美，无处不在

——浅谈“黄金分割”和信息学的联系

安徽芜湖一中 杨思雨

摘要

本文主要介绍了“黄金分割”与信息学竞赛的联系及其在一些信息学问题中的应用，全文可以分为四个部分。

第一部分引言，主要介绍了“黄金分割”的由来及其和各领域的联系，进而提出探索黄金分割与信息学竞赛的联系。

第二部分中逐渐深入的分析了一个例题，最终揭示了问题本质上与黄金分割数的联系，说明了黄金分割数在一些数学性较强的题目中有着广泛的应用。

第三部分通过分析另一个例题，介绍了一种优选法——“黄金分割法”。

第四部分指出黄金分割与信息学竞赛联系密切，总结全文。

关键字

黄金分割 信息学 联系

目录

引言

早在古希腊时期，人们就发现了“黄金分割”。两千多年前，数学家欧多克斯提出了这样的问题：如图 1，线段 AB 上有一点 P，将线段 AB 分割为两段——AP 和 PB，若它们长度之比恰好等于整条线段 AB 与较长一段 AP 的长度之比，那么 P 点应该在线段 AB 什么位置呢？

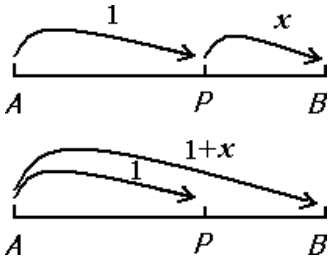


图 1

我们不妨假设线段 AP 的长度为 1，设线段 PB 的长度为 x，那么线段 AB 的总长度就是 1+x。由 $\frac{AP}{PB}=\frac{AB}{AP}$，得到方程 $\frac{1}{x}=\frac{1+x}{1}$。解出 $x=\frac{\sqrt{5}-1}{2}$，记为 ϕ，它的近似值为 0.618。而相应的，线段 AB 的长度就是 $1+x=\frac{\sqrt{5}+1}{2}$，记为 Φ。

在这之后，人们围绕这个比值开展了许多研究，意大利著名的艺术家、科学家达·芬奇还把 ϕ 命名为“黄金分割数（Golden Section Number）”。有趣的是，人们发现黄金分割在自然界和其它各个领域中到处可见。例如，人的肚脐是人体总长的黄金分割点，人的膝盖又是肚脐到脚跟的黄金分割点；在有些植物的茎上，相邻叶柄之间的夹角是 $137°28'$, 这恰好是把圆周分为 $1:0.618$ 的两条半径的夹角。

此外，黄金分割也被看作是美的化身，人们认为符合黄金分割比例的事物都非常协调，富有美感。这就使得它在建筑、绘画、雕塑、摄影和音乐等艺术领域也有着很广泛的应用，例如：在古希腊帕忒农神庙的设计中就存在着许多组黄金分割比；而在荷兰画家梵高的作品《岸边的渔船》中，整幅画的宽和高分别被桅杆和地平线分成两部分，这样的分割也正好符合了黄金分割比例。

