.2f\n", dist);
    }
}

```

## 最远点对

证明：最远点对  $\rightarrow$  同旁内角和小于  $180^\circ \rightarrow$  对踵点

对于要距离最近的平行线把凸包夹住，则改为取最小值即可

```

//已经通过 cugb1167, cugb1256
//凸包 p[] 中的最远点对，要求 p[n] = p[0]
pair<int, int> rotating_calipers(Point p[], int n){
    double ret = 0, tmp;
    int x, y;
    for (int i = 0, j = 1; i < n; i++){
        while (cross(p[i], p[i+1], p[j+1]) > cross(p[i], p[i+1], p[j]) + eps)
            if (++j == n) j = 0;
        if ((tmp = dis(p[i], p[j])) > ret) {
            ret = tmp; x = p[i].idx, y = p[j].idx;
        }
        if ((tmp = dis(p[i+1], p[j])) > ret) {
            ret = tmp; x = p[i+1].idx, y = p[j].idx;
        }
    }
    if (x > y) swap(x, y);
    return make_pair(x, y);
}

```

## 最近点对问题 - 分治 - O(nlogn)

```
//已经通过 hdu1007, cugb1251,
const int maxn = 100005;

struct Point{
    double x, y;
    int index;
    friend bool lessx(const Point& a, const Point& b){ return a.x < b.x; }
    friend bool lessy(const Point& a, const Point& b){ return a.y < b.y; }
};

double dis2(Point p, Point q){ return sqr(p.x - q.x) + sqr(p.y - q.y); }

void partition(Point c[], Point a[], Point b[], int left, int mid, int right){
    //将 b[left..right]分成 c[left..mid]和 c[mid+1..right]
    //使得集合{a[p..m]} = {c[p..m]}, {a[m+1..q]} = {c[m+1..q]}, 且 c 是对 y 有序的
    //初始条件: b[left..right]是对 y 有序的, 且集合{a[left..right]} = {b[left..right]}
    int j = left, k = mid+1;
    for (int i = left; i <= right; i++)
        if (b[i].index <= mid)
            c[j++] = b[i]; //数组 c 左半部保存划分后左部的点, 且对 y 是有序的.
        else
            c[k++] = b[i];
}

void merge(Point p[], Point q[], int left, int mid, int right){
    //q[left..mid], q[mid+1..right]分别是对 y 有序的, 将其合并储存到 p[left..right]
    int i = left, j = mid + 1, k = left;
    while (i <= mid && j <= right)
        if (q[i].y <= q[j].y) p[k++] = q[i++]; else p[k++] = q[j++];
    while (i <= mid) p[k++] = q[i++];
    while (j <= right) p[k++] = q[j++];
}

double closest2(Point a[], Point b[], Point c[], int left, int right){
    //求 a[p..q]的最近点对距离的平方
    //初始条件: 集合{b[left..right]}={a[left..right]}, 且 a 对 x 有序, b 对 y 有序
    if (right - left == 1)
        return dis2(a[left], a[right]);
    if (right - left == 2){
        double x1 = dis2(a[left], a[right]);
        double x2 = dis2(a[left + 1], a[right]);
    }
```

```

        double x3 = dis2(a[left], a[left + 1]);
        return min(min(x1, x2), x3);
    }

    int mid = (left + right) / 2;
    partition(c, a, b, left, mid, right);
    double d1 = closest2(a, c, b, left, mid);
    double d2 = closest2(a, c, b, mid + 1, right);
    double dm = min(d1, d2);
    merge(b, c, left, mid, right);

    //找出离划分基准左右不超过 dm 的部分,存到 c[left..k-1], 且仍然对 y 坐标有序.
    int k = left;
    for (int i = left; i <= right; i++)
        if (sqr(b[i].x - b[mid].x) < dm) c[k++] = b[i];

    //用 c[p..k-1] 中的点更新 dm
    for (int i = left; i < k; i++)
        for (int j = i + 1; j < k && sqr(c[j].y - c[i].y) < dm; j++)
            dm = min(dm, dis2(c[i], c[j]));

    return dm;
}

Point a[maxn], b[maxn], c[maxn];

int main(){
    int n, i; double d2;
    while(cin >> n && n){
        for (i = 0; i < n; i++)
            scanf("%lf%lf", &a[i].x, &a[i].y);
        sort(a, a + n, lessx);
        for (i = 0; i < n; i++) a[i].index = i;
        copy(a, a + n, b);
        sort(b, b + n, lessy);
        d2 = closest2(a, b, c, 0, n - 1);
        printf("%.2f\n", sqrt(d2) * 0.5);
    }
    return 0;
}

```

## 周长最小三角形 - 分治 - O(nlogn)

//已经通过 hdu3548

```

//基本 cp 某人标程 http://code.google.com/codejam/contest/dashboard?c=311101#s=a&a=1

template<class T> inline int size(const T&c) { return c.size(); }

const int BILLION = 1000000000;
const double INF = 1e20;
const double EPS = 1e-11;
typedef long long LL;

struct Point {
    int x,y;
    Point() {}
    Point(int x,int y):x(x),y(y) {}
};

inline Point middle(const Point &a, const Point &b) {
    return Point((a.x + b.x) / 2, (a.y + b.y) / 2);
}

struct CmpX {
    inline bool operator()(const Point &a, const Point &b) {
        if(a.x != b.x) return a.x < b.x;
        return a.y < b.y;
    }
} cmpx;

struct CmpY {
    inline bool operator()(const Point &a, const Point &b) {
        if(a.y != b.y) return a.y < b.y;
        return a.x < b.x;
    }
} cmpy;

inline LL sqr(int x) { return LL(x) * LL(x); }

inline double dist(const Point &a, const Point &b) {
    return sqrt(double(sqr(a.x - b.x) + sqr(a.y - b.y)));
}

inline double perimeter(const Point &a, const Point &b, const Point &c) {
    double x = dist(a,b), y = dist(b,c), z = dist(c,a);
    if (fabs(x + y - z) < EPS || fabs(x - y) > z - EPS) return INF;
    return x + y + z;
}

```

```

double calc(int n, const Point points[], const vector<Point> &pointsByY) {
    if(n < 3) return INF;

    int left = n / 2;
    int right = n - left;
    Point split = middle(points[left-1], points[left]);

    vector<Point> pointsByYLeft, pointsByYRight;
    pointsByYLeft.reserve(left);
    pointsByYRight.reserve(right);

    for (int i = 0; i < n; i++) {
        if(cmpx(pointsByY[i], split))
            pointsByYLeft.push_back(pointsByY[i]);
        else
            pointsByYRight.push_back(pointsByY[i]);
    }

    double res = INF;
    res = min(res, calc(left, points, pointsByYLeft));
    res = min(res, calc(right, points + left, pointsByYRight));

    static vector<Point> closeToTheLine;
    int margin = (res > INF / 2) ? BILLION : int(res / 2);
    closeToTheLine.clear();
    closeToTheLine.reserve(n);

    int start = 0;
    for(int i = 0; i < n; ++i) {
        Point p = pointsByY[i];
        if(abs(p.x - split.x) > margin) continue;
        while(start < size(closeToTheLine) && p.y - closeToTheLine[start].y > margin) ++start;
        for(int i = start; i < size(closeToTheLine); ++i) {
            for(int j = i+1; j < size(closeToTheLine); ++j) {
                res = min(res, perimeter(p, closeToTheLine[i], closeToTheLine[j]));
            }
        }
        closeToTheLine.push_back(p);
    }
    return res;
}

double calc(vector<Point> &points) {

```

```

sort(points.begin(), points.end(), cmpx);
vector<Point> pointsByY = points;
sort(pointsByY.begin(), pointsByY.end(), cmpy);
return calc(size(points), &points[0], pointsByY);
}

int main() {
    int t, n;
    scanf("%d", &t);
    for (int icase = 1; icase <= t; icase++){
        scanf("%d", &n);
        vector<Point> points; points.reserve(n);
        for (int i = 0; i < n; i++){
            int x, y;
            scanf("%d%d", &x, &y);
            points.push_back(Point(2 * x, 2 * y));
        }

        printf("Case %d: ", icase);
        double res = calc(points);

        if (res >= INF - EPS){
            printf("No Solution\n");
        }
        else {
            printf("%.3f\n", res / 2);
        }
    }
}

```

## 面积最大多边形

方法正确性详见 IEEE 论文 On a general method for maximizing and minimizing among certain geometric problems (由于论文思维跳跃过大，笔者没看懂 TAT)

```

//要求 p[1..n-1]是凸包且 p[n] = p[0], 时间复杂度 O(n)
double max_area_triangle(point p[], int n){ //hdu2202
    if (n < 3) return 0;
    int a = 0, b = 1, c = 2, ra = a, rb = b, rc = c;
    double A = area(p[a], p[b], p[c]);
#define inc(x) do { if (++x == n) x = 0; } while (0)
    for (;;){
        for(;;){
            while ( area(p[a], p[b], p[c+1]) >= area(p[a], p[b], p[c]) ) inc(c);

```

