**20141017 UPD.**

**注意事项！——**

DB sqr(DB x){return x \* x;} 和 inline DB sqr(DB x){return x \* x;} 是 5000ms+ TLE 和 562ms AC 的区别。

**20140525 UPT**

输出浮点数的时候几个小细节

-0.0000... 的输出 。。 正确做法是判断 if(fabs(x) < EPS) x = 0.0;

输出浮点数的时候，不能用 %lf，要用 %f，而输入的时候要用 %lf。

**20140607 UPD**

实用小技巧。（x, y）逆时针旋转 45 度，x + y, y - x 。。每个点的相对位置是不变的，实际坐标差了 sin45 度 。。画个图一下就知道了。

**20140803 UPD**

在给定一个 P 考虑 i \* j <= P 的时候，分成 O(sqrt(P)) 块后，可以发现第 i 块和倒数第 i 块是首尾相应的。

**20140804 UPD**

在处理数据量大约为 1000 的组合数的时候，一般情况下使用 double 是不会影响到精度的。反而取对数容易慢 + 挂精度。

**20140813UPD**

在二分浮点数的时候用控制次数的方法要比控制精度的方法要稳得多得多得多。

这道 sb 题弄精度弄了 9 发都没过去，不是 T 就是 WA。还是控制次数要靠谱。

**DP中要注意的：**

先写出暴力DP，如果 MLE 考虑压位或者有没有简单的降维方法（如 WC2010 的切蛋糕）或者减少状态（如Jakarta B题 D <= 100）；如果 TLE 考虑单调性或者凸性的优化、以及数据结构的优化。

**简单公式——**

**欧拉定理（平面图 + 拓扑）：**

平面几何的欧拉定理 (R - r) ^ 2 = d ^ 2 + r ^ 2，R 表示外接圆半径，r 表示内接圆半径，d 表示内外接圆心距离。

对于凸多面体（圆不考虑在内，圆有单独的吊炸的公式），肯定有 V - E + F = 2，对于非凸多面体，那个常数 2, 1, 0, -1 都有可能。。

对于一个联通的平面图来说 V - E + F = 1，可以通过一棵树不断加边来进行归纳；如果有 C 个联通块，则有 V - E + F - C = 1。

**平面对偶变换：**

求两两连线斜率第 K 大，方便起见，没有相同的 x 坐标。

Key：如何把求过两点直线变成两直线交点；将点 (a, b) 变成直线 y = ax - b；于是把经过 A，B 的直线变成了直线 D(a) D(b) 的交点。

于是题目就变成了，给定 N 条直线（没有平行），求直线交点中坐标第 K 大。这个可以二分答案，判定过程就是求逆序对。

**图论——**

**类字典序的拓扑排序：**

求一个拓扑排序的序列，使得 1 优先尽量靠前，其次 2 优先尽量靠前 …… （注意与字典序的区别）

直接用 heap 拓扑排序必挂无益。例如 3 个点一条边 3->1，答案是 3 1 2，输出的缺是 2 3 1。

正确做法是倒着建图，用大根堆拓扑排序，这么做的话意思为，能不取 1 就不取 1；能不取 2 就不取 2 …… 满足题意。

给定一个拓扑图，要求加上最多的边使得仍然是拓扑图。要求边按字典序排序。

首先只要拓扑排序然后把序列中 i, j （i < j）输出就是一组答案。不过如何字典序最小？

原来和上面的题一样，只要让序列中 1 优先尽量靠前，其次 2 优先尽量靠前即可 。。

**最大流与最小割：**

1、只有满流的边才有可能是割边。不一定都是割边。

2、一条满流边的两个点在一个scc里说明可以把满流边退流，故这条边不在割边集中。

3、反之如果不在一个scc中，则该边可以在割边集中。

4、要判定一条边是不是一定在割边集中，缩点后变成一个dag，任意一个st割边都是最小割。如果两个点分别和s，t在同一个scc中，则说明一定在最小割中。

**二分图：**

有可能在二分图中最小点覆盖的点 <=> 一定在二分图最大匹配中的点；相反，一定在二分图中最大独立集中的点 <=> 不是一定在二分图最大匹配中的点。

**Havel-Hakimi定理：**

给一个无向图的所有点度，问能不能构出一个原图出来，还是无解。

每次都排序后暴力贪心，每次拿最大的度（设为D）的点和第 2,3,...D+1 个点连边；连到最后全练完了就说明有解，如果没连完就没解。

**DAG上 求 u 到 v 是否可达：**

这个直接做是 O(n^3) 的，不过是可以利用 Bitset 进行优化的，即把每个点能够到的点用 Bitset 存起来，直接 DFS，然后异或即可，O(n^3/64)。