其实，在信息学也蕴含着这样的奥妙，本文就将通过两个例题，介绍信息学和黄金分割的一些联系。

例题一

取石子游戏（SHTSC2002,Day2）

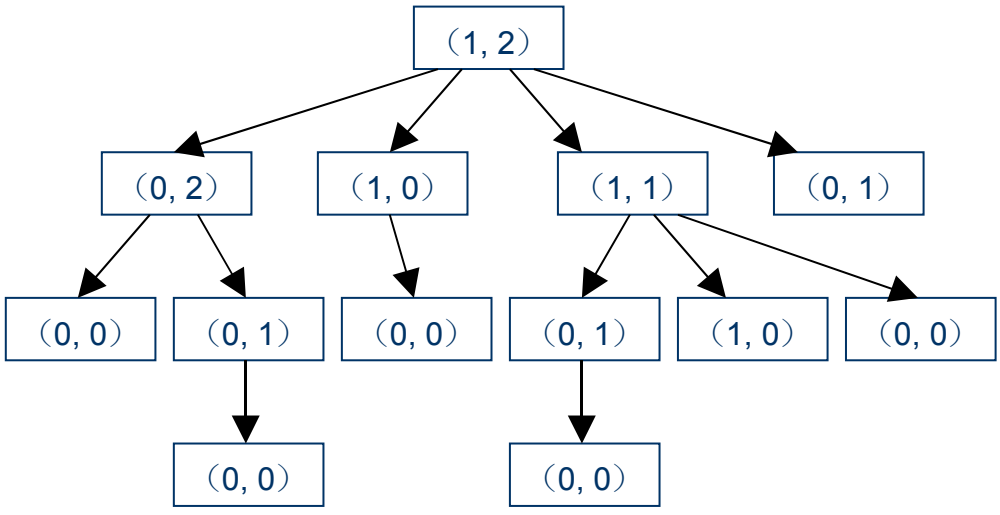
问题描述

有两堆石子，游戏开始后，由两个人轮流取石子，每次有两种取法：一是在任意一堆中取走任意数目的石子；二是可以在两堆中同时取走相同数量的石子。最后把石子全部取完的人是胜者。现在给出初始的两堆石子的数目 a 和 b（$0\le a,b\le 10^9$），假设双方都采取最好的策略，判断先手是否有必胜策略。

算法一

问题中，两堆石子的个数 a 和 b 决定了最后的胜负。因此，我们用(a,b)作为表示当前局面的“状态”。通过建立博弈树，判断给出的初始状态(a,b)是必胜还是必败。需要注意的是，由于规则中两个石子堆并没有差别，因此状态(a,b)和状态(b,a)实际上是等价的。

来看一个例子，游戏开始时，两堆石子的数目分别是 1 和 2，那么初始状态



为(1,2)，自顶向下构造出的博弈树如图 2（其中，重复子状态可以只扩展一个）。

图 2

接下来，就可以分三种情况，判断各个状态的胜负情况：
(1)如果一个状态没有子状态，根据规则判断胜负；
(2)如果一个状态至少有一个子状态先手败，那么该状态先手胜；
(3)如果一个状态的所有子状态都是先手胜，那么改状态先手败。

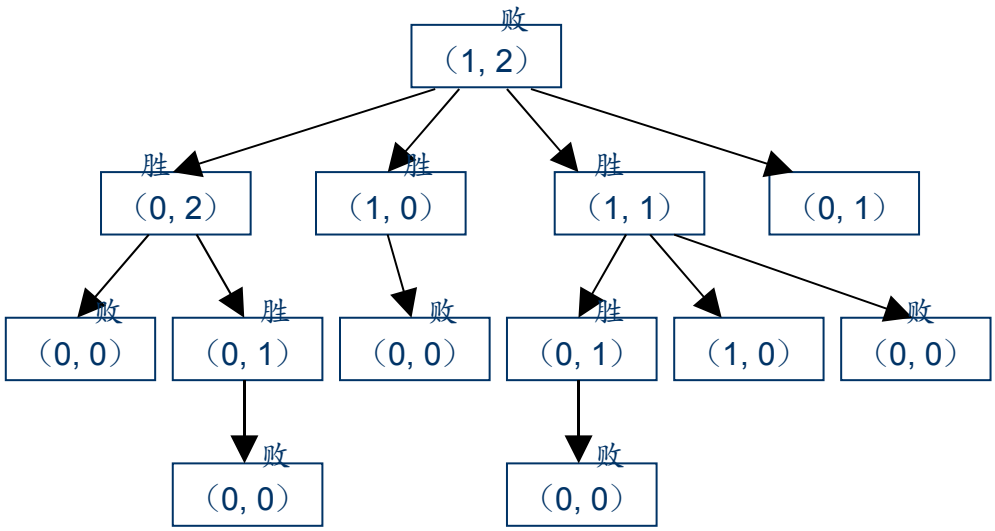


图 3

这样，我们就可以得出初始状态的胜负情况（如图3）。具体实现时，可以采用动态规划或记忆化搜索解决。这个算法的空最坏情况下查询的次数约为 2+[log1/ ϕ108]

