ACM-ICPC 培训资料汇编

（8）

博弈分册

（版本号 **1.0.0**）

哈尔滨理工大学 ACM-ICPC 集训队

2012 年 12 月

序

2012 年 5 月，哈尔滨理工大学承办了 ACM-ICPC 黑龙江省第七届大学生程序设计竞赛。做为本次竞赛的主要组织者，我还是很在意本校学生是否能在此次竞赛中取得较好成绩，毕竟这也是学校的脸面。因此，当 2011 年 10 月确定学校承办本届竞赛后，我就给齐达拉图同学很大压力，希望他能认真训练参赛学生，严格要求受训队员。当然，齐达拉图同学半年多的工作还是很有成效，不仅带着黄李龙、姜喜鹏、程宪庆、卢俊达等队员开发了我校的 OJ 主站和竞赛现场版 OJ，还集体带出了几个比较像样的新队员，使得今年省赛我校取得了很好的成绩（当然，也承蒙哈工大和哈工程关照，没有派出全部大牛来参赛）。

在 2011 年 9 月之前，我对 ACM-ICPC 关心甚少。但是，我注意到我校队员学习、训练没有统一的资料，也没有按照竞赛所需知识体系全面系统培训新队员。2011-2012 年度的学生教练们做了一个较详细的培训计划，每周都会给 2011 级新队员上课，也会对老队员进行训练，辛辛苦苦忙活了一年——但是这些知识是根据他们个人所掌握情况来给新生讲解的，新生也是杂七杂八看些资料和做题。在培训的规范性上欠缺很多，当然这个责任不在学生教练。2011 年 9 月，我曾给老队员提出编写培训资料这个任务，一是老队员人数少，有的还要去百度等企业实习；二是老队员要开发、改造 OJ；三是培训新队员也很耗费精力，因此这项工作虽很重要，但却不是那时最迫切的事情，只好被搁置下来。

2012 年 8 月底，2012 级新生满怀梦想和憧憬来到学校，部分同学也被 ACM-ICPC 深深吸引。面对这个新群体的培训，如何提高效率和质量这个老问题又浮现出来。市面现在已经有了各种各样的 ACM-ICPC 培训教材，主要算法和解题思路都有了广泛深入的分析和讨论。同时，互联网博客、BBS 等中也隐藏着诸多大牛对某些算法的精彩论述和参赛感悟。我想，做一个资料汇编，采撷各家言论之精要，对新生学习应该会有较大帮助，至少一可以减少他们上网盲目搜索的时间，二可以给他们构造一个相对完整的知识体系。

感谢 ACM-ICPC 先辈们作出的杰出工作和贡献，使得我们这些后继者们可以站在巨人的肩膀上前行。

感谢校集训队各位队员的无私、真诚和抱负的崇高使命感、责任感，能够任劳任怨、以苦为乐的做好这件我校的开创性工作。

唐远新

2012 年 10 月

编写说明

本资料为哈尔滨理工大学 ACM-ICPC 集训队自编自用的内部资料，不作为商业销售目的，也不用于商业培训，因此请各参与学习的同学不要外传。

本分册内容由周云矾编写和校核。

本分册内容大部分采编自各 OJ、互联网和部分书籍。在此，对所有引用文献和试题的原作者表示诚挚的谢意！

由于时间仓促，本资料难免存在表述不当和错误之处，格式也不是很规范，请各位同学对发现的错误或不当之处向acm@hrbust.edu.cn邮箱反馈，以便尽快完善本文档。在此对各位同学的积极参与表示感谢！

哈尔滨理工大学在线评测系统（Hrbust-OJ）网址：http://acm.hrbust.edu.cn，欢迎各位同学积极参与AC。

国内部分知名 OJ：

杭州电子科技大学：http://acm.hdu.edu.cn

北京大学：http://poj.org

浙江大学：http://acm.zju.edu.cn

以下百度空间列出了比较全的国内外知名 OJ:
http://hi.baidu.com/leo_xxx/item/6719a5ffe25755713c198b50

哈尔滨理工大学 ACM-ICPC 集训队

2012 年 12 月

目　录

第9章 博弈

9.1 取石子游戏

参考文献：**http://blog.csdn.net/acm_cxlove/article/details/7854530**
http://blog.163.com/zhong_yk/blog/static/72898053200822893315889/ （尼姆博弈）

扩展阅读：
编写：周云矾　　校核：周云矾

9.1.1 基本原理

1. 巴什博奕（Bash Game）

只有一堆 n 个物品，两个人轮流从这堆物品中取物，规定每次至少取一个，最多取 m 个。最后取光者得胜。

显然，如果 n=m+1，那么由于一次最多只能取 m 个，所以，无论先取者拿走多少个，后取者都能够一次拿走剩余的物品，后者取胜。因此我们发现了如何取

胜的法则：如果 n=（m+1）r+s，（r 为任意自然数，s≤m),那么先取者要拿走 s 个物品，如果后取者拿走 k（≤m)个，那么先取者再拿走 m+1-k 个，结果剩下

