摘 要

组合游戏是指，信息完全轮流操作的双人游戏。本文将从一个游戏——k 倍动态减法游戏（k-based dynamic subtraction game）的解答出发，讨论游戏中的一个重要分支——组合游戏，并探讨解决这一类的问题的高效方法。探索的过程力求兼顾游戏论中的普遍真理以及问题的独特性质。

作为一篇信息学的论文，本文将着重讨论与组合游戏有关的相关算法和这些算法时间复杂性分析，而不是游戏论中的一些经典的理论、典型的游戏实例或者一些存在性的证明。

“k 倍动态减法游戏”在 k 为一般正实数的情况，在 2002 年的国家集训队论文中已经给出 $O(S^3)$的算法，本文将在第 4．1 节给出一个优化到 O(S)的算法。

“Nim 积”运算是利用 SG 函数解决游戏论问题的必要工具之一。本文将在第 4．4．5 节给出一个 $O(\log^2 x)$的计算“Nim 积”的算法。

2008 年的 BOI（Baltic Olympiad in Informatics）中 knight 一题就是游戏论问题，官方解答给出了 $O(n^3)$的算法，但没有给出算法时间复杂度的证明。本文在第 5 章提出了对这个 $O(n^3)$的上界的质疑，并举出了质疑的确实依据，举出反例证明了官方解答时间复杂度估计的错误之处。同时，本文经过对算法进一步优化，得到了一个 $O(n^2)$的算法。

上面这些内容都是原创的独立研究成果。

关 键 字

NP 状态、单调性、SG 函数、“Nim 和” 、“Nim 积” 、对称性分析、贪心分析、时间复杂度分析

一．引言

1．1．组合游戏的定义

“游戏”这个词包含的范围很广，像“石头剪刀布”“斗地主”“军旗”“象棋”“Nim 游戏”等等。但是，一个游戏，作为组合游戏，必须具有以下特征：

1：双人游戏。

2：双方轮流操作。即，组合游戏不包含双方同时进行的操作。

3：在游戏中的任意一个时刻，当前玩家可以选择的操作，是一个有限的确定的集合。即，组合游戏不包含随机化的操作。

4：信息完全。即，在游戏中的任意时刻，游戏中的任意一方都知道游戏的初始状态，以及从“游戏开始”到“当前”为止的双方的所有的历史操作，没有任何的保密。

5：游戏中的某一些状态被规定为“胜利终止状态”，到达这些状态的玩家获胜（即对方把胜利终止状态交给你的话，你就输了）；某一些状态被规定为“失败终止状态”，到达这些状态的玩家失败。本文只讨论，有限次操作后能分出胜负的那一类组合游戏，这一类游戏中将没有“平局”的情况。由于平局的情况是复杂的，本文不讨论。例如，“中国象棋”“围棋”都是经典的组合游戏，但不属于本文讨论范围。

所以，像“推箱子”这样的单人游戏，不属于组合游戏；像“石头剪刀布”这样包含同时操作的游戏，不属于组合游戏；像“飞行棋”“大富翁”这样包含掷色子的随机行为的游戏不属于组合游戏；像“斗地主”“军旗”这样的对自己当前状态保密的游戏，不属于组合游戏。

所以总结起来，对于一个组合游戏，一般可以用 f：X→P(x)来描述游戏规则。其中 X 表示状态集，P(x)表示 X 的子集簇。f(x)就表示玩家可以从 x 出发一步操作后到达的状态的集合。我们称任一个 $y \in f(x)$，y 是 x 的一个后继。也就是说，本文讨论的组合游戏可以被理解成一个有向无环图 G=(V,E)，图的点集 V=X，边集 E=(xy| $y \in f(x)$)。

1．2．组合游戏需要解决的问题

由于组合游戏是一个双方博弈的游戏，所以对于一个游戏的研究一般需要回

答两个问题：谁有必胜策略？必胜策略是什么？

而在信息竞赛学的问题中需要解决的问题，也与此相对应，分三个层次：

（1）判断谁有必胜策略；

（2）寻找单回合的必胜选择；

（3）维护多回合的必胜选择（交互题涉及）。

1．3．衡量解决组合游戏的算法的时间的尺度

作为信息学的问题，算法的时间效率至关重要。对于一般的信息学问题，算法的时间复杂度是依据程序运行时间相对于输入文件规模增长的增长来衡量的。这一衡量尺度对于游戏论问题当然是适用的，但是对于游戏论问题，特别是涉及到前面提出的第三类交互式问题的时候，本文也会采用下面的衡量方式。

首先，所有当前状态的直接或间接的后继状态的数量 S(x)、描述当前状态所必需的数据规模 D(x)、最坏情况下结束游戏的回合数 T(x)都可以衡量一个游戏的复杂程度，因此程序运行时间的增长与此三者之一的增长速度的关系，也可以衡量一个算法的时间复杂度的尺度。

其次，一方面由于对当前状态的描述必须能是的当前状态能区别于其它状态（特别的，包括他的所有直接的或间接的后继状态），所以 $D(x)=\Theta(\log S(x))$。另一方面，由于表达游戏的图是有向无环的，所以对任意 $y\in f(x)$ 都有 $S(x)\geq S(y)+1$，所以 $S(x)=\Omega(T(x))$。有很多例子中，$S(x)=\Theta(T(x))$，下文中将会提及的“捡石子游戏”就是一个这样的例子；但是，在更多的游戏中 $S(x)\neq\Theta(T(x))$，例如在本文主要讨论的“k 倍动态减法游戏”，$S(x)=\Theta(T(x)^2)$，在著名的 Nim 游戏中 $S(x)=\Theta(T(x)^k)$（其中 k 是 Nim 中数的组数），在经典的 Green Hackenpush 中 $S(x)=\Theta(2^{T(x)})$。

鉴于上面得出的 S(x)D(x)T(x)的相互数量关系，本文将选择 T(e)(e 表示初始状态)这个游戏结束所需的最少时间作为衡量算法时间复杂度的尺度，正如一般以输入时间这一程序运行的最少时间作为程序的运行标准一样。

而且，在一些问题中，游戏状态的描述的规模 D(x)（这很有可能就是输入文件的规模，因为输入文件应当包含且只包含游戏规则描述以及初始状态描述，例如下面的“k 倍动态减法游戏”中，整数 k 是游戏规则描述，整数 S 是初始条件描述）可能很小，但 T(e)很大时，程序运行时间与输入规模的相对关系就意义不

大了。

对于第(3)类交互式问题而言，O(T(e))-O(1)的在线算法是最理想的。

1．4．游戏论问题对于信息学竞赛的普遍意义

游戏论题目的考察兼顾了动态规划的基本算法原理，和各种分析手段的灵活应用，是一类没有固定公式的综合型问题。这也正是游戏论越来越受欢迎的原因。近些年，尤其最近一年，关于游戏论的问题在国际各大竞赛，各题库网站上屡见不鲜。因此，游戏可以说已经成为信息学竞赛新热点。

二．问题的提出

k 倍动态减法游戏：有一个整数 S（>=2），先行者在 S 上减掉一个数 x，至少是 1，但小于 S。之后双方轮流把 S 减掉一个正整数，但都不能超过先前一回合对方减掉的数的 k 倍，减到 0 的一方获胜。问：谁有必胜策论。

根据引言部分游戏论的理论，把这个游戏抽象成数学模型，可以用一个有序数对(m,n)来表示状态，表示当前 S 剩下 m，当前回合操作者最多可以减去 n，于是初始状态为(S,S-1)。

对于 k=1 或 2 的情况，2002 年国家集训队的论文中已给出结论。当 k=1，当且仅当 S 是 2 的幂次时，后手胜。当 k=2，当且仅当 S 是一个费波纳切数时，后手胜。显然这两个判定的时间复杂度都是 O(logS)，如果考虑高精度计算的话时间复杂度是 $O(\log^2 S)$。若要更进一步，找到必胜策略，并且在整个游戏过程中维护它，这里先略去，后面会同一般的 k 的情况一起讨论。

三．基本理论——NP 状态定理；基本方法——动态规划

3．1．N 状态 P 状态的概念：

所谓 N 状态，是指当前即将操作的玩家有必胜策略（N 来源于 Next player wins.）。

所谓 P 状态，是指先前刚操作完的玩家有必胜策略（P 来源于 Previous player wins.）。

这里用一个简单的捡石子游戏（Take-Away Game）来说明这两个概念：

有两个游戏者：A 和 B。有 21 颗石子。两人轮流捡走石子，每次可取 1、2 或 3 颗。A 先取。取走最后一颗石子的人获胜，即没有石子可取的人算输。

如果操作完剩下 0 颗石子，那刚才操作的人就胜了，所以 0 是 P 状态。如果剩下 1、2 或 3 颗石子，那么接下来取的人就能获胜，这是 N 状态；如果剩下 4 颗，那么无论接下来的人怎么取，都会出现前面这种情况，所以接下来取的人一定会输，所以 4 是 P 状态；如果剩下 5、6 或 7 颗石子，那么接下来取的人只要使得剩下 4 颗石子，他就能获胜。0，4，8，12，……都是 P 状态。

可以发现规律，当且仅当剩余石子为 4 的倍数时是 P 状态。现在有 21 颗石子，所以是 N 在状态，所以 A 必胜。

3．2．NP 状态定理：

定理：P 状态是“胜利终止状态”或者它的一切后继都为 N 状态，N 状态是“失败终止状态”或拥有至少一个后继是 P 状态。

3．3．通式的动态规划解法

根据上面的定理，可以利用动态规划求得每个状态是 P 状态还是 N 状态。具体的实现方法，可以按照图的拓扑逆序倒推（常用方法），也可以从初状态出发进行记忆化搜索（这个留在后面讨论）。用递推实现，我们不难设计出下面算法：