```

        if ( area(p[a], p[b+1], p[c]) >= area(p[a], p[b], p[c]) ) inc(b); else break;
    }
    if ( area(p[a], p[b], p[c]) > A ){
        ra = a, rb = b, rc = c; A = area(p[a], p[b], p[c]);
    }
    inc(a);
    if (a == b) inc(b);
    if (b == c) inc(c);
    if (a == 0) break;
}
return A;
#undef inc
}

double max_area_quadrangle(point p[], int n){ //bzoj1069
if (n == 3) return area(p[0], p[1], p[2]);
if (n < 4) return 0;
int a = 0, b = 1, c = 2, d = 3, ra = a, rb = b, rc = c, rd = d;
double A = area(p[a], p[b], p[c], p[d]);
#define inc(x) do { if (++x == n) x = 0; } while (0)
for (;;){
    for (;;){
        for(;;) {
            while ( area(p[c], p[d+1], p[a]) >= area(p[c], p[d], p[a]) ) inc(d);
            if ( area(p[b], p[c+1], p[d]) >= area(p[b], p[c], p[d]) ) inc(c); else break;
        }
        if ( area(p[a], p[b+1], p[c]) >= area(p[a], p[b], p[c]) ) inc(b); else break;
    }
    if ( area(p[a], p[b], p[c], p[d]) > A ){
        ra = a, rb = b, rc = c, rd = d; A = area(p[a], p[b], p[c], p[d]);
    }
    inc(a);
    if (a == b) inc(b);
    if (b == c) inc(c);
    if (c == d) inc(d);
    if (a == 0) break;
}
return A;
#undef inc
}

void dfs(int dep, point p[], int n, int k, int idx[], int opt[]){
#define inc(x) do { if (++x == n) x = 0; } while (0)
if (dep == k - 1) {

```

```

        while( area(p[idx[k-2]],p[idx[k-1]+1],p[idx[0]])>=area(p[idx[k-2]],p[idx[k-1]],p[idx[0]])) {
            inc(idx[k-1]);
            return;
        }
        dfs(dep+1, p, n, k, idx, opt);
        while( area( p[idx[dep-1]], p[idx[dep]+1], p[idx[dep+1]]) >=
            area( p[idx[dep-1]], p[idx[dep]], p[idx[dep+1]]) ) {
            inc( idx[dep] );
            dfs(dep+1, p, n, k, idx, opt);
        }
    }

double max_area_kgon(point p[], int n, int k){ //bzoj 1069
    if (n < 3) return 0;
    static int maxk = 10;
    double A = 0;
    for (int i = 3; i < k; i++){
        A += area(p[0], p[i-2], p[i-1]);
        if ( n == i ) return A;
    }
    A += area(p[0], p[k-2], p[k-1]);
    int idx[maxk], opt[maxk];
    for (int i = 0; i < k; i++)
        idx[i] = opt[i] = i;
    for (;;){
        dfs(1, p, n, k, idx, opt);
        double A0 = 0;
        for (int i = 3; i <= k; i++)
            A0 += area(p[idx[0]], p[idx[i-2]], p[idx[i-1]]);
        if ( A0 > A ){
            for (int i = 0; i < k; i++) opt[i] = idx[i];
            A = A0;
        }
        inc(idx[0]);
        for (int i = 1; i < k; i++)
            if (idx[i] == idx[i-1] ) inc(idx[i]);
        if (idx[0] == 0) break;
    }
    return A;
#define inc
}

```

## 点集的最小覆盖圆 - 期望 O(n)

```
//已经通过 zju1450
struct Point{ double x, y;};
struct Circle{ Point c; double r;};

double Distance(const Point& a, const Point& b){
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}

Circle Circumcircle(const Point& a, const Point &b, const Point &c){
    //三角形的外接圆
    Circle ret;
    double a1 = 2 * (a.x - b.x), b1 = 2 * (a.y - b.y), c1 = sqr(a.x) + sqr(a.y) - sqr(b.x) - sqr(b.y);
    double a2 = 2 * (a.x - c.x), b2 = 2 * (a.y - c.y), c2 = sqr(a.x) + sqr(a.y) - sqr(c.x) - sqr(c.y);
    //a1 * x + b1 * y = c1, a2 * x + b2 * y = c2
    double d = a1 * b2 - a2 * b1, d1 = c1 * b2 - c2 * b1, d2 = a1 * c2 - a2 * c1;
    ret.c.x = d1 / d; ret.c.y = d2 / d; ret.r = Distance(a, ret.c); return ret;
}

Circle Circumcircle(const Point& a, const Point &b){
    Circle ret; ret.c.x = (a.x + b.x) * 0.5; ret.c.y = (a.y + b.y) * 0.5;
    ret.r = Distance(a, ret.c); return ret;
}

Circle Circumcircle(const Point& a){
    Circle ret; ret.c = a; ret.r = 0; return ret;
}

Circle minCircumcircle(Point p[], int n){
    Circle ret;
    if (n == 1){
        ret.c = p[0]; ret.r = 0; return ret;
    }
    // random_shuffle(p, p + n);
    ret = Circumcircle(p[0],p[1]);
    for (int i = 2; i < n; i++){
        if (Distance(ret.c, p[i]) > ret.r + eps){
            ret = Circumcircle(p[0],p[i]);
            for (int j = 1; j < i; j++){
                if (Distance(ret.c, p[j]) > ret.r + eps){
                    ret = Circumcircle(p[j], p[i]);
                    for (int k = 0; k < j; k++){
                        if (Distance(ret.c, p[k]) > ret.r + eps){
                            ret = Circumcircle(p[k], p[i]);
                        }
                    }
                }
            }
        }
    }
}
```

```

        if (Distance(ret.c, p[k]) > ret.r + eps){
            ret = Circumcircle(p[k], p[j], p[i]);
        }
    }
}
}

return ret;
}

const int maxn = 2000000;
Point p[maxn];

int main(){ //zju1450
    int n;
    while (scanf("%d", &n) != EOF && n){
        for (int i = 0; i < n; i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);
        Circle c = minCircumcircle(p, n);
        printf("%.2f %.2f %.2f\n", c.c.x, c.c.y, c.r);
    }
}

```

## 外接圆最大三角形 - 枚举 1 点，极角排序 - O(n<sup>2</sup>logn)

```

const int maxn = 1000;
const double eps = 1e-8;
const double inf = 1e18;

struct Point { double x, y; } p[maxn], q[maxn];
inline bool equal(const Point& a, const Point &b){ return fabs(a.x - b.x) + fabs(a.y - b.y) < eps; }

template<Point *p> bool less(Point a, Point b){
    a.x -= p->x; a.y -= p->y;
    b.x -= p->x; b.y -= p->y;
#define get_quad(x, y) ( (x) >= 0 ? ( (y) >= 0 ? 1 : 2 ) : ( (y) <= 0 ? 3 : 4 ) )
    int aq = get_quad(a.x, a.y), bq = get_quad(b.x, b.y);
    if (aq != bq) return aq < bq; else return a.x * b.y - b.x * a.y > 0;
}

double sqr_circumcircle(Point a, Point b, Point c){
#define sqr(x) ( (x) * (x) )
    double a1 = 2 * (a.x - b.x), b1 = 2 * (a.y - b.y), c1 = sqr(a.x) + sqr(a.y) - sqr(b.x) - sqr(b.y);

```

```

        double a2 = 2 * (a.x - c.x), b2 = 2 * (a.y - c.y), c2 = sqr(a.x) + sqr(a.y) - sqr(c.x) - sqr(c.y);
        double d = a1 * b2 - a2 * b1, d1 = c1 * b2 - c2 * b1, d2 = a1 * c2 - a2 * c1;
        if (fabs(d) < eps) return 0;
        double x = d1 / d, y = d2 / d;
        return sqr(a.x - x) + sqr(a.y - y);
    }

int main(){
    int casecnt, n;
    scanf("%d", &casecnt);
    while (casecnt--){
        scanf("%d", &n);
        for (int i = 0; i < n; i++)
            scanf("%lf%lf", &p[i].x, &p[i].y);
        double res2 = 0;
        for (int i = 0; i < n; i++){
            int c = 0;
            q[c++] = p[i];
            for (int j = 0; j < n; j++)
                if (!equal(p[j], p[i]))
                    q[c++] = p[j];
            sort(q + 1, q + c, ::less<q>);
            q[c] = q[1];
            for (int j = 1; j < c; j++){
                res2 = max(res2, sqr_circumcircle(q[0], q[j], q[j+1]));
            }
        }
        printf("%.3f\n", sqrt(res2));
    }
}

```

## 判断两个正交矩形是否有公共点