（m+1）（r-1）个，以后保持这样的取法，那么先取者肯定获胜。总之，要保持给对手留下（m+1）的倍数，就能最后获胜

2. Fibonacci Nim （斐波那契博弈）

有一堆个数为 n 的石子，游戏双方轮流取石子，满足：

1)先手不能在第一次把所有的石子取完；

2)之后每次可以取的石子数介于 1 到对手刚取的石子数的 2 倍之间（包含 1 和对手刚取的石子数的 2 倍)。

约定取走最后一个石子的人为赢家，求必败态。

这个和之前的 Wythoff’s Game 和取石子游戏 有一个很大的不同点，就是游戏规则的动态化。之前的规则中，每次可以取的石子的策略集合是基本固定

的，但是这次有规则 2：一方每次可以取的石子数依赖于对手刚才取的石子数。

这个游戏叫做 Fibonacci Nim，肯定和 Fibonacci 数列：f[n]：1,2,3,5,8,13,21,34,55,89,… 有密切的关系。如果试验一番之后，可以猜测：先手胜当且

仅当 n 不是 Fibonacci 数。换句话说，必败态构成 Fibonacci 数列。

就像“Wythoff 博弈”需要“Beatty 定理”来帮忙一样，这里需要借助“Zeckendorf 定理”（齐肯多夫定理)：任何正整数可以表示为若干个不连续的

Fibonacci 数之和。

先看看 FIB 数列的必败证明：

1、当 i=2 时，先手只能取 1 颗，显然必败，结论成立。

2、假设当 i<=k 时，结论成立。

则当 i=k+1 时，f[i] = f[k]+f[k-1]。

则我们可以把这一堆石子看成两堆，简称 k 堆和 k-1 堆。

（一定可以看成两堆，因为假如先手第一次取的石子数大于或等于 f[k-1]，则后

手可以直接取完 f[k]，因为 f[k] < 2*f[k-1]）

对于 k-1 堆，由假设可知，不论先手怎样取，后手总能取到最后一颗。下面我们分析一下后手最后取的石子数 x 的情况。

如果先手第一次取的石子数 y>=f[k-1]/3，则这小堆所剩的石子数小于 2y，即后手可以直接取完，此时 x=f[k-1]-y，则 x<=2/3*f[k-1]。

我们来比较一下 2/3*f[k-1]与 1/2*f[k]的大小。即 4*f[k-1]与 3*f[k]的大小，由数学归纳法不难得出，后者大。

所以我们得到，x<1/2*f[k]。

即后手取完 k-1 堆后，先手不能一下取完 k 堆，所以游戏规则没有改变，则由假设可知，对于 k 堆，后手仍能取到最后一颗，所以后手必胜。

即 i=k+1 时，结论依然成立。

对于不是 FIB 数，首先进行分解。

分解的时候，要取尽量大的 Fibonacci 数。

比如分解 85：85 在 55 和 89 之间，于是可以写成 85=55+30，然后继续分解 30,30 在 21 和 34 之间，所以可以写成 30=21+9，

依此类推，最后分解成 85=55+21+8+1。

则我们可以把 n 写成 n = f[a1]+f[a2]+……+f[ap]。（a1>a2>……>ap）

我们令先手先取完 f[ap]，即最小的这一堆。由于各个 f 之间不连续，则 a(p-1) > ap + 1，则有 f[a(p-1)] > 2*f[ap]。即后手只能取 f[a(p-1)]这一堆，