步骤 1：把所有“胜利终止状态”标记为 P 状态，“失败终止状态”标记为 N 状态。

步骤 2：找到所有的未定状态中，所有后继都被确定是 N 状态的状态，设置为 P 状态。

步骤 3：找到所有的未定状态中，可以一步到达 P 状态的状态，都设置为 N 状态。

步骤 4：若上两步中没有产生新的 P 状态或 N 状态，程序结束，否则回到步骤 2。

上面这个算法是解决一切游戏论问题的普遍方法。不难发现，对于游戏 G(V,E)，这个算法的时间复杂度是 O(|E|)。

但是，这个算法的普遍适应性，意味着它时间效率不高。对于“k 倍动态减法游戏”而言，$|E|=\Theta(S^3)$，$T(e)=\Theta(S)$，所以这个算法相对于 T(e)是立方阶的，并不十分优秀。在 2002 年的集训队论文中提出的对于一般的 k 的算法正是这个 $O(S^3)$的动态规划方案，接下去本文将在此基础上，获得一个相对于 T(e)线性的算法。

四．基于动态规划的算法优化

4．1．基于单调性的方法

4．1．1．“k 倍动态减法游戏”的状态单调性

在“k 倍动态减法游戏”，状态(m,n)是关于 n 单调的。即，若记 $NP(m,n)=\begin{cases}1(if\ \_(m,n)\_is\_an\_N-position)\\0(if\ \_(m,n)\_is\_a\_P-position)\end{cases}$，NP(m,n)关于 n 单调不减。

这是因为若正整数 m_0、n_0 使得 $NP(m_0,n_0)=1$，则对于任意 $n>n_0,m=m_0$ 都有 $NP(m,n)=1$。这是因为，状态(m_0,n_0)时玩家可以完成的操作，在状态(m,n)也允许操作（这利用了 $n>n_0$），且到达相同的状态(这利用 $m=m_0$)。

4．1．2．利用状态单调性优化“k 倍动态减法游戏”

由状态单调性，只需递推计算 f(m)=min{n|NP(m,n)=1}（由于 NP(m,m)=1 所以使得 NP(m,n)=1 成立的 n 是存在的，故可以取最小值）可存储 NP(m,n)的全部信息。

k=1 时的情况，就如图 4-1 所示。注意到，NP(m,n)=0 当且仅当对于任意 r=1,2,3…n 有 m-r>0 且 NP(m-r,kr)=1。所以，若 $n_0=f(m)$则 $NP(m-n_0,kn_0)=0$ 且以下两式中有一个成立：$n_0=1$ 或 $NP(m-n_0+1,k(n_0-1))=1$。即 $f(m-n_0)>kn_0$ 且 $f(m-(n_0-1))<=k(n_0-1)$。这样一来，朴素的实现这一过程只需逐个检验每个 n，第一个使得 f(m-n)>kn 成立的 n 就是 f(m)，即状态转移方程为 f(m)=min{n|f(m-n)>kn}（规定 f(0)为正无穷，因为当 S 剩余 0 时，无论现在能减去多少，都是 P 状态，即 NP(0,n)=0）。因此，计算每个 f(m)的时间复杂度为 O(m)。注意到，后手必胜的充要条件是 f(S)=S，所以算法只需逐次计算 f(1),f(2)…因此总时间复杂度为 $O(S^2)$。

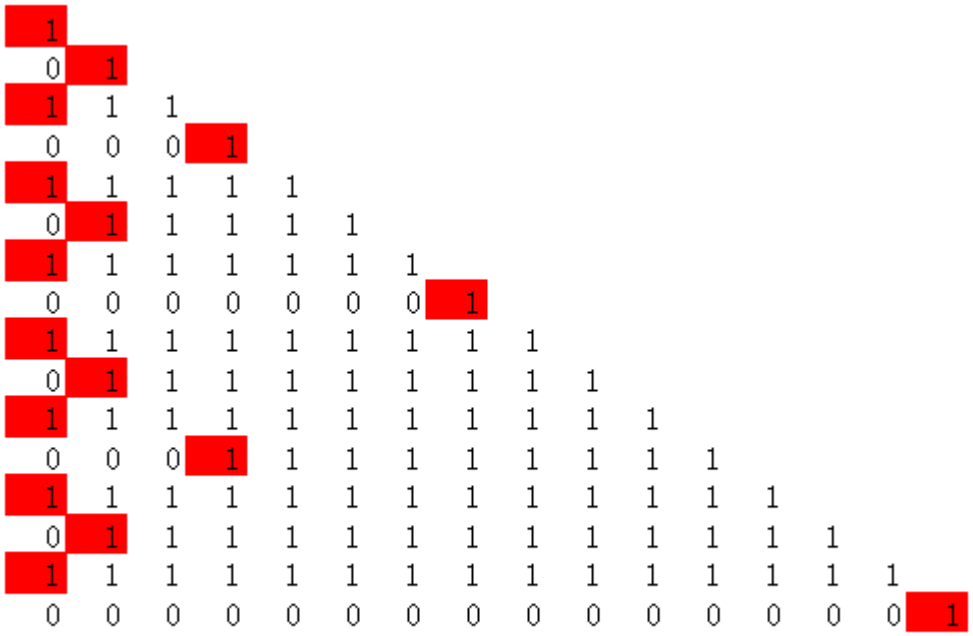

图4-1　k=1时NP(m,n)的情况　　红色格表示f(m)

4．1．3．利用决策单调性进一步优化“k 倍动态减法游戏”

前面将算法从 $O(S^3)$优化到 $O(S^2)$正是由于紧紧抓住了“k 倍动态减法游戏”中的状态单调性。下面的进一步优化用到的是状态转移方程“f(m)=min{n|f(m-n)>kn}”的决策单调性和栈这一数据结构。

这一次，本文为更容易理解，不是用晦涩的数学符号和状态转移方程的变形来表述，而是用下面这种更为形象的方式。如图 4-2 所示（k=1 的情况），如果把 NP(m,n)=0 的格子涂黑，就会在表格上看到黑色个连成的“墙”。由 f(m)=min{n|f(m-n)>kn}不难证明 f(m)就是从(m,0)格子出发，斜率为 k^{-1} 的直线到第一堵他遇到的墙的距离。

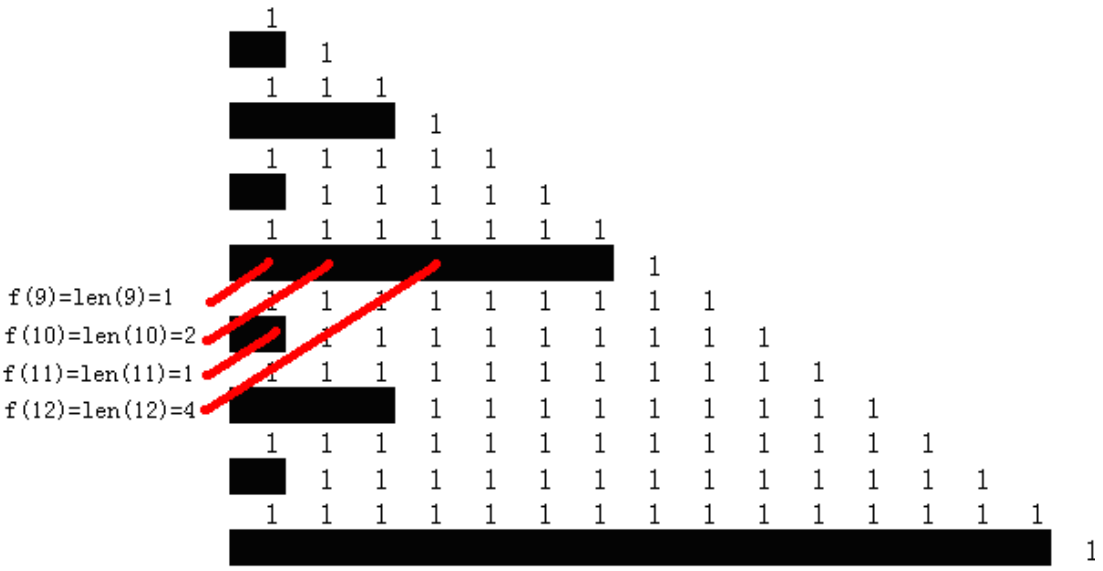

图4-2　k=1时NP(m,n)的情况　　NP(m,n)=0的连续的格子就像一堵堵墙

注意到，所有这些直线是平行的，且随着 m 增大逐渐向下向右移，同时每一堵墙都是固定的、右端有界的。因此，若某一堵墙不能挡住当前的直线，则以后也再也不能挡住直线！这一重要性质，让我们想到用栈这一数据结构来储存“墙”，并给出以下算法：

计算每个 f(m)时，逐个检验栈中的“墙”。若某堵“墙”不能挡住从(m,0)格子出发斜率为 k^{-1} 的直线，那么该“墙”出栈；否则，若这堵“墙”能挡住斜线，则循环结束并得出 f(m)的值。最后，根据 f(m)可确定一堵新“墙”的位置和长度（f(m)=0 的话就没有新“墙”），新“墙”入栈。

这样一来，每计算一个 f(m)至多产生一堵新“墙”，每堵“墙”恰好进栈一次，出栈一次。因此，这个判定胜负的算法的总的时间复杂度为 O(S)，也是关于 T(e)的线形算法！

4．1．4．进一步讨论“k 倍动态减法游戏”的必胜策略的寻找方式

事实上，在计算出所有的 f(m)之后，可以在 O(1)时间内完成必胜策略的寻找。

这是因为根据 f(m)的定义：f(m)=min{n|NP(m,n)=1}，可知在状态(m,n)时（其中 n>=f(m)），当前玩家减去 f(m)就是一个必胜策略（事实上这也是减去数字最小的必胜策略）。所以在线游戏的时间复杂度是 O(S)-O(1)，也就是 O(T(e))-O(1)。

当然对于 k=1 或 2 的特殊情况，可以找到 $O(\log^2 S)$-O(logS)的算法，由于只是利用了数学性质，而不是动态规划中的单调性，这里就不展开了。