```

bool intersect(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4){
    //已经通过 bnu4338
    //判断两个正交矩形(x1,y1) - (x2,y2), (x3,y3) - (x4,y4)是否有公共点
    if (x1 > x2) swap(x1, x2); if (y1 > y2) swap(y1, y2);
    if (x3 > x4) swap(x3, x4); if (y3 > y4) swap(y3, y4);
    return !(x2 < x3 || y2 < y3 || x4 < x1 || y4 < y1);
}

```

# 数论

## 欧几里德算法

最坏情况：Fibonacci 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 gcd2(T a, T b){ //不能传引用!!!
    if (b) while (b ^= a ^= b ^= a %= b); return a;
}
```

```
T gcd3(T a, T b){
    if (a == 0) return b; if (b == 0) return a;
    while (a >= b ? a %= b : b %= a); return a + b;
}
```

```
T gcd4(T a, T b){ //stein 算法, 复杂度 log(n),  not tested
    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;
        if (a > b) a -= b; else 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  $d \% g \neq 0$  无解, 否则

$$x = x_0 * (d / g) - (b / g) * t$$

$$y = y_0 * (d / g) + (a / g) * t$$

## 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 quot = a / b, t;
        // {a, b} = {b, a % b}
        // {nextx, x} = {x - quot * nextx, nextx}
        // {nexty, y} = {y - quot * nexty, nexty}
        t = a % b; a = b; b = t;
        t = x - quot * nx; x = nx; nx = t;
        t = y - quot * ny; y = ny; ny = t;
    }
    return a;
}
```

## 四则运算（乘除指对）

```
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;
}

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

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;
}

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

T pow_mod(T a, T n, T p){ //要求 a >= 0, 已经通过 hdu2837
    T ret = 1; a = a % p;
    for(;n;){
        if (n & 1) ret = mul_mod(ret, a, p);
        if (n /= 2) a = mul_mod(a, a, p);
    }
    return ret;
}

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;
}

//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 ret = 0, tmp = a % n;
    for (; b; b >>= 1) {
        if (b & 1)
            if ((ret += tmp) >= n) ret -= n;
        if ((tmp <= 1) >= n) tmp -= n;
    }
    return ret;
}

int sqrt_mod(int a, int n){
    //时间复杂度 b log n + log n * log n, 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 0;
}

```

```

}

//因式分解，最坏O( $n^{1/2}$ )，平均O( $n^{1/4}$ )，可在  $\Theta(\sqrt{p})$ 的期望时间找出 n 的一个小因子 p
//已经通过 cugb1082(数据可能很水)， bit2011 多校某题， poj2429
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 {
            i++;
            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);
    }
}

```

## 计算 $f[i] = a^i \bmod n$ (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

```

/*
 $ax \equiv b \pmod{n}$ 
若  $(a, n) \mid b \Rightarrow Z_n$  有  $(a, n)$  个根，否则无根
 $\Leftrightarrow ax - ny = b$ 
let  $d = gcd(a, n)$ 
if  $b \pmod{d} \neq 0$  no solution
else solve  $ax_0 - by_0 = d$ 
 $xi = ((x_0 * b/d) + i * (n/d)) \pmod{n}$ 
 $min(xi) = (x_0 * b/d) \pmod{n/d}$ 
*/
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  $a \pmod{n} = b$ , 则  $c \geq b$  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 &d_out, T &r_out){
    T q1, q2, D = exgcd(d1, d2, q1, q2);
    T c = r1 - r2; if (c < 0) c += d1;
    d_out = d1 / D * d2;
    if (c % D != 0) {

```

```

        r_out = -1; return false;
    }
    T d1divd = d1 / D;
    q2 *= c / D;
    q2 = q2 % d1divd; if (q2 <= 0) q2 += d1divd;
    r_out = q2 * d2 + r2;
    if (r_out >= d_out) r_out -= d_out;
    return true;
}

bool CRT(T d[], T r[], int n, T &d_out, T &r_out){
    //d[]除数, r[]余数 0..n-1
    d_out = 1, r_out = 0;
    for (int i = 0; i < n; i++)
        if (!CRT2(d_out, r_out, d[i], r[i], d_out, r_out)) 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 d_out, r_out;
        CRT(d, r, k, d_out, r_out);
        printf("%lld\n", r_out);
    }
}

```

## 素数

|    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|
| 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

|             |   |                                    |
|-------------|---|------------------------------------|
| $\psi_1$    | = | 2047                               |
| $\psi_2$    | = | 1373653                            |
| $\psi_3$    | = | 25326001                           |
| $\psi_4$    | = | 3215031751                         |
| $\psi_5$    | = | 2152302898747                      |
| $\psi_6$    | = | 3474749660383                      |
| $\psi_7$    | = | 341550071728321                    |
| $\psi_8$    | = | 341550071728321                    |
| $\psi_9$    | = | 3825123056546413051                |
| $\psi_{10}$ | = | 3825123056546413051                |
| $\psi_{11}$ | = | 3825123056546413051                |
| $\psi_{12}$ | = | 318665857834031151167461           |
| $\psi_{13}$ | = | 3317044064679887385961981          |
| $\psi_{14}$ | = | 6003094289670105800312596501       |
| $\psi_{15}$ | = | 59276361075595573263446330101      |
| $\psi_{16}$ | = | 564132928021909221014087501701     |
| $\psi_{17}$ | = | 564132928021909221014087501701     |
| $\psi_{18}$ | = | 1543267864443420616877677640751301 |
| $\psi_{19}$ | = | 1543267864443420616877677640751301 |
| $\psi_{20}$ | > | $10^{36}$ .                        |

|  |                           |
|--|---------------------------|
| $n < 25,000,000,000 \&\& n \neq 3,215,031,751(\text{prime})$ | 2, 3, 5, 7                |
| $n < 75,792,980,677$   | 2, 379215, 457083754      |
| $n < 21,652,684,502,221$                                     | 2, 1215, 34862, 574237825 |

```
bool miller_rabin(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 cugb1082(可能数据很水)、bit2011 多校某题

```
bool isprime(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 || !miller_rabin(r % n, n)) return 0;
}
return 1;
}

```

## 高效版

```

int strong_psudo_prime_test(LL n, int base){
    LL m = n - 1, res;
    int s = 0;
    while (m % 2 == 0) m >>= 1, s++;
    res = pow_mod(base, 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 == 321503175LL) 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];
int 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  $\leftrightarrow$  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++;
    }
}
```

## 枚举约数

```
//已经通过 PKU2480, PKU3696
T enumdivisor(T n, T m){
    //枚举约数
    static T p[20], product[20];
    static int rs[20], r[20], c;
    T ret = inf;

    factorize(n, p, r, c);
    rs[0] = 0;
    product[0] = 1;
    for (int i = 0; i >= 0;){
        if (i == c){
            //TO DO change your code here
            T d = product[c];
            if (pow_mod(10, d, m) == 1){
                if (d < ret) ret = d;
            }
        }

        ++rs[-i]; //n = 1 时这里会故意让 rs 越界, 所以定义 rs[]时要注意
        product[i] = product[i] * p[i];
    }

    if (rs[i] <= r[i]){
        rs[++i] = 0;
        product[i] = product[i-1];
    } else{
        ++rs[-i];
        product[i] = product[i] * p[i];
    }

    return ret;
}
```

## 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。

```
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  $ac \equiv bc \pmod{p}$  可以推出  $a \equiv b \pmod{p}$

证明: 因为  $ac \equiv bc \pmod{p}$ , 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  $a \neq b$  不成立, 则  $c \nmid kp$ . 因为  $c$  和  $p$  没有公因子, 因此显然  $c \nmid k$ , 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  $a \equiv b \pmod{p}$

如果  $a = b$ , 则  $a \equiv b \pmod{p}$  显然成立

得证

### 定义

欧拉函数是指: 对于一个正整数  $n$ , 小于  $n$  且和  $n$  互质的正整数(包括 1)的个数, 记作  $\phi(n)$ 。(也有材料定义  $\phi(n)$  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  $|Z_n| = \phi(n)$

类似的, 可以定义  $\Phi(n)$  为小于等于  $n$  且和  $n$  互质的正整数的个数, 则

$$\Phi(n) = \begin{cases} \phi(n) & (n \geq 2) \\ 1 & (n = 1) \end{cases}$$

## 性质

1. 对于素数  $p$ ,  $\phi(p) = p - 1$

2. 对于  $p^k$ ,  $\phi(p^k) = p^k - p^{k-1}$

证明: 小于  $p^k$  的正整数个数为  $p^k - 1$  个, 其中和  $p^k$  不互质的正整数有  $\{p * 1, p * 2, \dots, p * (p^{k-1} - 1)\}$  共计  $p^{k-1} - 1$  个, 所以  $\phi(p^k) = (p^k - 1) - (p^{k-1} - 1) = p^k - p^{k-1}$