**上下界网络流：**

<http://wenku.baidu.com/view/0f3b691c59eef8c75fbfb35c.html>

**数据结构——**

**线段树多个 Lazy 标记下传：**

三种操作，区间 1 覆盖、区间 0 覆盖、区间翻转。那么当一个区间本身有一个标记区间 1 覆盖的时候，来了个区间翻转，就变成区间 0 覆盖了，其他的类似。

看似标记之间的关系不好处理，实际上还是很好做的。如果只有加法和乘法的话，只要把每个区间看成 ax + b 的形式，然后来一个 \*p 的话就变成 pax + b；来个 +q 的话就变成 ax + (b+q)。

如果还有覆盖，首先把当前区间的乘法和加法标记都清零，接下来那么可以有两种选择。一，可以直接在覆盖上做打标记；二、可以选择在下放的时候先下放覆盖标记（应该是这种更好一些），再下放乘法加法标记。

**后缀数组：**

一、做后缀数组的时候一定要加上最后一个字符，而且最后一个字符一定要是 \0 ，它可以很好的控制不等长串之间的字典序大小。

二、在 contencate 两个串的时候，中间要用一个完全没有出现过的符号隔开，当然不能用 \0 ，一般用 $ 是比较好的。

**分块：遇到数据结构题，对于 lgn 的想法想不出来，多想想分块是否可以通过！**

一、题目大意：500\*500的矩阵，6e4 次询问一个子矩阵问第 k 大元素。

有一个简单的分块做法。把矩阵里的元素从大到小排序，用 2d 树状数组插入。每插入 500 个对所有询问查询一次。对于一个询问，如果在该区间内，则暴力往回数 n 个。这样做时间复杂度是有保证的。

二、题目大意：给出一堆离散的集合，总长度 <=N，修改 + 求和。

以 sqrtN 为分界分块。

1、小集合的查询，直接求和原序列的数，以及和每个大集合的相交个数\*对应的 delta 值：O(N^0.5)

2、大集合的查询，直接调去 ans 值：O(1)

修改操作：每次修改操作都要修改每个（N^0.5个）大集合的 ans 值：O(N^0.5)

3、小集合的修改，直接在原序列上修改、O(N^0.5)

4、大集合的修改，直接在 delta 值上改O(1)

**CDQ分治，详细代码见模板：**

前面的对后面的有影响，则可以用分治，考虑对于每个区间，前半部分对后半部分的影响，分治的复杂度是nlgn，排序或者再套用数据结构维护也就是nlgnlgn的级别。

对于边修改 + 边查询的题目，只要不强制在线我们可以对时间戳分治。

**Splay Tree：**

一、在最初建树的过程中不要图方便给建成一个链。。不如去写一个类似线段树那样 build(l, r, mid) 函数。。上来直接给弄成平衡的。。

二、最好手写一个 find(x) 函数。。用来返回排在第 x 的节点值。。find 后立即提到根节点。。

三、在某点后面插入一个节点时，先将该节点提到根，然后裸插就行，分类讨论该点右子是否为 Null

四、删除一个节点时，先将该节点提到根，分类讨论左子是否为 Null；在删除完节点后千万别让整棵树的 sz -- 。。因为整棵树的 sz 作用只在插入新节点的时候给节点编号起了作用。。删除节点后只是让这个节点和整个树都脱离了联系。。并不是真正意义上的删除。

五、在进行操作的过程中，随时注意哪个点时根。。

六、想要把某个点提到根的左子、右子上，只需先把根和那颗子树的边去掉；然后直接 Splay 那个点就行，最后再维护一下根和该点就好。

七、0 节点的父亲和 0 节点的 sz 值要时刻保持为 0 。。否则一定不对。

八、别忘了加上两个虚拟点；不然在询问区间设计到两头的是无法处理的。。进行询问的时候先把闭区间变成开区间，把开区间左端点转到根，右端点转到根的右子，直接返回右端点的左子树的答案就行。