4．1．5．小结

至此，“k 倍动态减法游戏”的胜负判定问题以及必胜策略维护问题已获完满解决。总结这个过程，核心就在于算法在“基于 NP 状态定理的动态规划”这一普遍方法的基础上充分利用了问题的特殊性——具体动态规划中的状态单调性和决策单调性。可以说，“k 倍动态减法游戏”是利用单调性优化经典游戏之一。

同时，单调性对游戏论问题的帮助是极具普遍意义的。很多情况下，状态的 NP 属性是分段的，就像“k 倍动态减法游戏”中那样，每一行中的前一部分都是 P 状态，后一部分都是 N 状态。另外，利用各种决策单调性优化动态规划更为常见，在所有动态规划书籍中都能找到。

4．2．基于记忆化搜索实现动态规划的优化模式

本文接下去的几部分，将举一些其它的游戏的例子，说明一些其他的常见的方法。引用的样例游戏有些是来自于经典的组合游戏问题，有一些来自于近些年特别是最近一年的国际上的各大信息学竞赛。

作为除递推之外的动态规划的另一种实现方式，记忆化搜索有着思维复杂度低，编程难度小的优势。在组合游戏的问题中，记忆化搜索有时非常有效。

注意到 NP 状态定理中：$NP(x)=\begin{cases}P(if\forall y\in f(x),NP(y)=N)\\N(if\exists y\in f(x),NP(y)=P)\end{cases}$。因此，若在搜索过程中，只要发现了一个 $y\in f(x)$，使得 NP(y)=P，就可以确定 NP(x)=N，而不需要再搜索 x 的其他后继了。这是重要的“搜索—剪枝”依据。据此，以下两种策略非常实用。

(1) 启发式：选择最有可能是必胜策略的操作，对此操作下的后继状态优先搜索。或者，对最容易判断状态胜负的后继先搜索。当然，对“最有可能是必胜策略”“最容易判断状态胜负”的估价因人而异，也因题而异，没有一个通用的公式。而且，值得一提的是，如果发现一种估价的计算代价，超过了它的剪枝价值，就应该舍弃。

(2) 随机化：顾名思义，就是利用随机函数打乱所有操作的搜索顺序，从而平均化各种操作选择之间的优先性，避免由于数据特殊性，而陷入搜索死胡同。

显然，“启发式”方法与“随机化”方法背道而驰，针对不同的游戏，必须慎重选择。

对比“记忆化搜索”与“递推”这两种实现方式，“记忆化搜索”意味着可

以避免处理许多实际游戏中根本不可能出现的状态（即使不剪枝），实现常数上的优化，同时好的剪枝策略能使搜索中重处理的状态总数大大减少；不过，“记忆化搜索”也意味着，对于基于决策单调性等的只适应与递推实现的优化，只能割爱了。

4．3．基于观察规律的优化

游戏中的一个状态是 N 状态还是 P 状态，如果可以通过一个数学公式简单的计算出，那么所有的时间复杂度将都变成常数。这似乎有些不可思议，但在有些问题中，这样的数学公式是确实存在的，而且十分简洁的。例如，在前面第三章开始处介绍的“捡石子游戏”中，当且仅当，剩余石子数是 4 的倍数是，当前状态是 N 状态。

4．3．1．knight 游戏

这个问题选自 CEOI2008（Central European Olympiad in Informatics）：

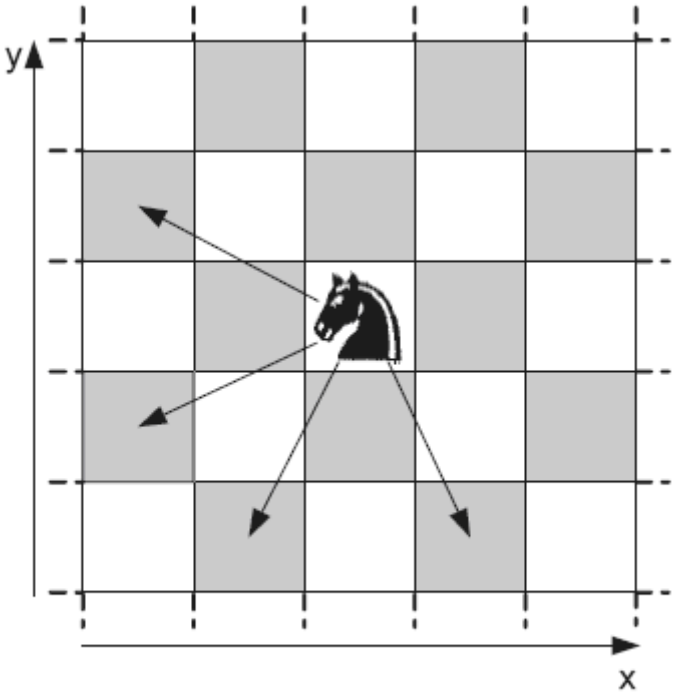

图4-3

CEOI 2008 knight：在一个 N*N 的国际象棋棋盘上，放了 K 只马。双方玩家轮流操作。对于单只马而言，只有图 4-3 中四种移动方向是有效的（不能移出棋盘）。每回和玩家操作时，若没有一匹马可以有效操作，则该玩家负；否则，它必须将所有可以移动的马按规则移动一步，不能移动的马放在原处不动。问，先

行的玩家是否有必胜策略？

4．3．2．从一匹马情况入手的初步分析

先分析简单的情况。如果棋盘上只有一匹马，那么，游戏就变成：双方轮流操作，谁不能操作谁输。利用 NP 状态定理，可以设计出 $O(N^2)$的动态规划算法。注意到，如果马的坐标是(x,y)，那么游戏过程中 x+y 不断减小。所以，这个动态规划的算法有递推实现也是很容易的。

但是，题目中的 N 很大，$O(N^2)$的算法太慢了。于是，尝试观察规律的方法。编出这个 $O(N^2)$的程序，用一些小规模的 N，可以观察出下面的规律：

7	N	N	N	N	N	N	N
6	P	P	N	N	P	P	N
5	P	P	N	N	P	P	N
4	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N
2	P	P	N	N	P	P	N
1	P	P	N	N	P	P	N
	1	2	3	4	5	6	7

N=7的情况

8	N	N	N	N	N	N	N	P
7	N	N	N	N	N	N	N	N
6	P	P	N	N	P	P	N	N
5	P	P	N	N	P	P	N	N
4	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N
2	P	P	N	N	P	P	N	N
1	P	P	N	N	P	P	N	N
	1	2	3	4	5	6	7	8

N=8的情况

9	P	P	P	P	P	P	P	N	P
8	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	N	N	P
6	P	P	N	N	P	P	N	N	P
5	P	P	N	N	P	P	N	N	P
4	N	N	N	N	N	N	N	N	P
3	N	N	N	N	N	N	N	N	P
2	P	P	N	N	P	P	N	N	P
1	P	P	N	N	P	P	N	N	P
	1	2	3	4	5	6	7	8	9

N=9的情况

10	P	P	N	N	P	P	N	N	P	P
9	P	P	N	N	P	P	N	N	P	P
8	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	N	N	N	N
6	P	P	N	N	P	P	N	N	P	P
5	P	P	N	N	P	P	N	N	P	P
4	N	N	N	N	N	N	N	N	N	N
3	N	N	N	N	N	N	N	N	N	N
2	P	P	N	N	P	P	N	N	P	P
1	P	P	N	N	P	P	N	N	P	P
	1	2	3	4	5	6	7	8	9	10

N=10的情况

尽管，N 除 4 的余数不同时，规律是不一样的，但这些不同的规律还是可以从上面这些表格中提炼出来。于是，判定胜负的时间复杂度变成了 O(1)。（虽然这些规律对 K 匹马的原题没有帮助，但只有找到单匹马时的这些规律，才会在 K 匹马的问题中想到有规律可循！）

4．3．3．对 K 匹马的情况的分析

显然，对以每匹单独的马，玩家都应该争取“获胜”，毕竟“获胜”的情况一定不比“输掉”更糟糕。所以，如果某个玩家可以在每匹马上获胜，他就胜定了。

那如果两个玩家都只能在部分“马”上“获胜”呢？根据规则：所有马都不能操作时当前玩家负。所以，在最晚获胜得“马”上“获胜”的玩家胜利！即，玩家的正确策略应该是：赢不了的尽快输掉，赢得了的慢慢地赢。

所以，在 k 匹马的问题中，需要计算的不仅是每一匹马谁胜，还有在多少回和后胜。记坐标(x,y)的马在 Round(x,y)回合后分出胜负。且规定 $RR(x,y)=\begin{cases} Round(x,y)(if\ \_NP(x,y)=N) \\ -Round(x,y)(if\ \_NP(x,y)=P) \end{cases}$。

所以，状态转移方程是：$RR(x,y)=\begin{cases} -\min\{RR(x^*,y^*)\}+1(if\ \_NP(x,y)=N) \\ -\min\{RR(x^*,y^*)\}-1(if\ \_NP(x,y)=P) \end{cases}$

（其中，(x^*,y^*)是(x,y)的后继）。最后，所有马中，RR 绝对值最大的若为正，则先手方胜；否则，后手方胜。虽然，这个动态规划计算的不是 NP 状态，但整个动态规划的流程确实完全类似的。

4．3．4．knight 问题的解决

现在，这个动态规划算法的最大缺陷就是时间复杂度太高——$O(n^2)$。但状态数也是 $O(n^2)$，动态规划已经无法再改进了。于是，想到处理一匹马的问题的方法——观察总结公式。依旧通过编写简单的验证程序，就会得到 Round(x,y)的公式。不过要观察出这个规律，是不容易的。要先通过 Round(x,y)表，猜出 Round 关于 x,y 的线性递增倍数：$\lim_{x\to+\infty,y\to+\infty}\frac{Round(x,y)}{x+y}=\frac{1}{2}$。然后，再计算表格 $Round(x,y)-\left[\frac{x}{2}\right]-\left[\frac{y}{2}\right]$，可以观察到，这个表格除(1,1)-(4,4)和最外层之外其他 4*4 的正方形都是一样的，都是 $\begin{matrix}2&-1&1&0\\1&0&0&1\\0&1&0&-1\\1&0&1&2\end{matrix}$。