3. 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  $Zp \times Zq$  之间的一一映射 ( $a \in Zp, b \in Zq \Leftrightarrow (a * q + b * p) \bmod n \in Zn$ )。所以  $\phi(p * q) = |Zn| = |Zp \times Zq| = \phi(p) * \phi(q)$

4. 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:

$$n = \prod_{i=1}^n p_i^{k_i}, \text{ 根据前面的结论很容易得到 } \phi(n) = \prod_{i=1}^n p_i^{k_i-1} (p_i - 1) = n \prod_{i=1}^n \frac{p_i - 1}{p_i}$$

5. 欧拉定理: 对于互质的正整数  $a$  和  $n$ , 有  $a^{\phi(n)} \equiv 1 \pmod{n}$

证明:

(1) 引理: 令  $Zn = \{x_1, x_2, \dots, x_{\phi(n)}\}$ ,  $S = \{a * x_1 \bmod n, a * x_2 \bmod n, \dots, a * x_{\phi(n)} \bmod n\}$ , 则  $Zn = S$ 。

① 因为  $a$  与  $n$  互质,  $x_i (1 \leq i \leq \phi(n))$  与  $n$  互质, 所以  $a * x_i$  与  $n$  互质, 所以  $a * x_i \bmod n \in Zn$

② 若  $i \neq j$ , 那么  $x_i \neq x_j$ , 且由  $a, n$  互质可得  $a * x_i \bmod n \neq a * x_j \bmod n$  (消去律)。

(2)  $a^{\phi(n)} * x_1 * x_2 * \dots * x_{\phi(n)} \bmod n$

$$\equiv (a * x_1) * (a * x_2) * \dots * (a * x_{\phi(n)}) \bmod n$$

$$\equiv (a * x_1 \bmod n) * (a * x_2 \bmod n) * \dots * (a * x_{\phi(n)} \bmod n) \bmod n$$

$$\equiv x_1 * x_2 * \dots * x_{\phi(n)} \bmod n$$

对比等式的左右两端, 因为  $x_i (1 \leq i \leq \phi(n))$  与  $n$  互质, 所以  $a^{\phi(n)} \equiv 1 \pmod{n}$  (消去律)。

a) 推论:  $a^{-1} \equiv a^{\phi(n)-1} \pmod{n}$

6. 费马小定理: 若正整数  $a$  与素数  $p$  互质, 则有  $a^{p-1} \equiv 1 \pmod{p}$ 。

证明: 由于  $\phi(p) = p - 1$ , 代入欧拉定理即可证明。

7. 利用欧拉函数如下性质, 可以快速求出欧拉函数的值( $a$  为  $n$  的质因数)

若  $(a|n \wedge a^2|n)$  则有:  $\phi(n) = \phi(n/a) * a$ ;

若  $(a|n \wedge \nexists a^2|n)$  则有:  $\phi(n) = \phi(n/a) * (a - 1)$ ;

8. 将  $i=1..n$  按照  $(i, n) = d$  分成不同的等价类, 则等价类  $\{x \mid (x, n) = d\} = \{d, 2d, \dots, kd, \dots, (n/d)d\}$   
 $| k \in Z_{n/d}\}$

9.  $1..n$  中与  $n$  互质的数的和为  $n * \Phi(n)/2$

证明提示: if  $\gcd(n, i) = 1$  then  $\gcd(n, n-i) = 1$  ( $1 \leq i \leq n$ )

10.  $1..n$  与  $n$  的最大公约数是  $d$  的数的个数为  $\Phi(n/d)$ , 和为  $(n/d) * \Phi(n/d)/2$

11. 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  $\{x, x^2, \dots, x^{p-1}\} = \{1, 2, \dots, n\}$ , 我们考虑  $x^i$  是不是原根。

考虑集合  $\{x^i, x^{2i}, \dots, x^{(p-1)i}\}$ , 假设存在  $x^{ai} \equiv x^{bi} (1 \leq a < b \leq p-1)$ , 则  $ai \equiv bi \pmod{p-1}$ ,  $(b-a) \not\equiv 0 \pmod{p-1}$ .

若  $(i, p-1) = 1$ , 则  $b-a \equiv 0 \pmod{p-1}$ ,  $a=b$ , 矛盾, 故  $x^i$  是原根

若  $(i, p-1) \neq 1$ , 无此结论。事实上  $x^{(p-1)/(i, p-1)} = 1$ , 即  $\text{ord}(x^i) \leq (p-1)/(i, p-1) < p-1$ , 故  $x^i$

不是原根。

综上  $x^i$  是原根当且仅当  $(i, p - 1) = 1$

## 求法

### 筛法

```
//时间复杂度 O(n)
//已经通过 pku2478, pku2480, zju1759, hdu2837, pku2992
const int maxn = 100000;
int phi[maxn+1]; //phi[i] = 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){ //大 phi
    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){
    //注意这里求的是大 phi
    //已经通过 PKU2480
    T ret = 1;
    for (int i = 0; i < c; i++)
        if (r[i] > 0)
            ret *= (p[i] - 1) * pow(p[i], r[i] - 1);
    return ret;
}

```

## Jordan's totient function 约旦欧拉函数

### DEFINITION 定义

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

### 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  $n = p_1^{\alpha_1} \dots p_m^{\alpha_m}$  then

$$J_k(n) = n^k \left(1 - \frac{1}{p_1^k}\right) \dots \left(1 - \frac{1}{p_m^k}\right) = \prod_i p_i^{k(\alpha_i-1)} (p_i^k - 1)$$

3. (Gauss' type formula) The following formula holds

$$n^k = \sum_{d|n} J_k(d)$$

### 其他积性函数

$n$  的所有因子经过  $f$  运算之后的和( $\text{sum}\{f(x) : x | n\}$ )

e.g.  $n$  的所有因子之和  $= (p_1^{0+...+r_1})(p_2^{0+...+r_2}) \dots (p_k^{0+...+r_k})$

## The Determinants of GCD matrices

DEFINITION Let  $S = \{x_1, \dots, x_n\}$  be a finite ordered set of distinct positive integers. The greatest common divisor matrix (or GCD matrix) defined on  $S$  is given 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  $S = \{x_1, \dots, x_n\}$ ,  $[S] = (s_{ij})_{n \times n}$ , 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  $S = \{x_1, \dots, x_n\}$  be an ordered set of distinct positive integers, and  $S' = \{x_1, \dots, x_n, x_{n+1}, \dots, x_{n+s}\}$  the minimal FC ordered set containing  $S$ , where  $x_{n+1} < \dots < x_{n+s}$ . Define the  $n \times (n+s)$  matrix  $A = (a_{ij})$  by

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

where  $\varphi$  is Euler's totient function. The  $[S] = A A^T$ .

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

$$\det[S] = \varphi(x_1) \dots \varphi(x_n).$$

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

$$\det[S] \geq \varphi(x_1) \dots \varphi(x_n).$$

And equality holds iff  $S$  is FC.

## 勒让德

### 勒让德函数

数学上，勒让德函数指以下勒让德微分方程的解：

$$(1 - x^2) \frac{d^2 P(x)}{dx^2} - 2x \frac{dP(x)}{dx} + n(n+1)P(x) = 0.$$

为求解方便一般也写成如下施图姆-刘维尔形式：

$$\frac{d}{dx} \left[ (1 - x^2) \frac{d}{dx} P(x) \right] + n(n+1)P(x) = 0.$$

当方程满足  $|x| < 1$  时，可得到有界解（即解级数收敛）。并且当  $n$  为非负整数，即  $n = 0, 1, 2, \dots$  时，在  $x = \pm 1$  点亦有有界解。这种情况下，随  $n$  值变化方程的解相应变化，构成一组由正交多项式组成的多项式序列，这组多项式称为勒让德多项式。

## 勒让德多项式

勒让德多项式  $P_n(x)$  是  $n$  阶多项式，可用罗德里格公式表示为：

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} [(x^2 - 1)^n].$$

$$\text{递推关系: } P_n(x) = \frac{2n-1}{n} x P_{n-1}(x) - \frac{n-1}{n} P_{n-2}(x) \quad (n \geq 2)$$

| $T_0(x)$ | $T_1(x)$ | $T_2(x)$   | $T_3(x)$    |
|----------|----------|------------|-------------|
| 1        | $x$      | $2x^2 - 1$ | $4x^3 - 3x$ |

## 勒让德多项式的性质

勒让德多项式的一个重要性质是其在区间  $-1 \leq x \leq 1$  关于 L2 内积满足正交性，即：

$$\int_{-1}^1 P_m(x) P_n(x) dx = \frac{2}{2n+1} \delta_{mn}$$

其中  $\delta_{mn}$  为克罗内克  $\delta$  记号，当  $m=n$  时为 1，否则为 0。

## 二次剩余

定义 1：设素数  $p > 2$ ,  $d$  是整数,  $p \nmid d$ , 若同余方程  $x^2 \equiv d \pmod{p}$  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  $x^2 \equiv d \pmod{p}$  的解数为 2