九、标记下传的时候和线段树是类似的，在某个点有标记的时候该点的值已经维护完毕；下传时需修改子节点的标记值和维护值；下传的时候在 update 和 splay 的时候。

**树状数组，1D2D区间修改区间查询，代码见模板：**

1、单点修改+任意区间查询。[l,r] = [1,r] - [1,l-1]。这个技巧下面每一个都会用到。

2、区间修改+单点查询。做法是把原序列弄成一次差分数列，然后单点查询就是求前缀和了。

3、区间修改+区间查询。首先把区间用1、的技巧拆开（这一步是必要的）做法还是弄成一次查分序列，按照单点查询的方法来查询一个区间（就是多个点的和）。。第 i 个数被加了(n-i+1)次，然后把 i\*d[i] 做为一个新的量来维护。两个树状数组就可以完成操作。

现在来考虑二维的

4、单点修改+区间查询。[x0,y0,x1,y1] = [1,1,x1,y1] - [1,1,x0-1,y1] - [1,1,x1,y0-1] + [1,1,x0-1,y0-1]。

5、区间修改+单点查询。把修改区间 [x0,y0,x1,y1] 变成修改点+[x0,y0] - [x0,y1+1] - [x1+1,y0] + [x1+1,y1+1]。

6、区间修改+区间查询。把上面两个技巧一起用，把查询区间用4、的技巧拆开，归为四个左上角为[1,1]的区间。查一个区间就相当于查这个区间的所有单点，对每个这样的区间分别考虑区间内每个点被计算了多少次，展开式子用三个树状数组来维护就行。

**动态规划——**

**求期望的姿势！！！：**

因为期望具有独立性，所以可以把单独的期望算出来然后相加。<-- 这点非常重要，一筹莫展时八成就是这个了。

具体题目，比如 TC 最近那道 500，说的是联通块个数，其实只要找到形成联通快的充要条件，然后就可以进行单独计算然后求和了。

再比如多校那场 DAG 的题目，对于每个点考虑被选中的期望，相加即可。

再比如某场 CF 的 D 题，说是有 100 个点，其实每个点都是独立的，只要考虑 1 个点然后乘以 100 即可。

**博弈DP：**

博弈 dp 的时候，肯定是 dag。

有两个人的话，没有必要去分来讨论。在状态里记录一个步数，然后直接当成一个人来转移。

比如必胜的人想要赶快赢，必输的人想要慢点输，可以不分类讨论地写。

**斜率优化：**

1、较优的决策点对后续状态影响具有持续性，简单来说就是如果在状态 I 处，选择了 J 作为决策点，那在 I+1 处的决策点一定不可能在J之前。

2、把DP方程写出来，左边化为J，K的形式，右边化为I的形式；如果有单调性的话，就可以进行队首队尾的维护了。

3、详细见代码模板

**花式DP：**

牡丹江网络赛，50 \* 20w 的二位数组 bool 背包。直接做是 400 \* 50 \* 20w 的 DP，明显通过不了。

不过注意到 50 这个数字很小！我们可以定义一个 vis[20w]，vis[i] 表示第二种数加起来得 i，第一种数加起来得 j 的状态压缩！可以压成一个 long long，然后转移就是 O(1) 的了。可以通过二进制的 or 和 << 操作直接把 50 给优化掉。

然后就是记录路径，同样可以用二进制来搞，然后要开一个 50\*20w 的数组来表示状态从哪里来。记录路径的时候会涉及到找到一个二进制数的最低位是第几位，这个操作均摊是 O(n) 的。

**矩阵DP：**

有时候矩阵只能够被运用到一部分的 DP 过程中。

**组合数学——**

**README：**

**组合的题目在一筹莫展时，可以考虑是否有能够一一对应的简单版！**

比较复杂的组合题目推导的时候思维要清晰，把思路写纸上的话会好不少，不然非常容易乱成一锅粥。

在遇到一个题目感觉很复杂的时候，看看有没有非常巧妙的转换让它变得很简单。（参见某场CF的D，两个乌龟从矩形两个对角爬）

**求 sum\_now[i] = Σ sum\_pre[j] (i & j = i)：**

for(int i = 0; i < 20; i ++)

for(int j = 0; j < (1 << 20); j ++)

if(j >> i & 1){

sum[j - (1 << i)] += sum[j];