间复杂度为 O(N^2)，时间复杂度为 O(N^3)。附程序 stone_1.pas。

算法二

显然上面的算法并不能有效的解决本题。我们需要进一步挖掘游戏规则的特点。根据两种取石子的方法，可以发现如果状态*(a,b)*是先手败，就能得到下面几种状态必为先手胜：

(1) 状态*(a,i)*（其中，$i \neq b$）或状态*(i,b)*（其中，$i \neq a$）；

(2) 状态*(a+i,b+i)*（其中，$i \neq 0$）。

也就是说，每个正整数在所有先手败状态中将出现且只出现一次，同时任何两个必败状态中两堆石子的差值各不相同。根据这个特点，我们可以构造出所有的先手败状态。首先，最小的先手败状态是(0,0)，根据上面的规律，之后的先手败状态中就不再存在某一堆为 0 个石子的情况，同时也不会出现两堆石子个数相等的情况。因此，第二种先手败的状态中较小的一堆石子个数将是尚未出现的最小的整数 1，而这种状态中两堆石子个数的差也是尚未出现的差值中最小

的一个。这样，第二种先手必败的状态就是(1,2)。

由此，我们得到了构造所有先手败状态的方法：开始时只有(0,0)一个先手败状态；第 *i* 次构造时，先找出在已知的先手败状态中没有出现的最小的整数 *a*，则新产生的先手败状态就是*(a,a+i)*。这个算法的时间复杂度和空复杂度均为 O(*N*)。附程序 stone_2.pas。

算法三

虽然算法二的时空复杂度相对算法一有了巨大的进步，但由于本题数据规模巨大，仍然没有办法彻底解决。而这时，我们的分析也陷入如了僵局。那么就让我们先从小规模的情况开始分析。

表 1 中列出了一些构造出来的必败状态（表中小数保留四位精度，下同）。可以发现，后面构造出来的状态中，较小一堆的石子数 *a* 与状态序号 *i* 的比值十

序号 i	***必败状态(a,b)***	***a / i***	***b / i***
1	(1,2)	1.0000	2.0000
2	(3,5)	1.5000	2.5000
3	(4,7)	1.3333	2.3330
4	(6,10)	1.5000	2.5000
5	(8,13)	1.6000	2.6000
…	…	…	…
997	(1613,2610)	**1.6179**	2.6179
998	(1614,2612)	**1.6172**	2.6172
999	(1616,2615)	**1.6176**	2.6176
1000	(1618,2618)	**1.6180**	2.6180

表 1

序号 i	***Φ·i***	***状态(a,b)***
1	**1**.6180	(**1**,2)
2	**3**.2361	(**3**,5)
3	**4**.8541	(**4**,7)
4	**6**.4721	(**6**,10)
5	**8**.0902	(**8**,13)
…	…	…
997	**1613**.1799	(**1613**,2610)
998	**1614**.7979	(**1614**,2612)
999	**1616**.4160	(**1616**,2615)
1000	**1618**.0340	(**1618**,2618)

表 2

分接近 Φ！这提示我们，也许 *a* 的值与 Φ·*i* 有一定的关系。所以，我们反过来观察 *Φ·i* 与 *a* 之间的关系。

在表 2 列出的小规模情况中，第 *i* 个必败状态中较小一堆的石子数 *a* 都恰好等于 Φ·*i* 的整数部分。于是我们提出了这样的猜想：

第 i 个必败状态是 **([Φ·*i*] , [(Φ+1)·*i*])**[1]。

[1] 本文中[*x*]表示 *x* 的整数部分，{*x*}表示 *x* 的小数部分。

可以证明，这个猜想是正确的[1]。因此，我们得出结论，对于状态*(a,b)*（其中，$a \le b$），如果 $\Phi \cdot \left(\left[\frac{\Phi}{a} \right] + 1 \right) = a$ 且 $a + \left[\frac{\Phi}{a} \right] + 1 = b$，则先手必败，否则先手必胜。

这样算法的时空复杂度均为常数级的，问题得到了圆满的解决。附程序 stone_3.pas。

解题小结

表 3 将上述三个算法进行了比较。算法一是解决博弈问题的常规方法，同时定义了状态；算法二，深入挖掘了游戏规则，找出了其中的特性；算法三则通过分析一些小规模的数据，找到了常数级的算法。通过这样不断的分析和挖掘，终于找到了问题本质上与黄金分割的联系，发现了这样一个博弈问题中蕴含的美。