和一匹马的游戏类似，最外层的规律对于不同的 N 被 4 除的余数是不同的。但归纳起来的话，对于给定的(x,y)都可以在 O(1)的时间内算出 Round(x,y)，故总时间复杂度为 O(K)。可见观察总结数学公式的威力强大。

不过，这个问题中，本文推荐更为明智的做法：最外层的 Round(x,y)留给动态规划，这样时间复杂度为 O(K+N)。这样，一方面避开了最外层 Round(x,y)的数学公式的冗繁归纳和检验；另一方面，由于本文题中 N 与 K 同阶，所以这样的改进并未增加时间复杂度，反而减少了编程时间，是非常好的变通方式。

从这个例子可以看出，用数学公式不但可以解决 NP 状态的判定，还能计算一些其他的与状态有关的数学量，包括后面一节就要提到的 SG 函数。

4．4．基于 SG 函数以及 SG 定理 Tartan 定理的优化

这一节引进的 SG 函数，本质上来说与上一节提到的“公式”一样，属于状态与状态之间的数学关系。不过，这一种数学关系是十分独特的。2002 年国家集训队关于由游戏论的论文中曾提到“构造一个函数，沟通与 NP 状态之间的关系”说的其实就是 SG 函数。

“Nim 和”与“Nim 积”是利用 SG 函数解游戏论问题的重要工具。但有关“Nim 积”的资料在国内非常少见，本文将在下面第 4．4．5 节给出原创的计算“Nim 积”的算法，并给出时间复杂度的分析与证明。

从第 4．4．1 节至第 4．4．4 节是一些相关概念及定理的介绍，对相关内容熟悉的读者可以跳过这些部分，直接阅读第 4．4．5 节。

另外在这一节中，假设所有探讨的游戏中不存在“失败终止点”，同时下面要

引入一些新的概念。

mex 记号：$mex(S)=\min\{x\mid x\notin S,x\geq 0\}$，其中 S 是一个由非负整数构成的有限集。

SG 函数：对于一个描述游戏的图 G(V,E)，以及游戏规则 f: V→P(V)其中 P(V)表示 V 的子集簇，记 $SG(x)=\begin{cases}0(x\_is\_a\_winning\_end\_position)\\ \\ mex\{SG(y)\mid y\in f(x)\}(other\_situation)\end{cases}$。由 NP 状态定理，不难发现，SG(x)=0 当且仅当 x 是一个 P 状态！

4．4．1．关于“Nim 和”与 SG 定理

4．4．1．1．Nim 和

“Nim 和”运算：即信息学中常说的 xor 运算，仅限于非负整数之间。这里称为“Nim 和”是因为此运算可用以分析著名的 Nim 游戏，且与后面会介绍的“Nim 积”相对应。本文中统一用⊕符号表示。$a\oplus b$ 的运算法则是：对于 a,b 的每一个对应的 2 进制位如下表计算：

a	b	$a\oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

由运算法则可以直接推出下列⊕运算的运算定律：(1)交换律 $a\oplus b=b\oplus a$；(2)结合律 $(a\oplus b)\oplus c=a\oplus(b\oplus c)$；(3)自反律 $a\oplus a=0$，$a\oplus 0=a$。由这些运算性质可知，方程 $a\oplus x=b$ 有且只有唯一解 $x=a\oplus b$。

简单证明如下：

$a\oplus x=b$

$\Leftrightarrow x\oplus a=b$

$\Leftrightarrow (x\oplus a)\oplus a=b\oplus a$

$\Leftrightarrow x\oplus(a\oplus a)=b\oplus a$

$\Leftrightarrow x\oplus 0=b\oplus a$

$\Leftrightarrow x=b\oplus a$

4．4．1．2．SG 定理

定义：n 个游戏 $G_1G_2…G_n$ 的和 $G(V,E)=G_1+G_2+...+G_n$ 为：

(i) $V=V_1\times V_2\times...\times V_n$ (笛卡尔积)

(ii) $E=\{xy\mid x=(x_1,x_2...x_n),y=(y_1,y_2...y_n),x_k y_k\in E_k,k=1...n,y_i=x_i,\forall i\neq k\}$

也就是说，G 相当于每次在 $G_1G_2…G_n$ 中选出一个游戏，在其中操作一次，最后不能操作的一方获胜（因为，这 n 个游戏都不包含“失败终止状态”）。

SG 定理：设 $G(V,E)=G_1+G_2+..+G_n$，$x=(x_1,x_2...x_n)\in V$，则 $SG(x)=SG(x_1)\oplus SG(x_2)\oplus...\oplus SG(x_n)$。

这个定理的证明可以在附录中找到。显然，利用这个定理，“基于 NP 状态定理的动态规划”的时间复杂度从 $O(E)=O(\prod_{i=1}^{n}E_i)$，改进到只需对每个单游戏分别动态归化计算 SG 函数即可，总时间复杂度只有 $\sum_{i=1}^{n}O(E_i)$。改进明显！

4．4．1．3．Nim 游戏

有时，每个子游戏的 SG 函数可以利用前面提过规律观察得到简明的数学公式，每个计算的时间复杂度可以降到对数阶，甚至常数阶，例如经典的 Nim 游戏就是这样的。

Nim 游戏：有 k 堆石子，各包含 $x_1,x_2…x_k$ 颗石子。双方玩家轮流操作，每次操作选择其中非空的一堆，取走其中的若干颗石子（1 颗，2 颗…甚至整堆），取走最后一颗石子的玩家获胜。

不难发现，Nim 游戏可以被理解成 k 个这样的游戏的和：有一堆石子，双方轮流操作，每次操作取走其中的若干颗石子，1 颗，2 颗…甚至整堆，取走最后一颗石子的玩家获胜。虽然这个本身，没有实际意义，因为无论如何，先行者只需直接取走整堆，就可获胜。也就是说，$NP(x)=\begin{cases}P(if\_x=0)\\N(if\_x>0)\end{cases}$。但他的 SG 函数意义重大！逐次计算得：SG(0)=0, SG(1)=mex(0)=1, SG(2)=mex(0,1)=2…可总结出：SG(x)=x。

所以，由 SG 定理和 SG 函数与 NP 状态的关系，Nim 游戏先手胜当且仅当 $x_1 \oplus x_2 \oplus ... \oplus x_k \neq 0$。所以“Nim 游戏”的胜负判定的时间复杂度为 $O(k)$。（出于实际，不考虑高精度计算，毕竟 10^9 颗石子或更多石子的“Nim 游戏”很有可能这辈子结束不了了）。

4．4．1．4．Nim 游戏的必胜策略

进一步，能否在 $O(k)$ 时间内找到“Nim 游戏”的必胜策略？答案是肯定的！因为必胜策略只需将 $(x_1, x_2, ..., x_k) \rightarrow (y_1, y_2, ..., y_k)$ 使得 $y_1 \oplus y_2 \oplus ... \oplus y_k = 0$。下面算法是可行的：

(i)计算 $X = x_1 \oplus x_2 \oplus ... \oplus x_k$，$X \neq 0$。

(ii)找到一个 i，使得 X 在 2 进制下的最高位（假设是 2^α 位）在 x_i 中为 1。（由于 $X = x_1 \oplus x_2 \oplus ... \oplus x_k$，所以必定存在这样的 i）

(iii)令 $y_i = x_i \oplus X$，$y_j = x_j$，$j \neq i$。（因为 y_i 的 2^α 位为 0，x_i 的 2^α 位为 1，且两者更高的位都相同，所以 $y_i<x_i$，所以，该操作合法。）

4．4．1．5．与对手交互 Nim 游戏的维护

作为一个求单次必胜策略的算法，上面一节中的 O(k)是一个好算法，因为在 Nim 游戏中 k<T(e)。不过，如果把这个算法用来在线游戏，时间复杂度就变成 O(k)-O(k)，这不能令人满意，因为这意味着总时间可能达到 T(e)的平方阶（这里用到引言中提到的评估在线的游戏论算法的时间复杂度的尺度）。因此，优化是必须的。

要减少在线部分的时间复杂度，就要减少重复的冗余计算。原算法中步骤(i)和步骤(ii)时间复杂度都是 O(k)，因此都是算法的时间瓶颈。分而治之！

对于步骤一，不难发现，当前状态与当前玩家上一次操作的状态中，最多有两堆石子数量不相同，假设是 $x_i \rightarrow y_i, x_j \rightarrow y_j$，于是 $Y = X \oplus x_i \oplus x_j \oplus y_i \oplus y_j$。时间复杂度改进到了 O(1)。

对于步骤二，可以用 $\log(\max\ x_i)$ 个双向链表，储存对于每一个 α，哪些 i 使得 x_i 的 2^α 位为 1。每次维护的时间是 $O(\log(\max\ x_i))$。

所以，在线算法的时间复杂度为 $O(k \bullet \log(\max\ x_i)) - O(\log(\max\ x_i))$。事实上，对于“和”游戏必胜策略的寻找，上面这种利用双向链表的方法是十分常见的手段。

Nim 游戏是十分经典的游戏，许多著名的数学家设计了许多 Nim 游戏的变种，这些游戏都可以转化为 Nim 游戏来解决，Nim 游戏的变种的例子可以在附录中查到。

利用 SG 定理可以解决许多“和”游戏。下面一个问题留给读者。

有 k 堆石子，各包含 $x_1,x_2 \ldots x_k$ 颗石子。双方玩家轮流操作，每次操作选择其中非空的一堆——第 i 堆，取走其中的若干颗石子，1 颗，2 颗…或 i 颗。取走最后一颗石子的玩家获胜。

（其他一些在题库网站上有，附录上也列有）

4．4．2．SG 定理的嵌套使用

事实上，SG 定理的使用相当灵活，组成“和”游戏的每个单游戏，本身也可以是复杂的。下面是一个著名的游戏。

n 枚棋子排成一排，两个玩家轮流操作，每次可以取走一颗棋子，或者连续的两颗棋子（所谓连续是指中间没有空格，例如 2 号棋子取走后，不能同时取走 1 号 3 号棋子）。取走最后一枚棋子者胜。