且不能一次取完。

此时后手相当于面临这个子游戏（只有 f[a(p-1)]这一堆石子，且后手先取）的必败态，即先手一定可以取到这一堆的最后一颗石子。

同理可知，对于以后的每一堆，先手都可以取到这一堆的最后一颗石子，从而获得游戏的胜利。

3. K 倍博弈

两人取一堆 n 个石子 先手不能全部取完 之后每人取的个数不能超过另一个人上轮取的 K 倍。 对于给定的 n, k， 先手是否有必胜的策略。

这种博弈的思考过程非常有意义。

当 k=1 的时候 可知必败局面都是 2^i 将 n 分解成二进制，然后先手取掉最后一个 1. 然后对方必然无法去掉更高的 1，而对方取完我方至少还能拿掉最后一

个 1 导致对方永远取不完。

当 k=2 的时候，必败局面都是斐波那契数列。利用“先手去掉最后一个 1，则后手必不能去掉更高阶的 1 导致取不完”的思想，斐波那契数列有一个非常好

的性质就是：任意一个整数可以写成斐波那契数列中的不相邻的项的和，于是将 n 写成这种形式，先取走最后一个 1，对方能取的数是这个数*2，小于高 2

位的 1，所以取不完。

当 K 的时候， 想办法构造数列，将 n 写成数列中一些项的和，使得这些被取到的项的相邻两个倍数差距>k 那么每次去掉最后一个 1 还是符合上面的条件。

设这个数列已经被构造了 i 项，第 i 项为 a[i]，前 i 项可以完美对 1..b[i]编码使得每个编码的任意两项倍数>K 那么有

a[i+1] = b[i] + 1;这是显然的 因为 b[i] + 1 没法构造出来，只能新建一项表示

然后计算 b[i+1] 既然要使用 a[i+1] 那么下一项最多只能是某个 a[t] 使得 a[t] * K < a[i+1] 于是

b[i] = b[t] + a[i+1]

然后判断 n 是否在这个数列里面

如果在，那么先手必败。否则不停的减掉数列 a 中的项构造出 n 的分解，最后一位就是了。

做一些解释：首先是 a[i]=b[i-1]+1; b[i-1]是由 a[0]……a[i-1]组成的最大的数，那么 b[i-1]+1 不可能用 a[0]……a[i-1]组成。

然后是：if(a[j]*k<a[i])

b[i]=b[j]+a[i]; else

b[i]=a[i];

要求 b[j]，表示 a[0]……a[i]组成，那么显然是要用到 a[i]的，不然不就成了 b[i-1]，既然用了 a[i]，但是又要使相邻的倍数在 K 以上。则找到最大的 j，

使

a[j]*k<a[i]那么满足条件，便是 a[0]……a[j]能组成的最大的数，加上 a[i]，那么后者表示当前项不能和之前项组合，那么最大的数就只能是本身

4. 威佐夫博奕（Wythoff Game）

有两堆各若干个物品，两个人轮流从某一堆或同

时从两堆中取同样多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

这种情况下是颇为复杂的。我们用（ak，bk）（ak ≤ bk ,k=0，1，2，…,n)表示

两堆物品的数量并称其为局势，如果甲面对（0，0），那么甲已经输了，这种局势我们

称为奇异局势。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)。

可以看出,a0=b0=0,ak 是未在前面出现过的最小自然数,而 bk= ak + k，奇异局势有

如下三条性质：

1。任何自然数都包含在一个且仅有一个奇异局势中。

由于 ak 是未在前面出现过的最小自然数，所以有 ak > ak-1 ，而 bk= ak + k > ak

-1 + k-1 = bk-1 > ak-1 。所以性质 1。成立。

2。任意操作都可将奇异局势变为非奇异局势。

事实上，若只改变奇异局势（ak，bk）的某一个分量，那么另一个分量不可能在其

他奇异局势中，所以必然是非奇异局势。如果使（ak，bk）的两个分量同时减少，则由

于其差不变，且不可能是其他奇异局势的差，因此也是非奇异局势。

3。采用适当的方法，可以将非奇异局势变为奇异局势。

假设面对的局势是（a,b)，若 b = a，则同时从两堆中取走 a 个物体，就变为了

奇异局势（0，0)；如果 a = ak ，b > bk，那么，取走 b - bk 个物体，即变为奇异局

势；如果 a = ak ， b < bk ,则同时从两堆中拿走 ak - ab - ak 个物体,变为奇异局

势（ ab - ak , ab - ak+ b - ak)；如果 a > ak ，b= ak + k,则从第一堆中拿

走多余

的数量 a - ak 即可；如果 a < ak ，b= ak + k,分两种情况，第一种，a=aj （j < k）

,从第二堆里面拿走 b - bj 即可；第二种，a=bj （j < k）,从第二堆里面拿走 b - a

j 即可。

从如上性质可知，两个人如果都采用正确操作，那么面对非奇异局势，先拿者必胜

；反之，则后拿者取胜。

那么任给一个局势（a，b），怎样判断它是不是奇异局势呢？我们有如下公式：

ak =[k（1+√5）/2]，bk= ak + k （k=0，1，2，…,n 方括号表示取整函数）

奇妙的是其中出现了黄金分割数（1+√5）/2 = 1。618…,因此,由 ak，bk 组成的矩形近

似为黄金矩形，由于 2/（1+√5）=（√5-1）/2，可以先求出 j=[a（√5-1）/2]，若 a=[

j（1+√5）/2]，那么 a = aj，bj = aj + j，若不等于，那么 a = aj+1，bj+1 = aj+1

+ j + 1，若都不是，那么就不是奇异局势。然后再按照上述法则进行，一定会遇到奇异

局势。

5. ＳＧ函数的求解(SG 博弈)