	时间复杂度	空间复杂度	基本算法
算法一	O(N^3)	O(N^2)	根据博弈树进行动归
算法二	O(N)	O(N)	构造必败状态
算法三	O(1)	O(1)	直接判断胜负情况

表 3

例题二

登山问题（根据经典问题改编）

问题描述

单峰函数[2]*f(x)*在区间*[0,L]*上有极大值，要求通过一系列查询，找到这个极大值的横坐标 x_0。每次查询中，向交互库[3]提供一个查询点 t，交互库将返回*f(t)*。

数据范围：$0<L\le10^4$，要求查询次数不超过 40 次，且计算结果与最优点 x_0 误差不得超过 10^{-3}。

算法分析

对于区间内的两个不同的查询点，我们将函数值较大的称为好点，将函数值较小的称为差点。下面，我们将说明，***最优点和好点必在差点的同侧***。

证明 首先需要明确，最优点显然不会与差点重合。

如果最优点与好点重合，那么命题显然成立。

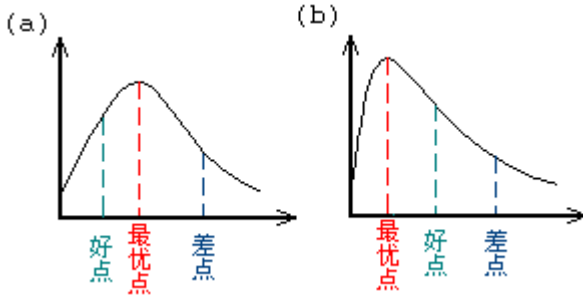


图 4

如果最优点并不在好点的位置，那么将存在下面两种情况：

(1) 如图 4(a)，好点与差点分别位于最优点的两侧，此时命题显然成立；

(2) 如图 4(b)，好点与差点位于最优点同侧；此时，由于 $f(x)$ 是单峰函数，离最优点较近的查询点必然是好点，因而最优点与好点仍在差点同侧。

证毕。

上面的结论，提示我们，可以通过比较目标区间内的两个查询点位置上的函数值，删去一部分目标区间。这样逐步缩小目标范围，不断逼近最优点，直到达到允许的误差范围内为止。

原题中，允许的误差范围是原区间长度的千万分之一，为了保证精度，我们需要将区间范围缩小为原来的 $1/10^8$。如何尽快的缩小目标范围，就成了我们需要研究的问题。而解决这个问题的关键，就是合理的选择查询点的位置。

下面我们就来分析一下具体应该如何选取查询点的位置。

设当前目标范围为 $[a,b]$，两个查询点为 x_1 与 x_2，$x_1<x_2$。

在查询之前，我们无法预先知道两个查询点孰好孰差，也就是说两个查询点成为差点的可能性是相同的。因此我们无法确定究竟会去掉哪一段区间范围。所以，我们在安排查询点时应该使它们关于当前目标范围的中点对称，即应使 $x_1-a = b-x_2$。这是我们在查询过程中应遵循的第一个原则——**对称原则**。

“对称原则”是十分常用的，例如我们经常在一些查找中使用的“二分法”，就完全符合这个原则。我们也就很容易想到使用二分法来解决本题：每次在当前目标范围中点附近，选择两个非常接近的查询点，并且根据这两个查询点的函数值，确定最优点的范围，缩小目标区间。这样每次可以舍去接近原区间一半长度的部分，为了保证精度，我们需要进行的查询次数就在 $2\log_2 10^8$ 左右，约为 54 次，达不到题目中的要求。

通过分析，我们可以发现造成查询次数较多的原因是，每次为了去掉区间

的一半，都需要重新进行两次查询；而实际上，原来的好点仍然处在当前的区间中，二分法却并没有利用到它的函数值信息。由此，我们想到，每次可以只在当前区间中进行一次查询，并与原来的好点进行比较，得出新的好点和差点，进而继续缩小目标范围。