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

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

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

定义 5：设奇数  $P > 1$ ,  $P = p_1 \cdots p_s$ ,  $p_j$  ( $1 \leq j \leq s$ ) 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  $\left(\frac{d}{p_j}\right)$  ( $1 \leq j \leq s$ ) 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

$$\bullet \quad \left(\frac{1}{p}\right) = 1, \quad \left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}, \quad \left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$$

- $\left(\frac{d}{P}\right) = \begin{cases} 0 & (d, P) > 1 \\ \pm 1 & (d, P) = 1 \end{cases}$
- $\left(\frac{d}{P}\right) = \left(\frac{d+P}{P}\right)$
- $\left(\frac{cd}{P}\right) = \left(\frac{c}{P}\right)\left(\frac{d}{P}\right)$
- $\left(\frac{d}{P_1 P_2}\right) = \left(\frac{d}{P_1}\right)\left(\frac{d}{P_2}\right)$
- (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 P > 1, 奇数 Q > 1, (P, Q) = 1, 我们有

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

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

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

其中  $P = p_1 \cdots p_s$ ,  $p_j$  ( $1 \leq j \leq s$ ) 是素数.

定义 8: 设素数  $p > 2$ ,  $d$  是整数,  $p \nmid d$ , 若同余方程  $x^n \equiv d \pmod{p}$  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  $x^n \equiv d \pmod{p}$  有解的充要条件是

$$d^{\frac{p-1}{k}} \equiv 1 \pmod{p}$$

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  $\varphi(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ , 根据  $m$  组数据  $(x_i, y_i)$  ( $1 \leq i \leq m, m > n$ ) 代入公式

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

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

$$\begin{bmatrix} m & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}$$

## 排列组合

### 组合数 $C(m, n)$

```
long long C(long long m, long long n){  
    if (n > m / 2) n = m - n; // of importance  
    long long res = 1;  
    for (long long i = 1; i <= n; i++){  
        res = res * m-- / i;  
    }  
    return res;  
}
```

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

Lanczos 估算公式

$$\begin{aligned}\Gamma(x + 1) &= (x + \gamma + 0.5)^{x+0.5} e^{-(x+\gamma+0.5)} \sqrt{2\pi} \left( c_0 + \frac{c_1}{x+1} + \frac{c_2}{x+2} + \dots + \frac{c_N}{x+N} + \varepsilon \right) \\ \ln(\Gamma(x)) &= -(x + \gamma - 0.5) + (x - 0.5) \ln(x + \gamma - 0.5) + \ln \left( \sqrt{2\pi} \left( c_0 + \sum_{i=1}^N \frac{c_i}{x+i-1} \right) \right)\end{aligned}$$

当  $\gamma = 5$ ,  $N = 6$  时, 误差  $|\varepsilon| < 2 \times 10^{-10}$ 。注意: 这里只能保证 10 位有效数字

```
double log_gamm(double x){
```

```

static double c[] = {
    1.000000000190015,
    76.18009172947148,
    -86.50532032941677,
    24.01409824083091,
    -1.231739572450155,
    0.1208650973866179e-2,
    -0.5395239384953e-5,
};

assert(x >= 0);
double s = c[0];
for (int i = 1; i <= 6; i++){
    s += c[i] / (x+i-1);
}
double t = x +4.5;
return -t + (x-0.5)*log(t) + log(2.506628274631005 * s);
}

double fact(double x){
    return exp(log_gamm(x+1.0));
}

pair<double, int> gamm(double x){
    static const double d = log10(exp(1.0));
    double e = log_gamm(x) * d; //log10_gamm
    int ie = (int)e;
    double a = pow(10.0, e - ie);
    return make_pair(a, ie);
}

```

## 莫比乌斯函数

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ 0 & \text{if } p^2 \mid n \text{ for some prime } p \\ (-1)^r & \text{if } n = p_1 \dots p_r, \text{ where the } p_i \text{ are distinct primes} \end{cases}$$

性质

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

## 莫比乌斯反演定理

设  $f(n)$  和  $g(n)$  是定义在自然数集  $\mathbb{N}$  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

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

反之亦然

## Squarefree 数

```
// n 是 squarefree 数 iff  $\mu(n) \neq 0$ .
// 小于 n 的 squarefree 数的个数  $Q(n) = \frac{6n}{\pi^2} + O(\sqrt{n})$ 
int sf[maxn], factcnt[maxn], sfcnt = 0;

void square_free(int n = maxn - 1){ // 复杂度  $n \log \log n$ , 已经通过 tju3121
    sfcnt = 0; factcnt[1] = 0; sf[sfcnt++] = 1;
    for (int i = 2; i <= n; i++) {
        if ( !factcnt[i] )
            for (int j = 1, k = i; k <= n; j++, k += i)
                if (j == i){
                    j = 0; factcnt[k] = -1;
                } else if (factcnt[k] != -1){
                    ++factcnt[k];
                }
        if ( factcnt[i] != -1 )
            sf[sfcnt++] = i;
    }
}
```

求 1..a 和 1..b 中互质的数(最大公约数为 k 的数)的对数——  
利用 squarefree 数进行容斥

```
LL coprime_pair_cnt(int a, int b){
    // 求[1..a] [1..b] 互质的对数, 已通过 tju3317, cugb1213
    // 注无序对为: cnt(a,b) - (cnt(a,a)-1)/2 (a <= b), 已通过 hdu1695
    if (a > b) swap(a, b);
    LL r = 0;
    for (int i = 0; i < sfcnt && sf[i] <= a; i++) {
        LL delta = (LL)(a / sf[i]) * (b / sf[i]);
        factcnt[sf[i]] & 1 ? r -= delta : r += delta;
    }
    return r;
```

}

## 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);
    }
}
```

### 典型例题

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

SOLUTION:

$$\text{ans} = \begin{cases} -1 & n = 4k + 2 \\ 2(-(a - b)^2)^{\frac{n}{4}} - 1 & n = 4k \end{cases} \text{ mod } m$$

PROOF:

题目保证了  $0 < \sqrt{b} - \sqrt{a} < 1$ , 考虑  $x_n = (\sqrt{b} + \sqrt{a})^{2n} = (a + b + 2\sqrt{ab})^n$ ,  $y_n = (\sqrt{b} - \sqrt{a})^{2n} = (a + b - 2\sqrt{ab})^n$  和  $z_n = x_n + y_n$ . 那么  $x_1$  和  $y_1$  是方程  $u^2 - 2(a + b)u + (a - b)^2 = 0$  的根, 于是有  $z_{n+2} = 2(a + b)z_{n+1} - (a - b)^2 z_n$ . 又显然  $y_n < 1$ , 所以  $|x_n| = z_n - 1$ , 于是由  $z_0 = 2, z_1 = 2(a + b)$ , 利用矩阵乘法就可以求出任意  $|x_n| \text{ mod } m$  了, 时间复杂度  $O(\log n)$

**Given n and k, find max{ i |  $k^i$  divides  $n!$  } – java**

```
import java.util.*;
import java.io.*;
import java.math.*;

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);
    if ( v == null ){
        Table.put(n, 1);
    } else {
        Table.put(n, ++v);
    }
} else {
    BigInteger d = rho(n);
    factorize(d); factorize(n.divide(d));
}
}

public static void main(String args[]){ //hdu3988
Scanner cin = new Scanner( new BufferedInputStream(System.in) );
int casecnt = cin.nextInt();
for (int icase = 1; icase <= casecnt; icase++){
    BigInteger n;
    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: ", icase);
    if (r.equals(inf)) {
        System.out.println("inf");
    } else {
        System.out.println(r);
    }
}

static BigInteger k;
static HashMap<BigInteger, Integer> Table;
static BigInteger inf = new BigInteger("9223372036854775808");
static Random rand = new Random();
}

```

## 多项式运算

### 多项式乘法 - FFT - O(nlogn)

```

const int maxn = 10000;
const double pi = acos(-1.0);
const double eps = 1e-12;

typedef complex<double> cdouble;

void FFT(cdouble a[], int n, cdouble y[], cdouble w){
    if (n == 1){
        y[0] = a[0];
        return;
    }

    int k = n / 2;
    cdouble *a0 = new cdouble[k], *a1 = new cdouble[k];
    cdouble *y0 = new cdouble[k], *y1 = new cdouble[k];

```