给定一个有向无环图和一个起始顶点上的一枚棋子，两名选手交替的将这枚棋子沿有向边进行移动，无法移 动者判负。事实上，这个游戏可以认为是

所有 Impartial Combinatorial Games 的抽象模型。也就是说，任何一个 ICG 都可以通过把每个局面看成一个顶点，对每个局面和它的子局面连一条有向边

来抽象成这个“有向图游戏”。下 面我们就在有向无环图的顶点上定义 Sprague-Garundy 函数。首先定义 mex(minimal excludant)运算，这是施加于一个

集合的运算，表示最小的不属于这个集合的非负整数。例如 mex{0,1,2,4}=3、mex{2,3,5}=0、mex{}=0。

对于一个给定的有向无环图，定义关于图的每个顶点的 Sprague-Grundy 函数 g 如下：g(x)=mex{ g(y) | y 是 x 的后继 }。

来看一下 SG 函数的性质。首先，所有的 terminal position 所对应的顶点，也就是没有出边的顶点，其 SG 值为 0，因为它的后继集合是空集。然后对于

一个 g(x)=0 的顶点 x，它的所有后继 y 都满足 g(y)!=0。对于一个 g(x)!=0 的顶点，必定存在一个后继 y 满足 g(y)=0。

以上这三句话表明，顶点 x 所代表的 postion 是 P-position 当且仅当 g(x)=0（跟 P-positioin/N-position 的 定义的那三句话是完全对应的）。我们通

过计算有向无环图的每个顶点的 SG 值，就可以对每种局面找到必胜策略了。但 SG 函数的用途远没有这样简单。如果将有 向图游戏变复杂一点，比如说，

有向图上并不是只有一枚棋子，而是有 n 枚棋子，每次可以任选一颗进行移动，这时，怎样找到必胜策略呢？

让我们再来考虑一下顶点的 SG 值的意义。当 g(x)=k 时，表明对于任意一个 0<=i<k，都存在 x 的一个后继 y 满足 g(y)=i。也 就是说，当某枚棋子的 SG 值

是 k 时，我们可以把它变成 0、变成 1、……、变成 k-1，但绝对不能保持 k 不变。不

知道你能不能根据这个联想到 Nim 游戏， Nim 游戏的规则就是：每次选择
一堆数量为 k 的石子，可以把它变成 0、变成 1、……、变成 k-1，但绝对不能保持 k 不变。这表明，如果将 n 枚棋子所在的顶 点的 SG 值看作 n 堆相应数量的石
子，那么这个 Nim 游戏的每个必胜策略都对应于原来这 n 枚棋子的必胜策略！
对于 n 个棋子，设它们对应的顶点的 SG 值分别为(a1,a2,…,an)，再设局面(a1,a2,…,an)时的 Nim 游戏的一种必胜策略是把 ai 变成 k，那么原游戏的一
种必胜策略就是把第 i 枚棋子移动到一个 SG 值为 k 的顶点。这听上去有点过于神奇——怎么绕了一圈又回到 Nim 游戏上了。
其实我们还是只要证明这种多棋子的有向图游戏的局面是 P-position 当且仅当所有棋子所在的位置的 SG 函数的异或为 0。这个证明与上节的 Bouton’s
Theorem 几乎是完全相同的，只需要适当的改几个名词就行了。
刚才，我为了使问题看上去更容易一些，认为 n 枚棋子是在一个有向图上移动。但如果不是在一个有向图上，而是每个棋子在一个有向图上，每次可
以任选一个棋子（也就是任选一个有向图）进行移动，这样也不会给结论带来任何变化。
所以我们可以定义有向图游戏的和(Sum of Graph Games)：设 G1、G2、……、Gn 是 n 个有向图游戏，定义游戏 G 是 G1、G2、……、Gn 的和(Sum)，游戏 G
的移动规则是：任选一个子游戏 Gi 并移动上面的棋子。Sprague-Grundy Theorem 就是：g(G)=g(G1)^g(G2)^…^g(Gn)。也就是说，游戏的和的 SG 函数值是
它的所有子游戏的 SG 函数值的异或。
再考虑在本文一开头的一句话：任何一个 ICG 都可以抽象成一个有向图游戏。所以“SG 函数”和“游戏的和”的概念就不是局限于有向图游戏。我们
给每 个 ICG 的每个 position 定义 SG 值，也可以定义 n 个 ICG 的和。所以说当我们面对由 n 个游戏组合成的一个游戏时，只需对于每个游戏找出求它的每个 局
面的 SG 值的方法，就可以把这些 SG 值全部看成 Nim 的石子堆，然后依照找 Nim 的必胜策略的方法来找这个游戏的必胜策略了！
回到本文开头的问题。有 n 堆石子，每次可以从第 1 堆石子里取 1 颗、2 颗或 3 颗，可以从第 2 堆石子里取奇数颗，可以从第 3 堆及以后石子里取任意颗…
… 我们可以把它看作 3 个子游戏，第 1 个子游戏只有一堆石子，每次可以取 1、2、3 颗，很容易看出 x 颗石子的局面的 SG 值是 x%4。第 2 个子游戏也是只有一
堆 石子，每次可以取奇数颗，经过简单的画图可以知道这个游戏有 x 颗石子时的 SG 值是 x%2。第 3 个游戏有 n-2 堆石子，就是一个 Nim 游戏。对于原游戏的每
个局面，把三个子游戏的 SG 值异或一下就得到了整个游戏的 SG 值，然后就可以根据这个 SG 值判断是否有必胜策略以及做出决策了。其实看作 3 个子游戏还
是保 守了些，干脆看作 n 个子游戏，其中第 1、2 个子游戏如上所述，第 3 个及以后的子游戏都是“1 堆石子，每次取几颗都可以”，称为“任取石子游戏”
，这个超简 单的游戏有 x 颗石子的 SG 值显然就是 x。其实，n 堆石子的 Nim 游戏本身不就是 n 个“任取石子游戏”的和吗？
所以，对于我们来说，SG 函数与“游戏的和”的概念不是让我们去组合、制造稀奇古怪的游戏，而是把遇到的看上去有些复杂的游戏试图分成若干个子游
戏，对于每个比原游戏简化很多的子游戏找出它的 SG 函数，然后全部异或起来就得到了原游戏的 SG 函数，就可以解决原游戏了。