}

**枚举子集：**

for(int i=(sta&(sta-1));i;i=(i-1)&sta)

**经典题目：**

BZOJ 1079：

K <= 15 个颜色不同的油漆，每个颜色够涂 Ci <= 5 块，要求相邻的砖块颜色不能相等，求方案数。

注意到 Ci <= 5，可以设计状态 dp[a][b][c][d][e][num]，然后用乘法原理 DP。

HDU 4532：

为上面题目的加强版，K <= 47，Ci <= 47，ΣCi <= 447。

f[i][j] 表示前 i 个颜色，有 j 个相邻的（即不合法空隙）。枚举第 i+1 个颜色的分组数 p，枚举 k 个组进入不合法空隙，则 p-k 个组进入合法空隙。全部用组合数乘起来，过程中要用下插板法。

理论复杂度 O(47\*447\*47\*47)，实际肯定也不会有这么大。

TC SRM 597 Div1 1000。

究极加强版。。K = 3，Ci <= 10^5。

当然这个就得利用 K = 3 的性质来做。设三个颜色分别为 X, Y, Z 。枚举放在第一个地方的颜色。假设为 X 。把 Y, Z 往空里插。（枚举差不插入最后一个空）枚举 Y, Z 的插入方式。分奇偶讨论。。解方程。比较复杂。

**卡特兰数：**

基本形态，n个左括号和n个右括号的合法括号序列；矩阵连乘凸多边形三角划分……

递推式：Catalan(n) = ΣCatalan(i) \* Catalan(n-1-i)、（0<=i<=n-1）；Catalan(0) = Catalan(1) = 1。通项 Catalan(n) = C(2n,n)/(n+1)。

证明：用总共括号序列减去非法括号序列，可以证明非法括号序列和n+1个+1、n-1个-1的括号序列是一一映射（先满射后单射），即 C(2n,n+1)，做减法即可

有了通项就可以写出更一般的递推式：Catalan(n) = Catalan(n-1) \* (4n-2) / (n+1)。

广义卡特兰序列，对于 n >= m 的方格，每一步都要满足 x >= y。

显然可以用同样地做法。答案是 C(n+m,n) - C(n+m,m-1)、而且可以写成一个更简单的、只带一个组合数的形式，从而在线性时间内计算；这种 “对称” 的思想 感觉还是蛮重要的。

**生成树计数（完全不懂原理）：**

完全图的生成树个数：n^(n-2)

G的度数矩阵D[G]是一个 n\*n 的矩阵，并且满足：当i≠j时,dij=0；当i=j时，dij等于vi的度数。

G的邻接矩阵A[G]也是一个 n\* n的矩阵，并且满足如果vi、vj之间有边直接相连，则Aij=1，否则为0。

我们定义 G 的 Kirchhoff 矩阵 C[G] = D[G] - A[G]，则 Matrix-Tree 定理可以描述为：G 的所有不同的生成树的个数等于其 Kirchhoff 矩阵 C[G] 任何一个 n-1 阶主子式的行列式的绝对值。所谓 n-1 阶主子式，就是对于 r(1 <= r <=n)，将 C[G] 的第 r 行、第 r 列同时去掉后得到的新矩阵，用Cr[G]表示。

**数论——**

**a^b mod c，欧拉函数：**

欧拉函数有个性质，就是对于两个互质的数a和p，a^phi(p) ≡ 1 (mod p)。

如果有a，c互质的话，答案显然就是a^b mod c = a^(b mod phi(c)) % c

但是如果没有a，c互质的话，就需要用这个公式

a^b mod c = a^(b mod phi(c) + phi(c)) % c ，仅在b>=phi(c)时才成立。

**带模除法：**

0、逆元，使用条件是模数为质数。

1、如果a/b是整数然后要求(a/b)%c，可以直接算(a%(b\*c))/b。证明很简单。

这种用法是用来处理取余的那个数不是一个质数的时候。而且只进行一次除法。

注意如果 b \* c 即新的取模的数达到了long long的范围，那之前的乘法就可能是long long和long long相乘，最后就直接爆出去了。所以这个时候在进行乘法的时候应该考虑用快速加法，就是logn的模拟加法。