```

for (int i = 0; i < k; i++){
    a0[i] = a[i * 2];
    a1[i] = a[i * 2 + 1];
}
FFT(a0, k, y0, w * w);
FFT(a1, k, y1, w * w);

cdouble x = 1, t; // x = w ^ i
for (int i = 0; i < k; i++){
    t = x * y1[i];
    y[i] = y0[i] + t;
    y[i+k] = y0[i] - t;
    x = x * w;
}

delete []a0; delete []a1;
delete []y0; delete []y1;
}

void c2p(cdouble a[], int n, cdouble y[]){
    //功能: 将多项式由系数表示转化为点表示
    //参数: a[]: 多项式系数, n: 系数个数, y[]: 计算出的点值
    //要求: n 是 2 的整数幂
    FFT(a, n, y, cdouble(cos(2 * pi / n), sin(2 * pi / n)));
}

void p2c(cdouble y[], int n, cdouble a[]){
    //功能: 将多项式由点表示法转化为系数表示
    //参数: y[]: 提供的点值, n: y 的长度, a[]: 计算出的多项式系数
    //要求: n 是 2 的整数幂
    FFT(y, n, a, cdouble(cos(-2 * pi / n), sin(-2 * pi / n)));
    for (int i = 0; i < n; i++){
        a[i] /= n;
    }
}

int cpmul(cdouble a[], int n, cdouble b[], int m, cdouble c[]){
    //功能: 复系数多项式 AB 相乘, 结果为 C
    //参数: a[]多项式 A 系数, n 系数个数; b[]多项式 B 系数, m 系数个数
    int nn = 1;
    while (nn < n + m) nn *= 2;
    cdouble *aa = new cdouble[nn], *bb = new cdouble[nn], *cc = new cdouble[nn];
    cdouble *ya = new cdouble[nn], *yb = new cdouble[nn], *yc = new cdouble[nn];

```

```

for (int i = 0; i < n; i++) aa[i] = a[i];
for (int i = m; i < nn; i++) aa[i] = 0;
for (int i = 0; i < m; i++) bb[i] = b[i];
for (int i = n; i < nn; i++) bb[i] = 0;

c2p(aa, nn, ya); c2p(bb, nn, yb);
for (int i = 0; i < nn; i++) yc[i] = ya[i] * yb[i];
p2c(yc, nn, cc);

for (int i = 0; i < n+m-1; i++) c[i] = cc[i];

delete []aa; delete []bb; delete []cc;
delete []ya; delete []yb; delete []yc;

return n+m-1;
}

```

## 多项式除法

```

void cpdiv(cdouble a[], int n, cdouble b[], int m, cdouble q[], cdouble r[]){
    //功能：复系数多项式 AB 相相除，商为 Q，余数为 R
    //参数： a[]多项式 A 系数，n 系数个数； b[]多项式 B 系数，m 系数个数
    //要求： 数组 r[]大小>=n
    //其中 Q 系数个数 k=max(n-m+1, 0)
    int k = n - m + 1;
    for (int i = 0; i < n; i++) r[i] = a[i];
    for (int i = 0; i < k; i++) q[i] = 0.0;
    assert(abs(b[m-1]) > eps);

    for (int i = 0; i < k; i++){
        int kk = k - i - 1;
        int jj = n - i - 1;
        q[kk] = r[jj] / b[m-1];
        r[jj] = 0;
        for (int j = 0; j < m-1; j++){ //注意是 < m-1
            r[j+kk] -= q[kk] * b[j];
        }
    }
}

```

# 数据类型

## 高精度正整数

已经通过 bhoj1001, bhoj1063, bnu4119(+), poj3982(+), bnu4120(-), hdu1042(n!)

```
typedef long long int64;
const int Base = 1000000000; //1e9
const int BaseDigit = 9;
const int Capacity = 500;
const int BufferSize = 500;
char buf[BufferSize];

struct bigint {
    int Len;
    int Data[Capacity];
    bigint() : Len(0) {}
    bigint(const bigint& A) : Len(A.Len) { //O(A.Len)
        memcpy(Data, A.Data, Len * sizeof(*Data));
    }
    bigint(int A) : Len(0) {
        for (; A > 0; A /= Base) Data[Len++] = A % Base;
    }
    bigint(char *s){
        Parse(s);
    }
    bigint& operator=(const bigint& A) { //O(A.Len)
        Len = A.Len;
        memcpy(Data, A.Data, Len * sizeof(*Data));
        return *this;
    }
    int& operator[](int i) { return Data[i];}
    int operator[](int i) const { return Data[i]; }
    bigint &Parse(const char s[]){
        static char buf[Capacity];
        if (strcmp(s, "0") == 0){
            Len = 0;
            return *this;
        }
        strcpy(buf, s);
        int sLen = strlen(buf), i;
        Len = sLen / BaseDigit + bool( sLen % BaseDigit);
        char *p = buf + sLen - BaseDigit;
```

```

        for (i = 0; i < Len-1; i++){
            sscanf(p, "%d", &Data[i]);
            p[0] = 0;
            p -= BaseDigit;
        }
        sscanf(buf, "%d", &Data[Len-1]);
        return *this;
    }

    char *toString(char *buf) const{
        char *p; int i;
        if (Len == 0){
            sprintf(buf, "0");
        }
        else{
            sprintf(buf, "%d", Data[Len - 1]);
            p = buf + strlen(buf);
            for (i = Len - 2; i >= 0; i--){
                sprintf(p, "%0*d", BaseDigit, Data[i]);
                p += BaseDigit;
            }
        }
        return buf;
    }

    char *toString() const{
        static char buf[BufferSize];
        return toString(buf);
    }
};

int read(bigint &A){
    if (scanf("%s", buf) == EOF) return 0;
    A.Parse(buf);
    return 1;
}

int writeln(const bigint& A){
    return printf("%s\n", A.toString());
}

bigint operator+(const bigint& A, const bigint& B) {
    bigint R;
    int i, Carry = 0;
    for (i = 0; i < A.Len || i < B.Len || Carry > 0; i++) {
        if (i < A.Len) Carry += A[i];

```

```

        if (i < B.Len) Carry += B[i];
        R[i] = Carry % Base;
        Carry /= Base;
    }
    R.Len = i;
    return R;
}

bigint operator-(const bigint& A, const bigint& B) {
    bigint R;
    int i, Carry = 0;
    R.Len = A.Len;
    for (i = 0; i < R.Len; i++) {
        R[i] = A[i] - Carry;
        if (i < B.Len) R[i] -= B[i];
        if (R[i] < 0) Carry = 1, R[i] += Base; else Carry = 0;
    }
    while (R.Len > 0 && R[R.Len - 1] == 0) R.Len--;
    return R;
}

bigint operator*(const bigint& A, int B) {
    int i; bigint R;
    if (B == 0) return 0;
    int64 Carry = 0;
    for (i = 0; i < A.Len || Carry > 0; i++) {
        if (i < A.Len) Carry += int64(A[i]) * B;
        R[i] = Carry % Base;
        Carry /= Base;
    }
    R.Len = i;
    return R;
}

bigint operator*(const bigint& A, const bigint& B) { //复杂度 O(A.Len * B.Len)
    int i, j; bigint R;
    if (B.Len == 0) return 0;
    for (i = 0; i < A.Len; i++) {
        int64 Carry = 0;
        for (j = 0; j < B.Len || Carry > 0; j++) {
            if (j < B.Len) Carry += int64(A[i]) * B[j];
            if (i + j < R.Len) Carry += R[i + j];
            if (i + j == R.Len) R[R.Len++] = Carry % Base; else R[i + j] = Carry % Base;
            Carry /= Base;
        }
    }
    return R;
}

```

```

        }
    }

    return R;
}

bigint operator/(const bigint& A, int B) { //复杂度 O(A.Len)
    bigint R; int i;
    int64 C = 0;
    for (i = A.Len - 1; i >= 0; i--) {
        C = C * Base + A[i];
        R[i] = C / B;
        C %= B;
    }
    R.Len = A.Len;
    while (R.Len > 0 && R[R.Len - 1] == 0) R.Len--;
    return R;
}

int operator%(const bigint& A, int B) { //复杂度 O(A.Len)
    int i; int64 C = 0;
    for (i = A.Len - 1; i >= 0; i--) {
        C = C * Base + A[i]; C %= B;
    }
    return C;
}

int cmp(const bigint& A, const bigint& B) {
    int i;
    if (A.Len != B.Len) return A.Len > B.Len ? 1 : -1;
    for (i = A.Len - 1; i >= 0 && A[i] == B[i]; i--);
    if (i < 0) return 0;
    return A[i] > B[i] ? 1 : -1;
}

bigint operator/(const bigint& A, const bigint& B) { //复杂度 O(32 * A.len * B.len)
    bigint R, Carry = 0;
    int Left, Right, Mid;
    for (int i = A.Len - 1; i >= 0; i--) {
        Carry = Carry * Base + A[i];
        Left = 0;
        Right = Base - 1;
        while (Left < Right) {
            Mid = (Left + Right + 1) >> 1;
            if (cmp(B * Mid, Carry) <= 0) Left = Mid; else Right = Mid - 1;
            Carry -= B * Mid;
        }
        R[i] = Carry;
    }
    return R;
}

```