由于我们每次选择的查询点要满足*对称原则*，因此，最开始选择的两个查询点的位置就决定了后面每次可以舍取得区间长度。我们发现，如果像二分法中一样，让 x_1 与 x_2 都尽量靠近，这样一次可以舍去整个因素范围*[a,b]*的将近50%，但是接下来每次只能舍去很小的一部分了，反而不利于较快地逼近最优点。这个情况又提示我们：最好每次舍去的区间都在全区间中占同样的比例（我们不妨称之为**"成比例的舍去"原则**）。

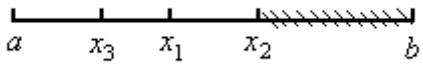


图 5

按照上述两个原则，如图 5 所示，设第一次和第二次查询分别在 x_1 点和 x_2 点，$x_1<x_2$，则在第一次比较结果后，由*对称性*，舍去的区间长度都等于 $b-x_2$。不妨设 x_1 是好点，x_2 是差点，舍去的是(x_2,b)。再设第三次试验安排在 x_3 点，则 x_3 点应在$[a,x_2]$中与 x_1 点对称的位置上。同时 x_3 点应在 x_1 点左侧，否则 x_3 点与 x_1 点比较查询结果后被舍去的将与上次舍去的是同样的长度，而不是同样的比例，违背*“成比例的舍去”原则*。由此可知，x_3 点与 x_1 点比较后，被舍去的区间长度都等于 x_2-x_1。于是按照*“成比例地舍去”原则*，我们得到等式：

$$\frac{b-x_2}{b-a}=\frac{x_2-x_1}{x_2-a}$$

经过变形、化简可得

$$\frac{x_2-a}{b-a}=\frac{x_1-a}{x_2-x_1}$$

这个等式和引言中关于线段分割的方程十分类似，这表明，x_2 正处于区间*[a,b]*的黄金分割点上。也就是说，查询点安排在区间的黄金分割点处较为妥当。因此，我们也将这种逼近最优值的方法称为“黄金分割法”。

效率分析

由于每次查询点都在当前目标区间的黄金分割点处，每次保留下来的区间和原来区间的长度比都是 ϕ:1。因此，最坏情况下查询的次数约为 $2+[\log_{1/\phi}10^8]$，恰好在 40 次之内，可以圆满的解决问题。附程序 explorer.pas。

解题小结

在这个问题中，我们摆脱了“二分法”思想的束缚，充分利用每次查询的信息，提出了两个选择查询点时需要遵循的原则。而由于黄金分割所特有的神奇的比例性质，恰好符合了*“成比例的舍去”原则*，为解决本题提供了出路。

总结

上文中两个例题的解决，都与“黄金分割数”有着密切的联系。

例一构造的数对中，蕴藏着黄金分割数的规律。其实信息学竞赛中，常常会出现一些数学性较强的题目。而黄金分割数，恰恰和构造数列、矩阵等密切相关，也就往往会成为解决这类题目的钥匙。

例二中的解决中，黄金分割所特有的比例性质使得它成为解决问题的关键。而在一些类似的查询问题中，黄金分割都有着广泛的应用，理论上可以证明，上面介绍的黄金分割法，查询的次数是各种优选法中最优的[2]。

当然，黄金分割与信息学竞赛的联系远不止这些，也还有很多奥妙需要我们去探索和发现。法国雕塑家罗丹曾说过：“美，是无处不在的，对于我们的眼睛，缺少的不是美，而是发现。”其实，只要勤于思考、勇于探索，每个人都可以拥有一双善于发现美的慧眼。

参考文献

[1] 姜璐主编，《中国少年儿童百科全书——科学·技术》，第 2 版，浙江教育出版社，1994.12.