2、当然还有一种处理方式。就是如果取模的数并不是质数，或者除数与取模的数并不互质（当然也就没有逆元直说），此时可以将取模的数进行分解质因数处理，对每个质因数分别算出答案，然后再用中国剩余定理把它们合并回去。

**快速 O(n) 处理 1~p-1 的逆元：**

inv[P mod i]\*(P mod i) mod P = 1

inv[P mod i]\*(P-P/i\*i) mod P = 1

inv[P mod i]\*i\*(-P/i) mod P = 1

i\*(P-P/i)\*inv[P mod i] modP = 1

一个 for 循环就搞定了。

**斐波那契相关：**

1.gcd(fib(n),fib(m))=fib(gcd(n,m))

Fibonacci 素数：和 Fibonacci 序列中比它小的数都互质的数称作 Fibonacci 素数。。注意 Fibonacci 素数并不一定都是素数。。但是除了前几项以外斐波那契的素数项的都是斐波那契素数。。

2.如果fib(k)能被x整除，则fib(k\*i)都可以被x整除。

3.f(0)+f(1)+f(2)+…+f(n)=f(n+2)-1

4.f(1)+f(3)+f(5)+…+f(2n-1)=f(2n)

5.f(2)+f(4)+f(6)+…+f(2n) =f(2n+1)-1

6.[f(0)]^2+[f(1)]^2+…+[f(n)]^2=f(n)·f(n+1)

7.f(0)-f(1)+f(2)-…+(-1)^n·f(n)=(-1)^n·[f(n+1)-f(n)]+1

8.f(m+n)=f(m-1)·f(n-1)+f(m)·f(n)

9.[f(n)]^2=(-1)^(n-1)+f(n-1)·f(n+1)

10.f(2n-1)=[f(n)]^2-[f(n-2)]^2

11.3f(n)=f(n+2)+f(n-2)

12.f(2n-2m-2)[f(2n)+f(2n+2)]=f(2m+2)+f(4n-2m) [ n>m≥-1,且n≥1]

**Lucas定理，详细见代码模板：**

C(n,m)%p ≡ C(n%p, m%p) \* C(n/p,m/p) % p

其中定义C(n,0) = 1, C(i,j) = 0 (i<j)。

注意在另求 C(x,y) 的时候要一个一个去乘，因为预处理阶乘在处理组合数上是有问题的。

Lucas定理的另外一种形式，更直观来说，就是把n和m写成p进制，然后取各自对应的位来得到组合数进行乘法。

**数值分析，论如何处理精度：**

首先分母必须不能取 0 ，这是要时刻回避的。

其次要尽可能地避免特别大的数和特别小的数相乘，虽然乘出来的可能是一个比较中和的值但是这之中就很容易产生误差，也要避免。

在数据输入的部分就把一些极限情况给特判掉，这是非常必要的。

还有时候取对数也可以把 “大数与小数相乘” 给化简掉，当然前提是并不要求一个完全精确的值。。

**拉格朗日插值法：**

对某个多项式函数，已知有给定的k + 1个取值点。

假设任意两个不同的自变量对应的函数值都互不相同，那么应用拉格朗日插值公式所得到的拉格朗日插值多项式为

L(x) = Σ yi \* li(x)

其中每个为拉格朗日基本多项式（或称插值基函数），其表达式为：

li(x) = Π (x-xi) / (xj-xi) (i≠j)

拉格朗日基本多项式 L(i) 的特点是在 Xi 上取值为1，在其它的点 Xj ( j≠i ) 上取值为0。

**Burnside定理：**

不同等价类的个数 = 每个置换对应的（在当前置换下保持不变的方案数，即不动点）的个数之和 / 总的置换个数

**小技巧：**

遇到 Lcm 。。 99% 要转化成 Gcd 来处理。。

组合数取模。。

如果底数特别大，且 mod 是可因式分解成多个质数的一次方相乘的（这些质数们不能特别大。。），可以用 Lucas 定理 + CRT 合并。。复杂度大概是 O(lgN\*P)。。