9.1.2 解题思路

根据题目的意思，看它属于哪种博弈，属于哪种博弈的变形。 然后根据对应的博弈模型的解题策略来求解， 有时候并不一定能够直接看出它属于哪种模型，那这

个时候就可以通过判断自己每步可选的策略，对于自己每步走的，对当前局势的影响。然后推断出与之相对应的博弈模型。

9.1.3 模板代码

1. 巴什博奕（Bash Game）

```
#include <iostream>
using namespace std;
int main() {
    int iCase;
    cin >> iCase;
    while (iCase--) {
        int n, m;
        cin >> n >> m;
        if (n % (m + 1) == 0)    cout << "lost" << endl;
        else cout << "win" << endl;
    }
    return 0;
}
```

2. Fibonacci Nim （斐波那契博弈）

```
#include <iostream>
using namespace std;
long long int f[50];
int main() {
    int n, i;
    f[0] = f[1] = 1;
    for (i = 2; i < 50; i++) {
        f[i] = f[i - 1] + f[i - 2];
    }
    while (cin >> n) {
        if (n == 0) break;
        for (i = 0; i < 50; i++) {
            if (f[i] == n) break;
        }
        if (i < 50) cout << "Second win" << endl;
        else cout << "First win" << endl;
    }
    return 0;
}
```

3. K 倍博弈

```
#include <stdio.h>
#include <string.h>
#include <iostream>
using namespace std;
const int maxn=2000000;
int a[maxn],b[maxn];
int main()
{
        int n,k;
```

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

4. 威佐夫博弈（Wythoff Game）

```
#include <stdio.h>
#include <math.h>
const double ep1 = (sqrt(5.0) - 1.0) / 2.0;
const double ep2 = (sqrt(5.0) + 1.0) / 2.0;
int main () {
    int m, n;
    while (scanf ("%d %d", &m, &n) != EOF) {
        if (m > n) {
            int tmp;
            tmp = m, m = n, n = tmp;
        }
        int id = m * ep1;
        int tmp1 = ep2 * id, tmp2 = id + tmp1;
        int tmp3 = ep2 * (id + 1), tmp4 = id + 1 + tmp3;
        if (tmp1 == m && tmp2 == n) printf ("0\n");
        else if (tmp3 == m && tmp4 == n) printf ("0\n");
        else printf ("1\n");
    }
    return 0;
}
```

SG函数的求解(SG博弈）

HDU 3032 Nim or not Nim?

Lasker's Nim 游戏：每一轮允许两会中操作之一：①、从一堆石子中取走任意多个，②、将一堆数量不少于 2 的石子分成都不为空的两堆。

很明显：sg(0) = 0，sg(1) = 1。

状态 2 的后继有：0，1 和（1，1），他们的 SG 值分别为 0，1，0，所以 sg(2)

=2。

状态 3 的后继有：0、1、2、(1，2)，他们的 SG 值分别为 0、1、2、3，所以 sg(3) = 4。

状态 4 的后继有：0、1、2、3、(1，3) 和 (2，2)，他们的 SG 值分别为 0，1，2，4，5，0，所以 sg(4) = 3.