[2] <美>Tobias Dantzig 著，苏仲湘译，《数：科学的语言》，第 1 版，上海教育出版社，2000.12

[2] 具体证明参见“参考文献[6]”。

[3]　《数学手册》编写组，《数学手册》，第 1 版，人民教育出版社，1979.05

[4]　张维勇主编，《青少年信息学奥林匹克竞赛试题与解析（安徽省 1994～2004 年）》，第 1 版，合肥工业大学出版社，2004.08

[5]　骆骥，《浅析解“对策问题”的两种思路——从〈取石子〉问题谈起》，2002 年国家集训队论文，2002.02

[6]　华罗庚，《优选学》，第 1 版，科学出版社，1981.04

附录

[1] 例题一，算法三，猜想的证明。

定义 1.1 根据算法二构造列数为 2 的矩阵 A，使得必败状态依次为 $(A_{1,1},A_{1,2})$，$(A_{2,1},A_{2,2})$，$(A_{3,1},A_{3,2})$,……。

定义 1.2 根据猜想构造列数为 2 的矩阵 B，其中，$B_{i,1}=[\Phi\cdot i\ ]$，$B_{i,2}=[(\Phi+1)\cdot i\ ]$。

引理 1.1(Betty 定理) 如果 a 和 b 是两个正无理数，且满足

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

且 $P=\{x\mid x=[at],t\in Z^+\}$，$Q=\{x\mid x=[bt],t\in Z^+\}$，则 P 和 Q 是正整数集 Z^+的一个划分，即

$$P\cup Q=Z^+,P\cap Q=\varnothing$$

证明 对于任意正整数 n，设 P 集合中小于 n 的元素的个数为 x，同样 Q 集合中小于 n 的个数为 y。

那么显然有

$$x<\frac{n}{a},y<\frac{n}{b}$$

由于 a, b 都是无理数，因此 $\frac{n}{a}, \frac{n}{b}$ 都不可能是整数，所以有

$$x = [\frac{n}{a}], y = [\frac{n}{b}]$$

那么考察 $x + y$

$$x + y = [\frac{n}{a}] + [\frac{n}{b}] < [\frac{n}{a}] + \{\frac{n}{a}\} + [\frac{n}{b}] + \{\frac{n}{b}\} = \frac{n}{a} + \frac{n}{b} = n$$

而

$$n - (x + y) = \{\frac{n}{a}\} + \{\frac{n}{b}\} < 2$$

综上，有

$$n - 2 < x + y < n \Rightarrow x + y = n - 1$$

那么可以知道，对于任意 n，在 P 和 Q 集合中小于 n 的元素共有(n - 1)个；而小于(n + 1)的元素共有 n 个。

所以，P, Q 两个集合中等于 n 的元素**有且仅有** 1 个！

因此 Z^+中的每一个元素不是在 P 集合中，就是在 Q 集合中，不可能既不在 P 中也不在 Q 中，也不可能既在 P 中又在 Q 中。所以说

$$P \cup Q = Z^+, P \cap Q = \varnothing$$

推论 1.1 如果 a 和 b 是两个正无理数，且满足：

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

那么任意正整数 x，可以且只可以表示为[at]和[bt]中的一种形式。（$t \in Z^+$）

推论 1.2 任意正整数 x 都在矩阵 B 中出现且只出现一次。

证明 由于有等式：

$$\frac{1}{\Phi} + \frac{1}{\Phi + 1} = 1$$

根据*推论 1.1* 和*定义 1.2*，*推论 1.2* 成立。

引理 1.2 对于任意的 i>1，都有 $B_{i,1}$ 是除矩阵 B 的前 i-1 行中数字外最小的正整数。

证明 根据*定义 1.2*，比 $B_{i,1}$ 小的整数都应出现在矩阵的前 *i*-1 行中。

在矩阵第一列中，所有数字都可以表示成为[Φ·*t*]的形式。其中，比[Φ·*i*]小的数字个数为$\left[\frac{[\Phi\cdot i]}{\Phi}\right]$。同理，矩阵 *B* 的第二列中，比[Φ·*i*]小的数共有$\left[\frac{[\Phi\cdot i]}{\Phi+1}\right]$个。

因此，在整个矩阵的前 *i*-1 行中出现的比[Φ·*i*]小的整数个数为

$$\left[\frac{[\Phi\cdot i]}{\Phi}\right]+\left[\frac{[\Phi\cdot i]}{\Phi+1}\right]<\frac{[\Phi\cdot i]}{\Phi}+\frac{[\Phi\cdot i]}{\Phi+1}=[\Phi\cdot i]$$