图4-4

这个游戏初看状态表达复杂，而且难以表示成游戏“和”的形式。事实上，这只是形式不同而已。考虑，第一个玩家操作以后的情况。要么在左侧或右侧顶端取走一颗或两颗棋子，这样操作得到的后继的 SG 函数分别是 SG(n-1)，SG(n-2)；要么在中间取走棋子，如图 4-4 所示，这样一来，整排棋子分成两段，但这两段至此不相关了，于是游戏成为了两个相互独立的游戏的“和”。于是可以给出递推求 SG 函数的递推式：SG(n)=mex{SG(n-1),SG(n-2), $SG(i) \oplus SG(n-i-1), SG(i) \oplus SG(n-i-2) | i=1,2 \ldots n\}$。

为便于理解，这里将 n 比较小的情况罗列出来：

SG(0)=0

SG(1)=mex(SG(0))=1

SG(2)=mex(SG(0),SG(1))=2

SG(3)=mex(SG(1),SG(2), $SG(1)\oplus SG(1)$)=3

SG(4)=mex(SG(2),SG(3), $SG(1)\oplus SG(1)$, $SG(1)\oplus SG(2)$)=1

SG(5)=mex(SG(3),SG(4), $SG(1)\oplus SG(3)$, $SG(2)\oplus SG(2)$, $SG(1)\oplus SG(2)$)=4

发现规律，当 n>0 时，SG(n)>0，也就是说，无论如何，都先手必胜？！这个结论是正确的，因为先手方只需在第一次操作时取走中间的 1 颗或 2 颗（1 颗还是 2 颗由 n 的奇偶性决定），把整排棋子变成对称的两堆就可以确保获胜了。尽管如此，这一个问题中计算 SG 函数的方法是很具有代表性的。同时，这个问题也告诉我们，解决组合游戏问题时千万不要将思路局限于 NP 状态定理得动态规划，或者计算 SG 函数！后面的第 4 章会给出一些这样的例子。

4．4．3．SG 定理的另一种推广——“N 阶 Nim 和”

除了上一节中提到的，通过游戏“和”的嵌套推广“Nim 和”运算外，“Nim 和”还有一种重要推广：“N 阶 Nim 和”。

N 阶 Nim 游戏：有 k 堆石子，各包含 $x_1,x_2\ldots x_k$ 颗石子。双方玩家轮流操作，每次操作选择其中非空的若干堆，至少一堆但不超过 N 堆，在这若干堆中的每堆各取走其中的若干颗石子（1 颗，2 颗…甚至整堆），数目可以不同，取走最后一颗石子的玩家获胜。

结论：当且仅当在每一个不同的二进制位上，$x_1,x_2\ldots x_k$ 中在该位上 1 的个数是 N+1 的倍数时，后手方有必胜策略，否则先手必胜。

这个结论的证明与 SG 定理的证明类似，由于本文是一篇信息学论文，这一个证明不是本文的重点，因此这里不再展开。具体的证明可以在相关文献上查找到。

注意，N 阶情况下，子游戏的 SG 函数只能用来分析胜负，并求必胜策略，而不能求出“N 阶和游戏”的 SG 函数。也就是说，二进制位下分别加和知识得到一组数，而非一个数。

这是十分重要的，因为正是因为此，在 N>1 时不能嵌套！原因还在于 SG 定理（可以理解成 1 阶“和”游戏的胜负判定）的证明中，每一个简单游戏都可以被理解成一个 1 阶“和”游戏（即，一个简单游戏与一个 1 阶“和”游戏性质相同）。但一个简单游戏与一个 N 阶“和”游戏性质显然是不同的，因此，当 N>1 时嵌套形式的推广不适应于 N 阶 Nim 和。

4．4．4．Nim 积

4．4．4．1．翻硬币游戏 I

翻硬币游戏 I：在一个 n+1 行 m+1 列的表格里，坐标从(0,0)标记到(n,m)，每一个格子中有一枚硬币，或正面朝上，或反面朝上。两个玩家轮流操作，每次操作同时翻 2 枚同行或同列硬币，如图 4-5 种那样，但要求，其中坐标较大的一枚硬币必须是从正面翻到反面的。即，当(x,y)正面向上的时候，可以同时翻(x,b)或(a,y)（其中 a<x，b<y）。双方如此不断操作，直至有一方无法操作为止，不能操作的人负，即不能操作的状态（只有(0,0)有正面向上的硬币，或者整个表格的硬币都反面向上）是胜利终止状态。

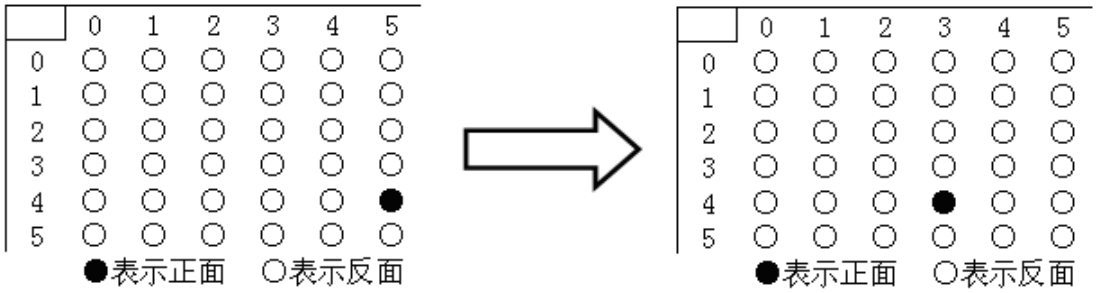

图4-5

这个问题有点复杂，用逐步简化的方法来分析它。首先，如果只有一行，即 n=0，每一枚向上硬币可以被理解成一个单独的游戏，于是游戏似乎就变成这些单独游戏的“和”，因为翻两枚硬币可以理解成把“坐标大”的正面硬币移到了坐标小的位置。唯一一点区别是，同时翻两枚正面向上的硬币，就相当于坐标小的被坐标大的硬币“消掉”了。这其实与游戏“和”模型是不矛盾的，因为两个相同位置的正面硬币的 SG 函数值必相同，因此他们的 Nim 和为 0，所以等于“没有”。

回到原来问题，显然问题也归结为若干个单独游戏的和——若干个单独的证面向上的硬币。在只包含一行的游戏中，一颗正面向上的硬币的 SG 函数就是它的列坐标。那么，行数不为一的情况下，SG 函数至如下：

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	0	3	2	5	4
2	2	3	0	1	6	7
3	3	2	1	0	7	6
4	4	5	6	7	0	1
5	5	4	7	6	1	0

可以发现规律，SG(x,y)= $x \oplus y$ 。这是因为，每个单个的棋子也可以理解成两堆石子的 Nim 游戏，即行坐标表示一堆，列坐标表示另一堆。下一个游戏更复杂，也将引出这一节的主题“Nim 积”。

4．4．4．2．翻硬币游戏 II

翻硬币游戏 II：在一个 n+1 行 m+1 列的表格里，坐标从(0,0)标记到(n,m)，每一个格子中有一枚硬币，或正面朝上，或反面朝上。两个玩家轮流操作，每次操作同时翻 4 枚在一个矩形（边长平行于行列的矩形）四顶点的硬币，而且要求，其中行列坐标都较大的一枚硬币必须是从正面翻到反面的。即，当(x,y)正面向上的时候，可以同时翻(x,y)(x,b)(a,y)(a,b)（其中 a<x，b<y）。双方如此不断操作，直至有一方无法操作为止，不能操作的人负，即不能操作的状态（只有第 0 行和第 0 列有正面向上的硬币，或者整个表格的硬币都反面向上）是胜利终止状态。

类似翻硬币游戏 I 中的分析，每枚正面向上的硬币可以理解成一个独立的简单游戏，而在这个游戏中，每次操作能加一个简单游戏拆分成 3 个游戏的“和”。这是游戏“和”的嵌套，前文已经介绍过这样的 SG 函数的求法。这里，略去具体的计算过程，通过下表给出 SG 函数的计算结果。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	2	3	1	8	10	11	9	12	14	15	13	4	6	7	5
3	0	3	1	2	12	15	13	14	4	7	5	6	8	11	9	10
4	0	4	8	12	6	2	14	10	11	15	3	7	13	9	5	1
5	0	5	10	15	2	7	8	13	3	6	9	12	1	4	11	14
6	0	6	11	13	14	8	5	3	7	1	12	10	9	15	2	4
7	0	7	9	14	10	13	3	4	15	8	6	1	5	2	12	11
8	0	8	12	4	11	3	7	15	13	5	1	9	6	14	10	2
9	0	9	14	7	15	6	1	8	5	12	11	2	10	3	4	13
10	0	10	15	5	3	9	12	6	1	11	14	4	2	8	13	7
11	0	11	13	6	7	12	10	1	9	2	4	15	14	5	3	8
12	0	12	4	8	13	1	9	5	6	10	2	14	11	7	15	3
13	0	13	6	11	9	4	15	2	14	3	8	5	7	10	1	12
14	0	14	7	9	5	11	2	12	10	4	13	3	15	1	8	6
15	0	15	5	10	1	14	4	11	2	13	7	8	3	12	6	9

前面已经知道，在“翻硬币游戏 I”中 SG(x,y)= $x \oplus y$，而这个“翻硬币游戏 II”的 SG 函数显然更为复杂，记这个游戏的 SG 函数，SG(x,y)= $x \otimes y$——Nim 乘法运算。

也就是说 $x \otimes y = mex\{(a \otimes y) \oplus (x \otimes b) \oplus (a \otimes b) \mid 0 \le a < x, 0 \le b < y\}$。这一定义式和游戏模型的定义是等价的，而且是所有下面将提到的运算定律和运算性质的重要证明依据和唯二证明途径。