由数学归纳法可以得出

sg(4k)=4k-1;sg(4k+1)=4k+1;sg(4k+2)=4k+2;sg(4k+3)=4k+4;

```
#include <iostream>
using namespace std;
int main() {
    int iCase;
    cin >> iCase;
    while (iCase--) {
        int n, cnt = 0, num;
        cin >> n;
        while (n--) {
            cin >> num;
            if (num % 4 == 0) num--;
            else if (num % 4 == 3) num++;
            cnt ^= num;
        }
        if (cnt == 0) cout << "Bob" << endl;
        else cout << "Alice" << endl;
    }
    return 0;
}
```

9.1.4 经典题目

9.1.4.1 题目 1

1．题目出处/来源

Hdu 1846 http://acm.hdu.edu.cn/showproblem.php?pid=1846

2．题目描述

有一堆石子一共有 n 个，两人轮流进行，每走一步可以取走 1…m 个石子，最先取光石子的一方为胜。

3．分析

这是一道典型的巴什博奕题，可以根据巴什博奕的思维方式很容易的求出解来。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include <iostream>
using namespace std;
int main() {
    int iCase;
    cin >> iCase;
    while (iCase--) {
        int n, m;
```

```
        cin >> n >> m;
        if (n % (m + 1) == 0)    cout << "second" << endl;
        else cout << "first" << endl;
    }
    return 0;
}
```

5．思考与扩展：可以借鉴北大培训教材中做法。

9.1.4.2 题目 2

1．题目出处/来源

杭电OJ2516 http://acm.hdu.edu.cn/showproblem.php?pid=2516

2．题目描述

1 堆石子有 n 个, 两人轮流取. 先取者第 1 次可以取任意多个，但不能全部取完. 以后每次取的石子数不能超过上次取子数的 2 倍。取完者胜. 先取者负输出"Second win". 先取者胜输出"First win". 。

3．分析

可以直接参照斐波那契博弈的分析。 根据分析的结果可以很容易写出代码。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include <iostream>
using namespace std;
long long int f[50];
int main() {
    int n, i;
    f[0] = f[1] = 1;
    for (i = 2; i < 50; i++) {
        f[i] = f[i - 1] + f[i - 2];
    }
    while (cin >> n) {
        if (n == 0) break;
        for (i = 0; i < 50; i++) {
            if (f[i] == n) break;
        }
        if (i < 50) cout << "Second win" << endl;
        else cout << "First win" << endl;
    }
    return 0;
}
```

9.1.5 扩展变型

尼姆博弈(**The Game of Nim**)

参考文献：

扩展阅读：

如 3xian退役贴：http://davidzai.blog.163.com/blog/static/187126212010182354095 2/

编写：周云矾　　校核：周云矾

9.1.6 基本原理

有多堆物品两个人轮流从某一堆取任意多的物品，规定每次至少取一个，多者不限，最后取光者得胜。

我们关心的是，对于一个初始局面，究竟是先行者（甲）有必胜策略，还是后行者（乙）有必胜策略。

这种情况最有意思，它与二进制有密切关系，我们用（a，b，c）表示某种局势，首

先（0，0，0）显然是奇异局势，无论谁面对奇异局势，都必然失败。第二种奇异局势是

（0，n，n），只要与对手拿走一样多的物品，最后都将导致（0，0，0）。仔细分析一

下，（1，2，3）也是奇异局势，无论对手如何拿，接下来都可以变为（0，n，n）的情

形。

计算机算法里面有一种叫做按位模 2 加，也叫做异或的运算，我们用符号（+）表示

这种运算。这种运算和一般加法不同的一点是 1+1=0。先看（1，2，3）的按位模 2 加的结

果：

1 =二进制 01

2 =二进制 10

3 =二进制 11 （+）

————————

0 =二进制 00 （注意不进位）

对于奇异局势（0，n，n）也一样，结果也是 0。

任何奇异局势（a，b，c）都有 a（+）b（+）c =0。

如果我们面对的是一个非奇异局势（a，b，c），要如何变为奇异局势呢？假设 a < b < c，我们只要将 c 变为 a（+）b，即可，因为有如下的运算结果：a（+）b（+）（a（+）b）=（a（+）a）（+）（b（+）b）=0（+）0=0。要将 c 变为 a（+）b，只要从 c 中减去 c-（a（+）b）即可。

9.1.7 解题思路

一般如果能够确定那道题可以用尼姆博弈的原型去解的话，根据题目中的数据。将它们转化为尼姆博弈中所对应的数据，然后直接用尼姆博弈提供的方法去逐个的抑或。再根据得到的结果去判断那种是必胜态，那种事必败态。

9.1.8 模板代码

```
#include <iostream>
using namespace std;
int main() {
    int n;
    while (cin >> n) {
        if (n == 0) break;
        int cnt = 0, m;
        for (int i = 0; i < n; i++) {
            cin >> m;
```

```
            cnt ^= m;
        }
        if (cnt == 0) cout << "lost" << endl;
        else cout << "win" << endl;
    }
    return 0;
}
```