同时，有

$$\left[\frac{[\Phi\cdot i]}{\Phi}\right]+\left[\frac{[\Phi\cdot i]}{\Phi+1}\right]=\frac{[\Phi\cdot i]}{\Phi}+\frac{[\Phi\cdot i]}{\Phi+1}-\left\{\frac{[\Phi\cdot i]}{\Phi}\right\}-\left\{\frac{[\Phi\cdot i]}{\Phi+1}\right\}>[\Phi\cdot i]-2$$

因此，在整个矩阵的前 *i*-1 行中出现的比[Φ·*i*]小的整数个数为[Φ·*i*]-1，综合*推论 1.2* 可以得出，$B_{i,1}$ 是除矩阵 *B* 中前 *i*-1 行中的数以外最小的正整数。

定理 1 矩阵 *A* 和矩阵 *B* 等价，即对任意的 *i* 均有 $A_{i,1}=B_{i,1}$ 且 $A_{i,2}=B_{i,2}$。

对行数 *i* 进行归纳：当 *i*=1 时，显然有

$$A_{1,1}=B_{1,1}，A_{1,2}=B_{1,2}。$$

假设 *i*<*k* 是均有 $A_{i,1}=B_{i,1}$ 且 $A_{i,2}=B_{i,2}$。根据*定义 1.1*，$A_{k,1}$ 是除掉矩阵 *A* 中前 *k*-1 行中数字外最小的正整数。而根据*引理 1.2*，也有 $B_{k,1}$ 是除掉矩阵 *B* 中前 *k*-1 行中数字外最小的正整数。根据归纳假设，有

$$A_{k,1}=B_{k,1}$$

根据算法二的构造方法，有 $A_{k,2}=A_{k,1}+k$。而根据*定义 1.2* 有

$$B_{k,2}=[(\Phi+1)\cdot k]=[\Phi\cdot k]+k=B_{k,1}k$$

因此，可以得出

$$A_{k,2}=B_{k,2}$$

综上，定理成立，算法三中关于必败状态的猜想也成立。

[2] 例题二，单峰函数的定义

根据题目，这里只给出存在极大值的单峰函数*f(x)*定义：

设 x_0 是区间*[a,b]*上的函数最大点，则有

(1) 对于任意的 $x_1<x_2<x_0$，都有 $f(x_1)<f(x_2)<f(x_0)$；

(2) 对于任意的 $x_0<x_1<x_2$，都有 $f(x_0)>f(x_1)>f(x_2)$。

[3] 例题二，交互库

C++程序员使用的交互库：tools_c.h

Pascal 程序员使用的交互库：tools_p.ppu

特别感谢，安徽省芜湖市第一中学的周冬同学为本题编写交互库。

程序描述

[1] Stone_1.pas

```
Program Stone_Algorithm1;

Const
    inf='stone.in';                              //输入文件
    outf='stone.out';                            //输出文件
    maxn=100;

Var
    a,b:longint;                                 //石子个数
    ans:array[0..maxn,0..maxn]of longint;        //各个状态的胜负情况

function dfs(a,b:longint):longint;               //深度搜索
var i,t:longint;
begin
    if ans[a,b]<>0 then begin                    //状态(a,b)已经计算过
       ans[b,a]:=ans[a,b];
       dfs:=ans[a,b]; exit;
    end;
    if ans[b,a]<>0 then begin                    //状态(b,a)已经计算过
       ans[a,b]:=ans[b,a];
       dfs:=ans[a,b]; exit;
    end;

    for i:=1 to a do                             //在第一堆中取 i 个石子
      begin
        t:=dfs(a-i,b);
```

```
        if t=1 then begin                    //有一个子状态先手败
           ans[a,b]:=2; ans[b,a]:=2;         //先手胜
           dfs:=2;
           exit;
        end;
     end;
   for i:=1 to b do                          //在第二堆中取 i 个石子
     begin
        t:=dfs(a,b-i);
        if t=1 then begin                    //有一个子状态先手败
           ans[a,b]:=2; ans[b,a]:=2;         //先手胜
           dfs:=2;
           exit;
        end;
     end;
   for i:=1 to a do                          //在两堆石子中同时取走 i 个
     if i<=b then begin
        t:=dfs(a-i,b-i);
        if t=1 then begin                    //有一个子状态先手败
           ans[a,b]:=2; ans[b,a]:=2;         //先手胜
           dfs:=2;
           exit;
        end;
     end
     else break;
   dfs:=1;                                   //所有子状态都先手胜，该状态先手败
end;

Begin
   assign(input,inf); reset(input);
   assign(output,outf); rewrite(output);
     fillchar(ans,sizeof(ans),0);            //初始化
     ans[0,0]:=1;                            //状态(0,0)，先手败
     while not(seekeof) do                   //多组数据
       begin
          read(a,b);                         //读入 a,b
          ans[a,b]:=dfs(a,b);                //进行搜索
          writeln(ans[a,b]-1);               //输出结果
       end;
   close(input); close(output);
End.
```