4．4．4．3．Tartan 定理

上面这个运算之所以被称之为“乘法”运算，是因为它可以向 SG 定理一样做如下推广：

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

(i) $V = V_1 \times V_2 \times \ldots \times V_n$ (笛卡尔积)

(ii) $E = \{(xy) \mid x_i^* \in f_i(x_i)\}$，其中 $x = (x_1, x_2 \ldots x_n), y = \sum_{y_i \in \{x_i, x_i^*\}} (y_1, y_2 \ldots y_n)$。这里 $\sum$ 号代表相同游戏规则下，不同状态所表示的不同游戏的和。

那么，$SG(x) = SG(x_1) \otimes SG(x_2) \otimes \ldots \otimes SG(x_n)$，其中 $x = (x_1, x_2 \ldots x_n) \in V$。

4．4．4．4．“Nim 积”的运算定律

$x \otimes y$ 满足以下运算定律：

(1)单位元：$x \otimes 1 = 1 \otimes x = x$；

(2)交换律：$x \otimes y = y \otimes x$；

(3)结合律：$(x \otimes y) \otimes z = x \otimes (y \otimes z)$；

(4)对 $\oplus$ 的分配律：$(x \oplus y) \otimes z = (x \otimes z) \oplus (y \otimes z)$。（类似于，普通乘法与普通加法的关系）

4．4．5．Nim 积的计算方法和时间复杂度估计

4．4．5．1．Nim 积的重要运算性质

朴素的计算 Nim 积$x\otimes y$，需要先求出所有$(a\otimes y)\oplus(x\otimes b)\oplus(a\otimes b)$，其中$0\le a<x,0\le b<y$。这样的时间效率相当低下。要快速高效的计算出的$x\otimes y$，除了要利用上一节的一些运算律，还要利用一些性质，这些运算律和运算性质的证明并不容易，更多是数学的推导，与信息学竞赛无关，故本文不作介绍。

Nim 积的性质：对于$x,y<2^{2^\alpha}$：

(1) $x\otimes 2^{2^\alpha}=2^{2^\alpha}x$；

(2) $x\otimes y<2^{2^\alpha}$；

(3) $2^{2^\alpha}\otimes 2^{2^\alpha}=\frac{3}{2}\cdot 2^{2^\alpha}$。

4．4．5．2．利用运算定律和运算性质计算 Nim 积的实例

接下来先给出一些计算的例子，再总结成算法，并估计时间复杂度。

(1)计算$24\otimes 17$

$$24\otimes 17=(16\oplus 8)\otimes(16\oplus 1)=(16\otimes 16)\oplus(16\otimes 8)\oplus(16\otimes 1)\oplus(8\otimes 1)$$

$$=24\oplus 128\oplus 16\oplus 8=128$$

(2)计算$8\otimes 8$

$$8\otimes 8=(2\otimes 4)\otimes(2\otimes 4)=(2\otimes 2)\otimes(4\otimes 4)=3\otimes 6$$

$$=(1\oplus 2)\otimes(4\oplus 2)=(1\otimes 4)\oplus(1\otimes 2)\oplus(2\otimes 4)\oplus(2\otimes 2)=4\oplus 2\oplus 8\oplus 3=13$$

4．4．5．3．计算 Nim 积的算法

其实，上面这两个例子指引出了计算$x\otimes y$的算法。

下面将给出的算法的基本思路是递归：当需要计算$x\otimes y$时（不妨假设$x\ge y$），把 x,y 表示成这样的形式：$x=pM+q$，$y=sM+t$，其中$2^{2^a}\le x<2^{2^{a+1}},M=2^{2^a}$。这样一来，利用分配律和性质（1）（3），就只需要计算$M=2^{2^a}$更小的 Nim 积，即α更小的时候的 Nim 积。

同时，一个小的优化对时间复杂度的解释重要的：利用运算性质（1）（3）的时候会产生许多，计算 $x \otimes y$，其中 x 是 2 的幂次的需要。而这种情况较一般的情况要简单，耗时也少，因此单独坐一个字程序。

下面就给出这个算法，这个算法在时间上是最优的，其他一些实现方便的朴素方法可以在附录中查到。

用递归函数 Nim_Multi(x,y)来计算，Nim_Multi_Power(x,y)是一个辅助的递归函数，处理 x 是 2 的幂次的情况。

Nim_Multi(x,y)：

1.若 x<y，返回 Nim_Multi(y,x)

2.若 x<16，查表并返回值

3.找到正整数 a 使得 $2^{2^a} \le x < 2^{2^{a+1}}$。设 $M = 2^{2^a}$。

4.将 x,y 表示成：$x = pM + q$, $y = sM + t$。其中 $p,q,s,t \in N, 0 \le q,t < M$。

5.递归计算：

c_1=Nim_Multi(p,s)

c_2=Nim_Multi(p,t)$\oplus$Nim_Multi(q,s)

c_3=Nim_Multi(q,t)

6.返回$(c_1 \oplus c_2)M \oplus c_3 \oplus$Nim_Multi_Power($\frac{M}{2}$,$c_1$)

Nim_Multi_Power(x,y)：（这个函数确保 x 是一个 2 的幂）

1.若 x<16，查表并返回值

2.找到正整数 a 使得 $2^{2^a} \le x < 2^{2^{a+1}}$。设 $M = 2^{2^a}$。

3.将 x,y 表示成：$x = PM$, $y = SM + T$。

4.递归计算：

d_1=Nim_Multi_Power (P,S)

d_2=Nim_Multi_Power (P,T)

5. 返回 $M(d_1 \oplus d_2) \oplus$Nim_Multi_Power($\frac{M}{2}$,d_1)

4．4．5．4．算法的时间复杂度估计：

设 f(a)表示在最坏情况下，依据上述算法计算 Nim_Multi(x,y)的时间复杂度，其中 $0 \le x, y < 2^{2^{a+1}}$。

设 g(a)表示在最坏情况下，依据上述算法计算 Nim_Multi_Power(x,y)的时间复杂度，其中 $0 \le x, y < 2^{2^{a+1}}$，且 x 是一个 2 的幂。

则 f(n)=4f(n-1)+g(n-1)，g(n)=3g(n-1)。

由第 2 式知：$g(n)=O(3^n)$。代入前一式，得到：$f(n)=4f(n-1)+3^{n-1}$。

即：$f(n)+3^n=4f(n-1)+3^n+3^{n-1}$

$f(n)+3^n=4f(n-1)+4\cdot 3^{n-1}$

所以：$f(n)+3^n=4(f(n-1)+3^{n-1})$

所以：$f(n)=O(4^n)-O(3^n)=O(4^n)$

故，这个递归算法的时间复杂度是 $O(4^{\log\log \max(x,y)})$，即 $O(\log^2 x)$。

4．4．5．5．算法的证明：

Nim_ Multi_Power(x,y)= $PM \otimes (SM+T)$

$=(P\otimes M)\otimes((S\otimes M)\oplus T)$

$=((M\otimes M)\otimes(P\otimes S))\oplus(M\otimes(P\otimes T))$

$=\left(\left(M\oplus\frac{M}{2}\right)\otimes(P\otimes S)\right)\oplus(M\otimes(P\otimes T))$

$=\left(\frac{M}{2}\otimes(P\otimes S)\right)\oplus\left[M\otimes\left((P\otimes T)\oplus(P\otimes S)\right)\right]$

$= M(d_1\oplus d_2)\oplus \text{Nim\_Multi\_Power}(\frac{M}{2}, d_1)$

Nim_Multi(x,y)= $(pM+q)\otimes(sM+t)$

$=((p\otimes M)\oplus q)\otimes((s\otimes M)\oplus t)$

$=((M\otimes M)\otimes(p\otimes s))\oplus(M\otimes(p\otimes t))\oplus(M\otimes(q\otimes s))\oplus(q\otimes t)$

$=\left[\left(M\oplus\frac{M}{2}\right)\otimes c_1\right]\oplus\left(M\otimes c_2\right)\oplus c_3$

$=\left[(c_1\oplus c_2)\otimes M\right]\oplus\left(\frac{M}{2}\otimes c_1\right)\oplus c_3$

$=(c_1\oplus c_2)M\oplus c_3\oplus \text{Nim\_Multi\_Power}(\frac{M}{2}, c_1)$

同时注意到 P, $\frac{M}{2}$ 都是 2 的幂，故算法正确性获证！必须指出，在与“Nim 积”有关的游戏的问题中，这个计算 $x \otimes y$ 的算法是至关重要的。

五．不基于动态规划的问题思考

基于 NP 状态定理的动态规划方法是解决游戏论问题的普遍方法，本文前面的部分介绍了许多对动态规划解法的优化。不过，有些问题终究是不能有动态规划解决的，或者说从 NP 状态的角度分析只能的时间复杂度非常高的算法。前面在 SG 函数的一节中就已经有一个被分析 NP 状态和计算 SG 函数“误入歧途”的例子。这一章再给出几个这样的例子，此间的分析方法，仅作抛砖引玉之效。

5．1．对称化分析

所谓对称化，就是顾名思义，就是设法使两方的情况对称起来。这样对方能操作，自己也能操作，对于“不能操作就胜”的问题，对称化就意味着自己站在了不败之地。

下面这一个例子是一个简单的问题，它选自 2008 年的 US open。

US open 2008 the Ultimate Battle：一个战场被分割成 W*H 的网格，游戏双方开始时，站在两个不同行也不同列的的格子里。双方轮流操作。每个玩家每次操作，可以水平的或垂直的移动若干格，不能走出战场。另外，两个人手中都是有枪的，而只有双方在同一水平线上，或同一垂直线上时，才能开枪。因此，“移动”操作不能穿过对方所在的水平线或垂直线。问谁有必胜策略？

这个例子可以转化为只有两堆石子的 Nim 游戏来分析，不过问题远没有那么复杂。因为，事实上，要获胜，只要维持操作完之后，双方横纵坐标之间的差距相等就行了！

具体地说，如果对方沿横（纵）坐标倒退，只需沿横（纵）坐标前进同样的格子；如果对方沿横（纵）坐标前进，只需沿纵（横）坐标前进同样的格子。这样的方法，无非就是对称化分析。注意，由于“你”始终是前进的，所以，对手到最后必将没有合法的操作。