9.1.9 经典题目

9.1.9.1 题目 1

1．题目出处/来源

杭电OJ1849 http://acm.hdu.edu.cn/showproblem.php?pid=1849

2．题目描述

1 棋盘包含 1*n 个方格，方格从左到右分别编号为 0，1，2，…，n-1；

m 个棋子放在棋盘的方格上，方格可以为空，也可以放多于一个的棋子双方轮流走棋；每一步可以选择任意一个棋子向左移动到任意的位置（可以多个棋子位于同一个方格），当然，任何棋子不能超出棋盘边界；如果所有的棋子都位于最左边（即编号为 0 的位置），则游戏结束，并且规定最后走棋的一方为胜者。

3．分析

每个棋子初始的位置与最左边的方格的距离可以看着是一堆石子，m 个棋子到棋盘最左边的方格可以看着是 m 堆石子，每堆石子的数目分别为它初始时离最左边方格的距离。每次移动任意一个棋子的步数相当于从所对应的堆里取走相应的石子数。 因此，上面的题完全可以转化为尼姆博弈的模型。 代码挺好写的，下面是对应的 AC 代码：

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include <iostream>
using namespace std;
int main() {
    int n;
    while (cin >> n) {
        if (n == 0) break;
        int cnt = 0, m;
        for (int i = 0; i < n; i++) {
            cin >> m;
            cnt ^= m;
        }
        if (cnt == 0) cout << "Grass Win!" << endl;
        else cout << "Rabbit Win!" << endl;
    }
    return 0;
}
```

5．思考与扩展：可以借鉴北大培训教材中做法。

9.1.9.2 题目 2

1．题目出处/来源

HDOJ1730　http://acm.hdu.edu.cn/showproblem.php?pid=1730

2．题目描述

游戏在一个 n 行 m 列（1 ≤ n ≤ 1000 且 2 ≤ m ≤ 100）的棋盘上进行，每行有一

个黑子（黑方）和一个白子（白方）。执黑的一方先行，每次玩家可以移动己方的任何一枚棋子到同一行的任何一个空格上，当然这过程中不许越过该行的敌方棋子。双方轮流移动，直到某一方无法行动为止，移动最后一步的玩家获胜。

9.1.10 扩展变型

3．分析

转换成之间距离的 NIM 博弈。 通过转换它们之间的距离，就可以得到尼姆博弈的原型。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include <iostream>
#include <cstring>
#include <cstdio>
#include <cstdlib>
using namespace std;
int abst(int a,int b)
{
    int t=a-b;
    if(t>0)
        return t;
    else
        return 0-t;
}
int main()
{
    int ans;
    int n,m;
    int a,b;
    while(scanf("%d %d",&n,&m)!=EOF)
    {
        ans=0;
        for(int i=0;i<n;i++)
        {
            scanf("%d %d",&a,&b);
            ans^=(abst(a,b)-1);
        }
        if(ans==0)
        {
            printf("BAD LUCK!\n");
        }
        else
            printf("I WIN!\n");
    }
    return 0;
}
```

9.2 寻找必败态

参 考 文 献 ：**http://read.pudn.com/downloads167/sourcecode/book/769974/%D1%B0%D5%D2%B1%D8%B0%DC%CC%AC%A1%AA%A1%AA%B2%A9%DE%C4%CE%CA%CC%E2%B5%C4%BF%EC%CB%D9%BD%E2%B7%A8.pdf**

扩展阅读：

编写：**XXX** 校核：**xxx**

9.2.1 基本原理

必败态就是“在对方使用最优
策略时，无论做出什么决策都会导致失败的局面”。其他的局面称为胜态，值得注意的是在
胜态下做出错误的决策也有可能导致失败。此类博弈问题的精髓就是让对手永远面对必败
态。
必败态和胜态有着如下性质：
1、若面临末状态者为获胜则末状态为胜态否则末状态为必败态。
2、一个局面是胜态的充要条件是该局面进行某种决策后会成为必败态。
3、一个局面是必败态的充要条件是该局面无论进行何种决策均会成为胜态。
这三条性质正是博弈树的原理，但博弈树是通过计算每一个局面是胜态还是必败态来解
题，这样在局面数很多的情况下是很难做到的，此时，我们可以利用人脑的推演归纳能力找
到必败态的共性，就可以比较好的解决此类问题了。