[2] Stone_2.pas

```
Program Stone_Algorithm2;
```

```
Const
   inf='stone.in';                          //输入文件
   outf='stone.out';                        //输出文件
   maxn=1000000;

Var
   a,b:longint;                             //石子个数
   f:array[0..maxn]of boolean;              //各整数是否出现

procedure swap;                             //交换a和b
var t:longint;
begin
   t:=a; a:=b; b:=t;
end;

procedure work;                             //计算过程
var i,t:longint;
begin
   fillchar(f,sizeof(f),0);                 //初始化
   t:=0;                                    //构造出的比败状态的个数
   for i:=0 to a-1 do
     if not f[i] then begin                 //整数i尚未出现
        f[i+t]:=true;
        inc(t);
     end;
   if not(f[a]) and (a+t=b)                 //判断
      then writeln(0) else writeln(1);
end;

Begin
   assign(input,inf); reset(input);
   assign(output,outf); rewrite(output);
     while not(seekeof) do                  //多组数据
       begin
          read(a,b);                        //读入a和b
          if a>b then swap;                 // a≤b
          work;                             //计算过程
       end;
   close(input); close(output);
End.
```

[3] Stone_3.pas

```
Program Stone_Algorithm3;
```

```
Const
   inf='stone.in';                                  //输入文件
   outf='stone.out';                                //输出文件
   phi=(sqrt(5)+1)/2;                               //黄金分割数

Var
   a,b:longint;                                     //石子个数

procedure swap;                                     //交换a和b
var t:longint;
begin
   t:=a; a:=b; b:=t;
end;

procedure main;
var t:longint;
begin
   assign(input,inf); reset(input);
   assign(output,outf); rewrite(output);
     while not(seekeof) do                          //多组数据
       begin
          read(a,b);                                //读入a和b
          if a>b then swap;                         //a≤b
          if a+b=0 then begin                       //a=b=0，先手败
             writeln(0); continue;
          end;
          t:=trunc(a/phi+1);
          if (a=trunc(t*phi)) and (b=a+t)           //判断胜负情况
          then writeln(0) else writeln(1);
       end;
   close(input); close(output);
end;

Begin
   main;
End.
```

[4] Explorer.pas

```
Uses tools_p;

Const
   zero=1e-4;                                   //误差
   phi=(sqrt(5)-1)/2;                           //黄金分割数
```

```
Var
   L,h,t:extended;                          //区间长度L，当前目标区间[h,t]

procedure work;                             //计算过程
var lastf,lastx,x,f:extended;
begin
   lastx:=t*phi;                            //第一个查询点
   lastf:=ask(lastx);                       //查询函数值
   while abs(h-t)>zero do                   //直到在误差范围内为止
     begin
        x:=h+t-lastx;                       //根据对称原则，找到查询点
        f:=ask(x);                          //查询函数值
        if f>lastf then begin               //判别好点差点
           if x>lastx then h:=lastx         //缩小目标范围
           else t:=lastx;
           lastx:=x; lastf:=f;              //更改区间内好点
        end
        else begin
           if x>lastx then t:=x             //缩小目标范围
           else h:=x;
        end;
     end;
end;

Begin                                       //主过程
   L:=Start;                                //初始化
   h:=0; t:=L;                              //开始目标区间位[0,L]
   work;                                    //计算过程
   Answer((h+t)/2);                         //返回最终结果
End.
```