如果 mod 是任意给定的，则底数必定不会太大。由于组合数一定是正整数，记录每个质数的幂即可，最后快速幂。。复杂度是 O(NlgN) 。。

底数特别大 + mod 任意给定貌似不可做。。

**积性函数（呵呵）：**

n = ∏ pi^ai -> f(n) = ∏ f(pi^ai) 。。为积性函数。。

n = ∏ pi^ai -> f(n) = ∏ f^ai(pi) 。。为完全积性函数。。

一般来说非完全积性函数比较常见。。而且在计算的时候其实复杂度基本是一样的，因为积性函数都有一些其他的性质。。

常见的积性函数。。欧拉函数 φ 。。莫比乌斯函数 μ 。。正约数数目 d。。正约数之和 σ。。因子函数 σk，即所有约数的 k 次幂之和。。这些函数都是可以通过 线性筛法 求出来的。。比如在求 d 和 σ 的时候要记录 最小质因子 的数目（。。求 σ 的时候线性筛目测效率不高。。）。。不过为了求稳不如不在迫不得已时先写 枚举约数 的筛吧。。

常见的完全积性函数。。单位函数 id 。。对于狄利克雷卷积的乘法单位 e (ε) 。。在数论变换有很重要的作用。。

其中非常重要的变换 N = Σ φ(d) d|n ; e(N) = Σ μ(d) d|n 。。前者考虑每个数与 n 的 gcd。。后者二项式定理。。

对于两个数论函数 f、g。。

(f\*g)(N) = f(N) \* g(N)

(f×g)(N) = ∑ f(d) \* g(N/d) (d|n) 。。 （。。显然满足交换律。。）

如果 f、g 均为积性函数。。f\*g、f×g 也均为积性函数。。后者成为狄利克雷卷积。。在数论变换中很常见。。

之前两个变换的狄利克雷积表示。。id = φ × 1 。。e = μ × 1 。。再举个例子。。ΣGcd(n,k) (k<=n) = φ × id。。

id = φ × 1，φ = id × μ。。1 = e × 1，e = 1 × μ。。

莫比乌斯反演。。对于任意积性函数 f、g 有。。f = g × 1 <==> g = 1 × μ。。理论上来说用不到。。太复杂了。。

一些常见的数论变换！

1D。。(k<=n)

Σ e(Gcd(n,k)) 。。φ。。；

Σ Gcd(n,k) 。。 φ × id。。；O(N) - O(sqrtN)。。

Σ e(Gcd(n,k))\*k 。。可容斥。。也可数论变换。。id\*(φ+e)/2。。O(N) - O(1)。。

2D。。(i<=n，j<=m)

ΣΣ e(Gcd(i,j)) 。。 Σ μ(d)\*(n/d)\*(m/d) (d<=n)。。；O(N) - O(sqrtN)。。

ΣΣ Gcd(i,j)。。 Σ φ(d)\*(n/d)\*(m/d) (d<=n)。。；O(N) - O(sqrtN)。。

1D Lcm Sum。。Σ Lcm(n,k) = n(((id\*φ)+1)×1)/2。。由 1D 中互质的和推出。。O(N) - O(sqrtN)。。

2D Lcm Sum。。ΣΣ Lcm(i,j)。。

O(n) 做法：

= Σ d\*S(n/d,m/d) (d<=n) 。。其中 S(p,q) 表示 (i<=p,j<=q) 中互质对数的乘积。。

这里第一次分块：

S(n,m) = ΣΣ i\*j\*e(gcd(i,j)) = ΣΣ i\*j\*Σμ(d) (d|i && d|j) = Σ μ(d)\*d\*d\*Sum(n/d,m/d)。。

这里第二次分块：

其中 Sum(n,m) 为 (1~n)到(1~m) 中对数的乘积，即 Σi \* Σj。。

**博弈论——**

SG函数 和 SG定理 是博弈中的核心。

因为一个组合游戏的一个状态要么必胜要么必败。必败态后继没有必胜态，必胜态后继必有一个必败态。根据这个可以更加转换成 SG函数。SG定理其实就是多个组合游戏的和，道理是一样的。