```

    }
    R[i] = Left;
    Carry = Carry - B * Left;
}
R.Len = A.Len;
while (R.Len > 0 && R[R.Len - 1] == 0) R.Len--;
return R;
}

bigint operator%(const bigint& A, const bigint& B) {
    //只需要 copy 除法的函数并且把 return 的改为 Carry 即可,复杂度 O(32 * A.len * B.len)
    bigint Carry = 0;
    int Left, Right, Mid;
    for (int i = A.Len - 1; i >= 0; i--) {
        Carry = Carry * Base + A[i];
        Left = 0; Right = Base - 1;
        while (Left < Right) {
            Mid = (Left + Right + 1) >> 1;
            if (cmp(B * Mid, Carry) <= 0) Left = Mid;
            else Right = Mid - 1;
        }
        Carry = Carry - B * Left;
    }
    return Carry;
}

bigint sqrt(const bigint& A) { //复杂度 O(32 * A.Len * A.len * A.len)
    //已经通过 http://acm.hrbeu.edu.cn/index.php?act=problem&id=1003&cid=16
    bigint Left = 0, Right = A, Mid, Ret;
    while (cmp(Left, Right) <= 0){
        Mid = (Left + Right) / 2;
        int c = cmp(Mid * Mid, A);
        if (c <= 0){
            Left = Mid + 1; Ret = Mid;
        } else {
            Right = Mid - 1;
        }
    }
    return Ret;
}

istream& operator>>(istream& cin, bigint& A) {
    char ch;
    if (cin.flags() & ios::dec){ //十进制

```

```

for (A = 0; cin >> ch;) {
    A = A * 10 + (ch - '0');
    if (cin.peek() <= ' ') break;
}
} else if (cin.flags() & ios::oct){ //八进制, 复制时只需要把 10 改成 8
    for (A = 0; cin >> ch;) {
        A = A * 8 + (ch - '0');
        if (cin.peek() <= ' ') break;
    }
} else if(cin.flags() & ios::hex){
    for (A = 0; cin >> ch;) {
        A = A * 16 + (ch > '9' ? toupper(ch) - 'A' + 10 : ch - '0');
        if (cin.peek() <= ' ') break;
    }
}
return cin;
}

```

```

ostream& operator<<(ostream& cout, const bigint& A) {
    int i, j;
    if (cin.flags() & ios::dec){
        cout << (A.Len == 0 ? 0 : A[A.Len - 1]);
        for (i = A.Len - 2; i >= 0; i--)
            for (j = Base / 10; j > 0; j /= 10)
                cout << A[i] / j % 10;
    }
    else {
        stack<int> st;
        bigint tmp(A);
        if (cin.flags() & ios::oct){
            if (tmp.Len == 0) cout << 0;
            else{
                while (tmp.Len){
                    st.push(tmp % 8); tmp = tmp / 8;
                }
            }
        } else if(cin.flags() & ios::hex){
            while (tmp.Len){
                st.push(tmp % 16); tmp = tmp / 16;
            }
        }
        while (!st.empty()){
            cout << "0123456789ABCDEF"[st.top()]; st.pop();
        }
    }
}

```

```

    }
    return cout; //千万别忘了加这句!!!
}

```

## 高精度实数

```

const int maxExp = 100;;

struct bigdec{
    mutable bigint Data;
    mutable int Sign, Exp; //这里 Exp 中 1 相当于 BaseDigit
    bigdec(){}
    bigdec(int a, int e = 0){
        if (a > 0) { Data = a; Sign = 1; }
        if (a == 0) {Data = 0; Sign = 0;}
        if (a < 0) { Data = -a; Sign = -1;}
        Exp = e;
        if (Exp > maxExp) ShiftRight(*this, Exp - maxExp);
    }
    bigdec(const bigint& A, int s, int e = 0){
        Data = A; Exp = e;
        if (A.Len == 0) Sign = 0; else Sign = s;
    }
    char *toString() const{
        static char buf[Capacity];
        char *p = buf; int i;
        if (Sign == 0){
            strcpy(p, "0"); return buf;
        }
        if (Sign < 0) *p++ = '-';
        if (Exp >= 0){
            p += sprintf(p, "%s", Data.toString());
            for(i = Exp;i--;) p += sprintf(p, "%0*d", BaseDigit, 0);
            return buf;
        }
        if (-Exp >= Data.Len){
            p += sprintf(p, "0.");
            for(i = -Exp-1;i >=Data.Len;i--) p += sprintf(p, "%0*d", BaseDigit, 0);
            for(;i >= 0;i--) p += sprintf(p, "%0*d", BaseDigit, Data[i]);
        }
        else{
            p += sprintf(p, "%d", Data[Data.Len-1]);
            for (i = Data.Len-2; i >= 0; i--){
                if (i == -Exp-1) p += sprintf(p, ".");
            }
        }
    }
}
```

```

        p += sprintf(p, "%0*d", BaseDigit, Data[i]);
    }
}

while (p[-1] == '0') p--;
if (p[-1] == '.') p--;
p[0] = 0;
return buf;
}

bigdec &Parse(char *s){
    static char buf[Capacity];
    Sign = 1;
    while (s[0] == '+' || s[0] == '-') {
        if (s[0] == '-') Sign = -Sign; s++;
    }
    strcpy(buf, s);
    char *p = buf, *e = buf + strlen(s);
    for (; *p != 0; p++)
        if (*p == '.') break;
    if (*p != 0){
        *p++ = 0;
        while ((e - p) % BaseDigit != 0)
            *e++ = '0';
        *e = 0;
    }
    return *this = bigdec(bigint(buf), Sign, 0) + bigdec(bigint(p), Sign, -(e - p) / BaseDigit);
}

friend void ShiftRight(const bigdec &A, int s);
friend bigdec operator+ (const bigdec& A, const bigdec& B);
};

int read(bigdec &A){
    if (scanf("%s", buf) == EOF) return 0;
    A.Parse(buf);
    return 1;
}

int writeln(const bigdec &A){ return printf("%s\n", A.toString());}

void ShiftLeft(const bigdec &A, int s){
    int i;
    A.Data.Len += s; A.Exp -= s;
    for (i = A.Data.Len - 1; i >= s; i--)
        A.Data[i] = A.Data[i-s];
    while (i >= 0)

```

```

        A.Data[i--] = 0;
    }

void ShiftRight(const bigdec &A, int s){
    int i;
    A.Data.Len -= s; A.Exp += s;
    for (i = 0; i < A.Data.Len; i++)
        A.Data[i] = A.Data[i+s];
}

bigdec operator +(const bigdec &A, const bigdec &B){
    if (A.Sign == 0) return B; if (B.Sign == 0) return A;
    bigdec C;
    if (A.Exp > B.Exp) ShiftLeft(A, A.Exp - B.Exp);
    if (A.Exp < B.Exp) ShiftLeft(B, B.Exp - A.Exp);
    C.Exp = A.Exp;
    if (A.Sign == B.Sign){
        C.Sign = A.Sign; C.Data = A.Data + B.Data;
    } else{ // A.Sign != B.Sign
        if (cmp(A.Data, B.Data) == 0) C.Sign = 0;
        else if (cmp(A.Data, B.Data) > 0){
            C.Sign = A.Sign; C.Data = A.Data - B.Data;
        } else{ // cmp(A, B) < 0
            C.Sign = B.Sign; C.Data = B.Data - A.Data;
        }
    }
    if (C.Exp > maxExp) ShiftRight(C, C.Exp - maxExp);
    return C;
}

bigdec operator -(bigdec &A, bigdec &B){
    B.Sign = -B.Sign;
    bigdec C = A + B;
    B.Sign = -B.Sign;
    return C;
}

bigdec operator *(bigdec &A, bigdec &B){
    if (A.Sign == 0 || B.Sign == 0) return 0;
    bigdec C;
    C.Sign = A.Sign * B.Sign; C.Data = A.Data * B.Data; C.Exp = A.Exp + B.Exp;
    if (C.Exp > maxExp) ShiftRight(C, C.Exp - maxExp);
    return C;
}

```

## 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