可以说，“对称化”的分析是一种以不变应万变的想法，也就是说无论对手的一次操作将局面变得多复杂，自己都能通过一次操作是游戏回到简单且有利于自己的状态。上面的 US open 2008 the Ultimate Battle 就是极好的例子。

5．2．贪心分析

5．2．1．贪心思想在游戏论的应用浅析

贪心是一种信息学中的普遍思想。再游戏论中，贪心是限制操作的可选择范围的优秀手段。与动态规划相对比，贪心的思路是“我怎样操作，目前会变得更好”，而动态规划是“不同的操作，分别会怎样——递归计算出来”。

贪心在游戏论中的常见应用方法是：如果操作 a 的结果一定不比操作 b 更好，那么一定不选择操作 a。

5．2．2．BOI 2008 Game

BOI 2008 Game：一个 n*n 的棋盘，每个格子要么是黑色要么是白色。白格子是游戏区域，黑格子表示障碍。指定两个格子 AB，分别是先手方和后手方的起始格子。A 和 B 这两格子不重合。游戏中，双方轮流操作。每次操作，玩家向上下左右四个格子之一走一步，但不能走进黑色格子。有一种特殊情况，当一方玩家，恰好走到当前对方所在的格子里，他就可以再走一步（不必是同一方向），“跳过对手”。胜负的判定是这样的，若有一方走进对方的起始格子，就算获胜，即使是跳过对方，也算获胜。输入一个棋盘和双方开始位置，判定胜负归属。

这个问题 BOI 给出的官方解答是时间复杂度 $O(n^3)$的，算法的思想就是贪心。然而，官方解答中关于时间复杂度是 $O(n^3)$的估计是有明显的逻辑漏洞的，事实证明也是错误的。本文将找到一个使得官方解答需要 $O(n^{3.5})$时间的例子。

不过，本文不会拘泥于官方解答的思路或者错误，而会进一步深入分析，并最终得到一个 $O(n^2)$的算法。

5．2．2．1．基于 NP 状态的初步分析

如果用 NP 状态来分析，那么状态应该这样表示：(x_1,y_1,x_2,y_2)。其中(x_1,y_1)是 A 的当前位置，(x_2,y_2)是 B 的当前位置。另外，还需要一位状态表示当前的操作这是 A 或 B。（事实上，若利用奇偶性可以把这一维省掉，但由于这一维是常数的，这样做的意义并不大）

因此，状态总数至少为 $O(n^4)$个，尽管每个状态的状态转移代价为 O(1)，但总时间复杂度为 $O(n^4)$，太高了。

而且状态数为 $O(n^4)$也意味着动态规划已经没有优化的余地，算法的设计必须跳出动态规划的框架。

5．2．2．2．贪心的思路

由于胜负判定条件是“先到达对方起始点着胜”，这个“先”是贪心的信号。

所以，事实上，游戏的双方都应该尽快地到达对方起始点！即，两人都应沿着两起始点间的最短路径走，这正是贪心的思想。注意到，两个人需要走的路程是相等的，所以如果没有“跳过对手”的规则，先行者将必胜！

因此，如果先行方 A 能避免 B“跳过 A”，则 A 获胜；如果后手方 B 能确保在最短路径上“跳过 A”，则 B 获胜。

5．2．2．3．BOI 官方解答

有了这样一层分析，AB 两方的决策数就会少很多，进一步，游戏中可能出现的状态也少很多。因此，BOI 的官方解答选择回到基于 NP 状态定理的动态规划，采用下面的算法（具体的符号与 BOI 官方 booklet 略有区别）。

记 d 为 AB 之间最短路的距离。用数组 LA_i 存贮，在 AB 最短路径上，且与距离 A 为 i 的格子。记 NP_A[i,j,k]表示，轮到 A 操作时，A 在 LA_i 中的第 j 个格子上，B 在 LA_{d-i} 中的第 k 个格子上的状态。NP_B[i,j,k]表示，轮到 B 操作时，A 在 LA_{i+1} 中的第 j 个格子上，B 在 LA_{d-i} 中的第 k 个格子上的状态。

于是，BOI 的官方解答认为，由 NP 状态定理和上面的状态描述就可以设计出 $O(n^3)$的动态规划算法了，其中状态数 $O(n^3)$，每个状态的状态转移代价为 O(1)。不仅如此，BOI 的官方解答还提出，可以利用滚动数组，优化空间。所以只需存储包含 j,k 两维的数组，于是空间复杂度为 $O(n^2)$。

BOI 的 booklet 只给出了上面算法，单位给出证明，事实上，这些上界无法被证明。本文接下去将指出，时间复杂度 $O(n^3)$和空间复杂度 $O(n^2)$这两个上界都是不正确的。

5．2．2．4．BOI 官方解答的错误

BOI 的官方解答中认为，数组 NP_A[i,j,k]和数组 NP_B[i,j,k]表示的状态总数为 $O(n^3)$数量级。但是，形式上的三位数组并不等于包含的数据为立方阶。事实上，这三位都不是 O(n)的。

首先，虽然大多数情况下 $d=\Theta(n)$，然而 d 是 $O(n^2)$的，图 5-1 是使得 d 是 $\Theta(n^2)$的例子。

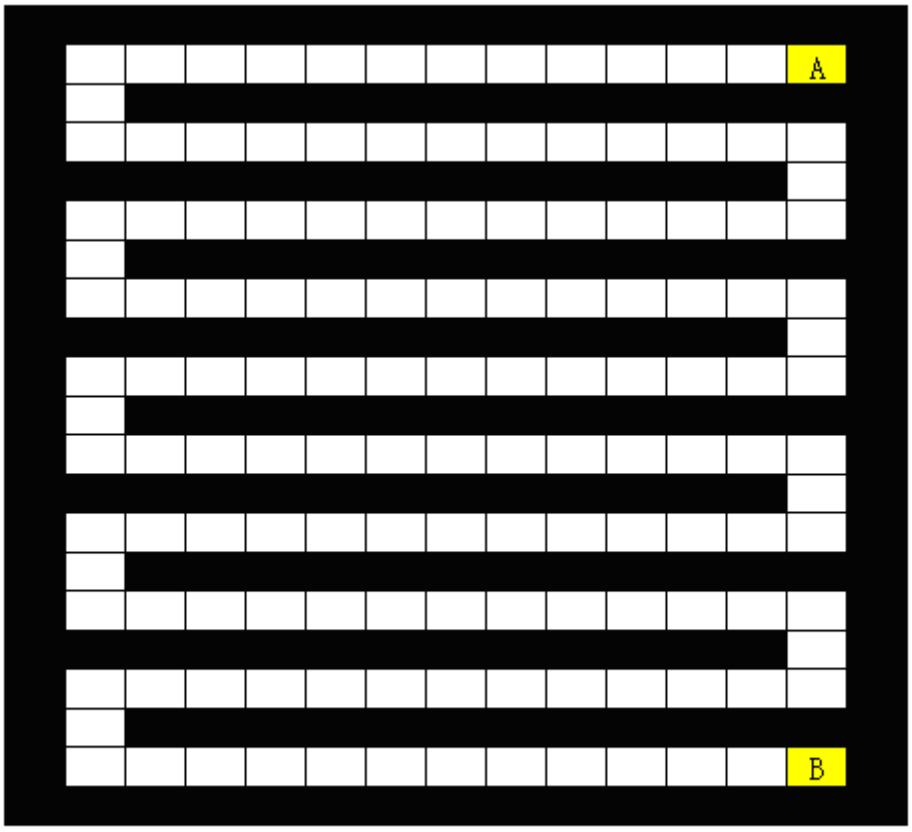

图5-1 d是n的平方阶的例子

其次，$|LA_i|$=O(n)的结论也是不正确的，BOI 官方解答中甚至认为$|LA_i| \le n$。图 5-2 这个例子虽不能否定$|LA_i|$=O(n)，但已使得 $\lim_{n\to+\infty} \frac{\max|LA_i|}{n} = 4$。同时，下一节即将给出的完整的反例也是从这个例子的图形联想到的（完整的反例中 $\max|LA_i|$是$\Theta(n^{1.5})$的）。当 n 变大时，周围的“通道”是不变的，中间的四块含黄色方格的矩形区域不断随 n 增大。同时，所有黄色的方格距 A 是相等的，所有黄色的方格的数量是 4n+o(n)。