好多看似新颖的游戏都可以转化成一些经典的游戏。转化的时候只要每个状态（的值）每个转移都能对应上，即便看似再荒谬也是一个等价转化！如果实在无法找到转化的方法当然打表找规律也是比较常见的。

Anti - Nim。。谁走最后一步即是 Loser 。。

网上那道多校题目完全是错的。。Hdu 3590。。09年论文中很详细地有写：

对于任意一个Anti-SG游戏，如果我们规定当局面中所有的单一游戏的SG值为0时，游戏结束，则先手必胜当且仅当：（1）游戏的SG函数不为0且游戏中某个单一游戏的SG函数大于1；（2）游戏的SG函数为0且游戏中没有单一游戏的SG函数大于1。

这个前提条件是非常重要的。。

Every - SG：每次行动每个子游戏都要走一步。

考虑让每个必胜的子游戏尽可能长地玩下去，必败的尽可能短。然后就是简单的 DP 了。

dp(u) = 0, max(step(v))+1), min(step(v)) + 1  
结论：先手必胜当且仅当单一游戏中最大的 dp 值为奇数。关键在于 SG=0 和 SG!=0 的点的 dp 值的奇偶性是不同的。

树的删边游戏：SG(u) = (SG(v0)+1) ^ (SG(v1)+1) ^ ... 非常好证。

无向图删边游戏（环不共用边 && 与基础树只有一个公共点），奇环切断后链长同奇偶，SG值为1，用长为 1 的链替代。偶环切断后链长异奇偶。SG值为0，直接删掉，变成了树的删边游戏。

翻硬币游戏：绝大多数情况下都可以转化成一些经典的 Nim 游戏，在有条件规定最右必须从 Heads 翻到 Tails 这种条件时是可以归纳成每个 Heads 朝上棋子时单一存在时的 SG 值的异或和。。

不平等博弈

呵呵。。QAQ

**语法——**

**STL中的归并排序：**

merge 和 inplace\_merge 是 C++ 自带的归并排序，也就是可以把两个有序数组 O(n) 来合并

merge(a + l, a + mid + 1, a + mid + 1, a + r + 1, b, cmp)

意思是把 a.l ~ a.mid 和 a.mid + 1 ~ a.r 这两段有序数合并后放到 b 里，cmp 可加可不加，就是用来比较的函数。

这里要注意的是不能使用 vector ，必须是数组。（是 list 也可以，不过一般不用。）

而且新的那个数组 b 不能是之前的数组，必须是新的。当然函数中前后两个部分的数组可以不一样。

inplace\_merge 则是一个完完全全可以代替归并排序的归并步骤的函数。

inplace\_merge(a + l, a + mid + 1, a + r, cmp)

也就是把 a.l ~ a.mid 和 a.mid + 1 ~ a.r 这两段有序数合并后放到 a.l ~ a.r 里，cmp 可加可不加，也只是用来比较的函数。

有了 inplace\_merge 后，最近点对再也不会超过 10 行了！

**小根堆：**

priority\_queue <int, vector <int>, greater <int> > Q;

**开栈：**

int size = 256 << 20; // 256MB

char \*p = (char\*)malloc(size) + size;

\_\_asm\_\_("movl %0, %%esp\n" :: "r"(p));

#pragma comment(linker, "/STACK:102400000,102400000")

**JAVA：**

比较吊的用法就是，可以自定义 radix。。输出的时候也只要 .toString(radix) 也可以自动转换成该进制下的整数 （小数是不可以的。。因为可能出现无限不循环小数）。。 吊飞

想要调用 BigInteger 的 长度的话，一个可行的做法是先 .toString() ，然后 .length();

pow 是可以调用的，不过奇慢无比，要注意。

.shiftLeft(n) 和 .shiftRight(n) 就和正常的左右移位是一样的。

当然加减乘除操作和常见的二进制操作也都是可以使用的。

至于 BigDecimal，就是多了个除法，和保留小数位数。

.setScale(n, BigDecimal.ROUND\_HALF\_UP); 就是直接保留 n 位小数，四舍五入

.divide(BigDecimal, n, BigDecimal.ROUND\_HALF\_UP); 就是除法的时候保留 n 位小数。