9.2.2 解题思路

分析初始局势是属于哪种形态，然后根据博弈中的些结论去推导当前状态是否是必败态。

9.2.3 模板代码

9.2.4 经典题目

9.2.4.1 题目 1

1．题目出处/来源
POJ1740 A New Stone Game

2．题目描述
有 N 堆石子，两人轮流进行操作，每一次为“操作者指定一堆石子，先从中扔掉一部分（至少一颗，可以全部扔掉），然后可以将该堆剩下的石子中的任意多颗任意移到
其他未取完的堆中”，操作者无法完成操作时为负。
3．分析
只有一堆时先手必胜。
有两堆时若两堆相等则后手只用和先手一样决策即可保证胜利，后手必胜。若不同则先
手可以使其变成相等的两堆，先手必胜。
有三堆时先手只用一次决策即可将其变成两堆相等的局面，先手必胜。
有四堆时由于三堆必胜，无论先手后手都想逼对方取完其中一堆，而只有在四堆都为一
－ 1 － 颗时才会有人取完其中一堆，联系前面的结论可以发现，只有当四堆可以分成两两相等的两
对时先手才会失败。

分析到这里，题目好像已经有了一些眉目了，凭借归纳猜想，我们猜测必败态的条件为

“堆数为偶数（不妨设为 2N），并且可以分为两两相等的 N 对”。

下面只需证明一下这个猜想。其实证明这样的猜想很简单，只用检验是否满足必败态的

三条性质即可。

首先，末状态为必败态，第一条性质符合。

其次，可以证明任何一个胜态都有策略变成必败态（分奇数堆和偶数堆两种情况讨论）。

最后，证明任何一个必败态都无法变成另一个必败态（比较简单）。

由于篇幅关系，这里就不具体证明了，如果有兴趣可以自己试试

接下来的程序就相当简单了，只用判断一下即可。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
using namespace std;

bool f[1005];

int main()
{
    int n;
    while (scanf("%d", &n), n)
    {
        memset(f, 0, sizeof(f));
        int ans = 0;
        for (int i = 0; i < n; i++)
        {
            int a;
            scanf("%d", &a);
            if (f[a])
                ans--;
            else
                ans++;
            f[a] = !f[a];
        }
        if (ans)
            printf("1\n");
        else
            printf("0\n");
    }
    return 0;
}
```

5．思考与扩展：可以借鉴北大培训教材中做法。

9.2.4.2 题目 2

1．题目出处/来源

POJ1704 Georgia and Bob

2．题目描述

一个 1*M 的棋盘上有 N 个棋子，初始位置一定，两人轮流操作，每次移动一枚

棋子，要求只能向左移且至少移动一格，而且不能到达或经过以前有棋子的格子，谁无法移

动棋子就算输。

3．分析

乍一看这一题棋子移动还要受其他棋子的限制，好像无法求出通解，但仔细分析会发现

别有洞天。

一个棋子每一次向左移的最大步数是固定的，而且随着移动减少，不是和取石子很像么？那么和取石子的区别在哪呢？就在于每一次移动时都会让右边相邻的那颗棋子移动空

间变大，这样就和取石子只减不增有所不同了，我们应该怎样解决这个问题呢？

我们并不放弃将其与我们熟悉的取石子对应，但我们将策略做小小的变动：

将棋子从右端向左端每相邻两个分为一对，如果只剩一个就将棋盘左端加一格放一颗棋

子与之配对，这样配对后好像和以前没有什么区别，但决策时就方便多了，因为我们大可不

必关心组与组之间的距离，当对手移动一组中靠左边的棋子时，我们只需将靠右的那一颗移

动相同步数即可！同时我们把每一组两颗棋子的距离视作一堆石子，在对手移动两颗棋子中

靠右的那一颗时，我们就和他玩取石子游戏，这样就把本题与取石子对应上了。

本例说明有许多模型看似复杂，但经过一些巧妙的变换，便可以转化成一些我们熟悉的

模型，同时也充分体现了博弈的灵活。

如果说前面的例子是介绍这种思想的运用的话，下面的方法就是讲这种思路的优越性了，因为这一题并不是走的“手推小数据=〉猜想=〉证明”的老路，而是直接利用性质推导

必败条件。

4．代码（包含必要注释，采用最适宜阅读的 Courier New 字体，小五号，间距为固定值 12 磅）

```
#include<algorithm>
#include"stdio.h"
using namespace std;
 int main()
{
       int t,n,a[10002],i,c,k,b[5001],j;
       freopen("1704.txt","r",stdin);
       scanf("%d",&t);
       for(i=0;i<t;i++)
       {
              a[0]=0;
              scanf("%d",&n);
              for(j=1;j<=n;j++)
                     scanf("%d",&a[j]);
              sort(a,a+n+1);
```

```
        if(n%2==0)
              {
                     for(j=1,k=0;j<n;j+=2,k++)
                     b[k]=a[j+1]-a[j]-1;
              }
              else for(j=n,k=0;j>0;j-=2,k++)
                     b[k]=a[j]-a[j-1]-1;
              c=0;
           for(k=0;k<(n+1)/2;k++)
                     c=c^b[k];
              if(c==0)printf("Bob will win\n");
              else printf(" will win\n");
       }
       return 0;
}
```