```
#define gcd(a, b) __gcd(a, b)
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){
        if (f.a == 0) cout << 0;
        else if (f.b == 1) cout << f.a;
        else cout << f.a << "/" << 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

```

//#define COMPLEX_ELE
const int maxn = 100;

#ifdef COMPLEX_ELE
    typedef complex<double> Tvalue;
#else
    typedef double Tvalue;
#endif
typedef double Tabs;
const Tabs eps = 1e-8;
#define sqr(x) ((x)*(x))

ostream& operator<< (ostream &cout, complex<double> c){
    int width = cout.width();
    cout << setw(0) << "(";
    cout << setw(width) << (abs(c.real()) < eps ? 0 : c.real());
    cout << ",";
    cout << setw(width) << (abs(c.imag()) < eps ? 0 : c.imag());
    cout << ")";
    return cout;
}

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;
}
};

istream &operator >>(istream &cin, mat &a){
    cin >> a.m >> a.n;
    for (int i = 0; i < a.m; i++){
        for (int j = 0; j < a.n; j++){
            cin >> a[i][j];
        }
    }
    return cin;
}

ostream &operator <<(ostream &cout, const mat &a){
    for (int i = 0; i < a.m; i++){
        for (int j = 0; j < a.n; j++){
            cout << setw(10);
            cout << (abs(a[i][j]) < eps ? 0 : a[i][j]) << "\t";
        }
        cout << endl;
    }
    return cout;
}

mat operator* (const mat& a, const mat& b){
    //时间复杂度 O(n ^ 3)
    mat c;
    assert(a.n == b.m);
}

```

```

c.m = a.m; c.n = b.n;
memset(c.data, 0, sizeof c.data);
for (int i = 0; i < c.m; i++)
    for (int k = 0; k < a.n; k++)
        if (a[i][k]) //超强剪枝
            for (int j = 0; j < c.n; j++)
                c[i][j] += a[i][k] * b[k][j];
return c;
}

mat transpose(const mat& a){
    mat r;
    r.m = a.n; r.n = a.m;
    for (int i = 0; i < a.m; i++){
        for (int j = 0; j < a.n; j++){
            r[j][i] = a[i][j];
        }
    }
    return r;
}

mat &transpose_self(mat &a){
    swap(a.m, a.n);
    if (a.m < a.n){
        for (int i = 0; i < a.m; i++){
            for (int j = i+1; j < a.n; j++){
                swap(a[i][j], a[j][i]);
            }
        }
    }
    else{
        for (int j = 0; j < a.n; j++){
            for (int i = j+1; i < a.m; i++){
                swap(a[i][j], a[j][i]);
            }
        }
    }
    return a;
}

int rank(mat a){
    //求矩阵的秩，时间复杂度 O(n ^ 3)
    int p = min(a.m, a.n);
    int rank = 0;
}

```

```

int 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)
    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];
static int 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) continue;
        for (int j = 0; j < n; j++){
            if (j == k) continue;
            a[i][j] -= a[k][j] * a[i][k];
        }
    }
    for (int i = 0; i < n; i++){
        if (i == k) continue;
        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;
    int is;
    static int 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;
    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) continue;
    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 sqrt_s = sqrt(fabs(s));
    if (s > 0.0){
        if (b > 0.0) z[0] = cpl((b + sqrt_s) * 0.5, 0);
        else z[0] = cpl((b - sqrt_s) * 0.5, 0);
        z[1] = b - z[0];
    }
    else{
        z[0] = cpl(b * 0.5, sqrt_s * 0.5);
        z[1] = conj(z[0]); //x[1]为 x[0]的共轭
    }
    return 1;
}
int is1 = 0;
int is2 = 0;
int 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;
Tvalue 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

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

```

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

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

```

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)

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

```

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

LL det\_mod(mat a, LL m){ //O(n ^ 3 log(m)), 已经通过 spoj DETER3

```

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*/

```

求  $E + A + A^2 + \dots + A^k$

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

# 线性规划

## 线性规划对偶问题

$$\begin{aligned} \min f &= c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq (=, \geq) b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \leq (=, \geq) b_2 \\ \vdots \quad \vdots \quad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq (=, \geq) b_m \\ x_j \geq 0 (\leq 0, 或 符号不限) j = 1 \sim n \end{array} \right. \end{aligned}$$

对偶问题对应表

$$\begin{aligned} \max z &= b_1y_1 + b_2y_2 + \cdots + b_my_m \\ \left\{ \begin{array}{l} a_{11}y_1 + a_{12}y_2 + \cdots + a_{1n}y_m \leq (\geq, =) c_1 \\ a_{21}y_1 + a_{22}y_2 + \cdots + a_{2n}y_m \leq (\geq, =) c_2 \\ \vdots \quad \vdots \quad \vdots \\ a_{m1}y_1 + a_{m2}y_2 + \cdots + a_{mn}y_m \leq (\geq, =) c_n \\ y_j \leq 0 (\text{符号不限, 或 } \geq 0) j = 1 \sim m \end{array} \right. \end{aligned}$$

| 原问题(对偶问题)   | 对偶问题(原问题)   |
|---|---|
| 目标函数 <b>min</b>   | 目标函数 <b>max</b>   |
| 约束条件: $m$ 个<br>第 $i$ 个约束类型为 “ $\geq$ ”<br>第 $i$ 个约束类型为 “ $\leq$ ”<br>第 $i$ 个约束类型为 “ $=$ ” | 变量数: $m$ 个<br>第 $i$ 个变量 $\geq 0$<br>第 $i$ 个变量 $\leq 0$<br>第 $i$ 个变量是自由变量                  |
| 变量数: $n$ 个<br>第 $j$ 个变量 $\leq 0$<br>第 $j$ 个变量 $\geq 0$<br>第 $j$ 个变量是自由变量                  | 约束条件: $n$ 个<br>第 $j$ 个约束类型为 “ $\geq$ ”<br>第 $j$ 个约束类型为 “ $\leq$ ”<br>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  $m \times n$  行棋盘有一个  $b$ -牌的完美覆盖，当且仅当  $b|m$  或  $b|n$ 。

$m \times n (m, n \geq 2)$  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  $m \times n (m, n \geq 2)$  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) ( $0 \leq x \leq y$ ) 的最少步数

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

$t \in \{0, 1\}$  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， $\dots$ ，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）

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

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

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， $\dots$ ，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  $B = b_{n-1}b_{n-2}\dots b_1b_0$

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

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

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

$$G = B \mid (B \gg 1)$$

$n$  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

### 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

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

其中

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

最少的步数

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

算法摘自论文《A Non-recursive Algorithm for 4-Peg Hanoi Tower》

```

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

```

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  $n \geq 1$  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  $x \rightarrow n^2 - n + 1 - x$

|      |      |      |      |      |      |      |      |
|------|------|------|------|------|------|------|------|
| <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  $n (n \geq 3)$  阶方阵为  $A = [a_{ij}]$ ，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。

## 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.
```

## 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。

## 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  $\gcd(a_1, \dots, a_n) = 1$ )。

**解法:** 设  $a_1 \leq a_2 \leq \dots \leq a_n$ , 所有由  $a_1, \dots, a_n$  构成的元素集合为  $\Omega$ , 即  $\Omega = \{c_1a_1 + \dots + c_na_n \mid c_i \geq 0\}$ , 将这些数按照  $\mod a_1$  划分等价类  $R_0 \dots R_{a_1-1}$ , 每个等价类  $R_j$  求出一个最小的代表元  $r_j$

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

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,  $a_2 \% a_1, \dots, a_n \% a_1$  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

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

## 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);
    }
}

```

## 统计 1..n 的数中含 49 的数的个数

```

//HDU3555
int TransState(int state,int num){
    switch (state){
        case 0://没出现过 4
            return num == 4 ? 0;
        case 1://4xxxxxx
            if (num == 4) return 1;
            else if (num == 9) return 2;
            else return 0;
        case 2://出现过 49
            return 2;
    }
}

long long cnt49(int dig[], int len){
    //统计 1..x(不含 x)出现的含 49 的数的个数, x = dig[len..1]
    const static int maxlen = 15;
    static long long dp[maxlen][4]; //dp[i][j] 前 i 位, 状态为 j 的数字的个数
    long long ret = 0;
    int tmplen = 0; //已经处理过的位数
    int state = 0; //当前的状态
    for (int i = len; i >= 1; i--){
        tmplen++;
        for (int j = 0; j < dig[i]; j++){
            memset(dp, 0, sizeof(dp));
            dp[tmplen][TransState(state, j)] = 1;
            for (int k = tmplen + 1; k <= len; k++){
                dp[k][0] = 9 * dp[k-1][0] + 8 * dp[k-1][1];
                dp[k][1] = dp[k-1][0] + dp[k-1][1];
                dp[k][2] = dp[k-1][1] + 10 * dp[k-1][2];
            }
            ret += dp[len][2];
        }
        state = TransState(state, dig[i]);
    }
}

```

```
    }
    return ret;
}
int main()
{
    int i,j,k,t;
    long long n;
    scanf("%d",&t);
    while (t--){
        int dig[20];
        cin >> n;
        n++;
        int len = 0;
        for (;n; n /= 10){
            dig[++len] = n % 10;
        }
        cout << cnt49(dig, len) << endl;
    }
    return 0;
}
```