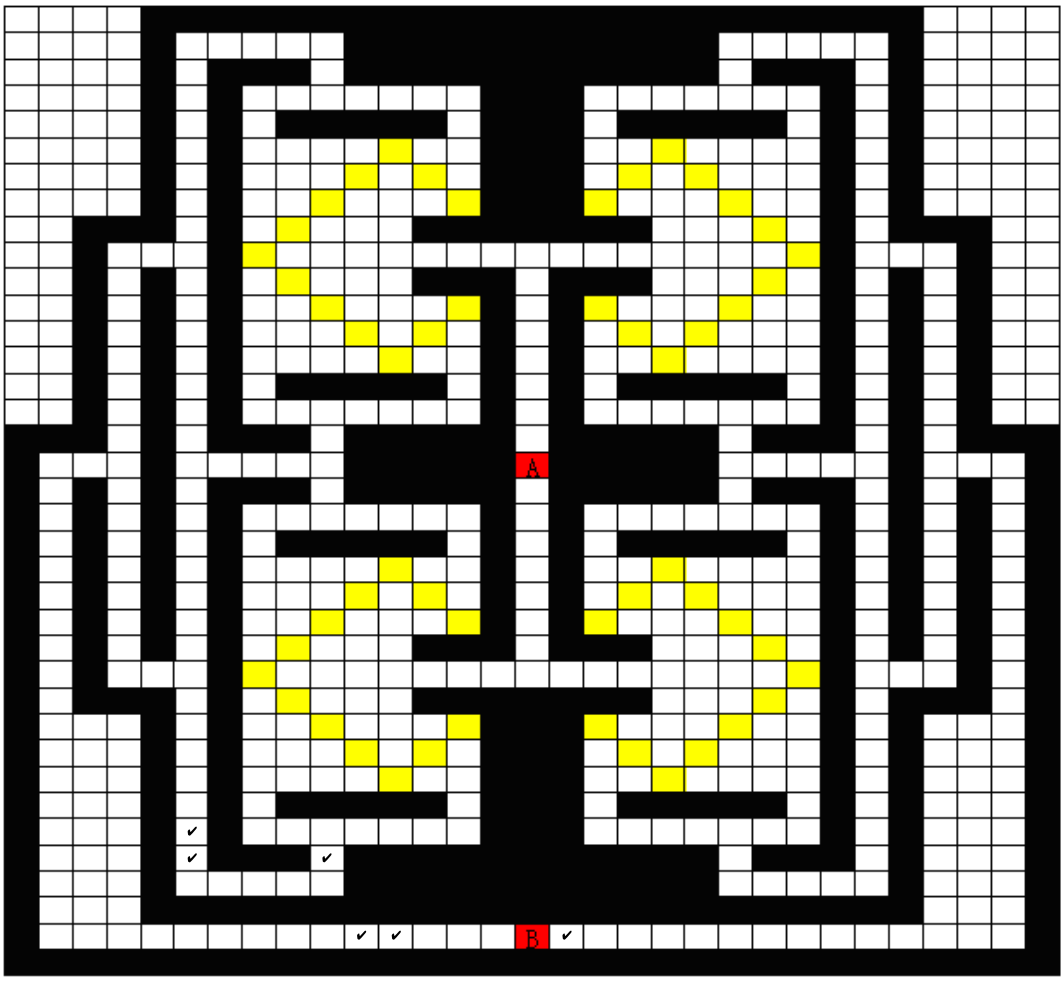

图5-2

所以，数组 NP_A[i,j,k]和数组 NP_B[i,j,k]的三维都不是 O(n)的。时间复杂度的平凡上界是 $O(n^2 \cdot n^{1.5} \cdot n^{1.5}) = O(n^5)$，空间复杂度的平凡上界是 $O(n^{1.5} \cdot n^{1.5}) = O(n^3)$。（这两个上界不是紧的）

当然，这些还不能证明时间和空间上界是不正确的。因为要证明这个算法的时间复杂度是 $O(n^3)$的，需要证明的是 $\sum_{i=1}^{\frac{d}{2}} |LA_i \| LA_{d-i}|$ 是 $O(n^3)$的。要证明这个算法的空间复杂度是 $O(n^2)$的，需要证明的是 $\max_{i=1}^{\frac{d}{2}} |LA_i \| LA_{d-i}|$ 是 $O(n^2)$的。因此，要否定这两个上界也必须否定这两个结论。

5．2．2．5．针对 BOI 官方解答时空复杂度的反例

反例是复杂的，不能在一张图上全部展示，因此这一节中，本文将逐层次的

阐述反例的构造。

图 5-3 是反例的大结构框架：“水管+水池”。

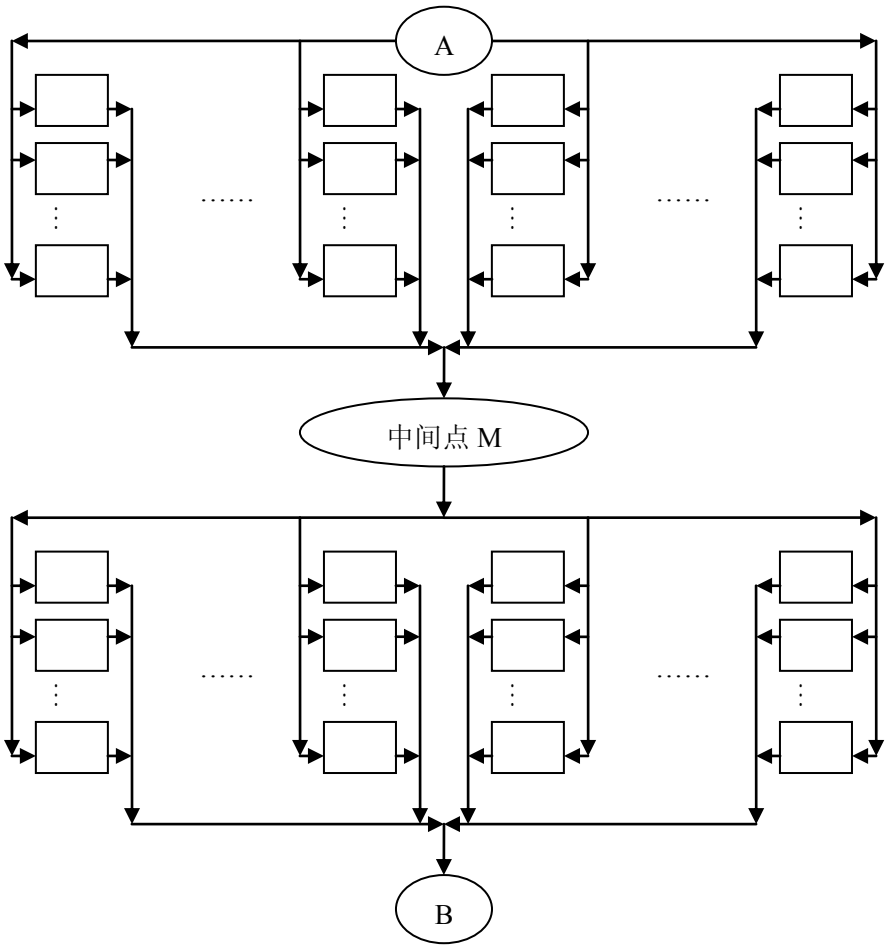

图 5-3

其中带箭头的线表示“进水管”“出水管”。正方形表示“水池”，共有 $O(\sqrt{n})$ 行，$O(\sqrt{n})$ 列。从上图可以看到，每个正方形有一个出口一个入口。小正方形除出入口以外的内部构造如图 5-4：

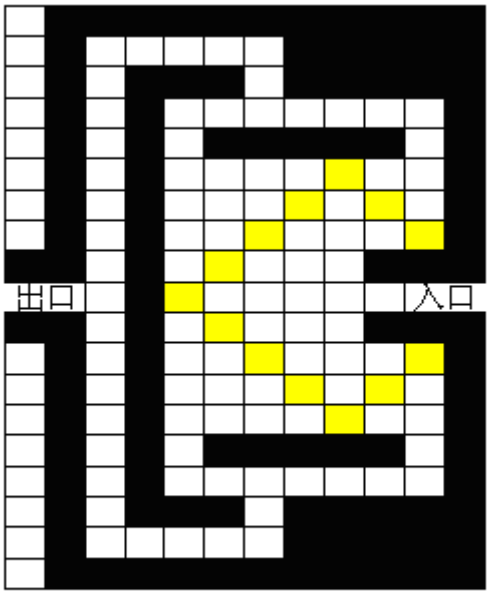

图5-4

同时要求这个方形的尺寸是$O(\sqrt{n})\times O(\sqrt{n})$的（这样的话$O(\sqrt{n})$行$O(\sqrt{n})$列个水池不会超过大正方形尺寸）。注意到，上图中的黄色格是在同一个 LA_i 数组中的，一个水池中这样的黄色格有$O(\sqrt{n})$个。因此，如果能够使得起点 A 到所有入口距离相等的话，那么所有水池中的黄色格都在同一层，那就有$O(\sqrt{n})\cdot O(\sqrt{n})\cdot O(\sqrt{n})=O(n^{1.5})$个格子同一个 LA_i 数组中了。下面图 5-5 的横向入水管的设计使得这一点得以实现（只给出了左半侧，右半侧对称后类似），注意，每条总水管的宽度也是$O(\sqrt{n})$。

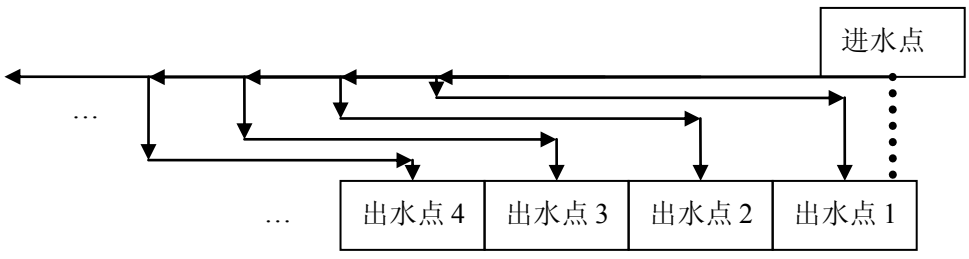

图 5-5

这样一来，进水点到每个出水点的距离相等。纵向进水管也类似构造，便使得从顶点 A 到每个（大图中）上半侧的水池距离相等！顶点 B 到下半侧的水池也是。

这样一来，空间复杂度就需要：$\max_{i=1}^{\frac{d}{2}} |LA_i \| LA_{d-i}| = O(n^{1.5} \cdot n^{1.5}) = O(n^3)$（因为 A 到上半侧黄色格的距离都下相等，也等于 B 到下半侧黄色格的距离）。

而时间复杂度 $\sum_{i=1}^{\frac{d}{2}} |LA_i \| LA_{d-i}| \geq \sum_{i=1}^{\lambda\sqrt{n}} \left(i \cdot \sqrt{n} \cdot \sqrt{n}\right)^2 = O(n^{3.5})$（这只估计了水池中的状态总数，是一个下界）。

这就证明了 BOI 官方解答的复杂度估计是错误的。

5．2．2．6．BOI 2008 Game 进一步分析及优化

尽管 BOI 官方解答的时空复杂度的估计是错误的，但贪心的思想是正确的。下面，本文会沿着这一方向进一步分析。

注意到，不属于任何一个数组 LA_i 的白格子等同于黑格，所以把它们涂黑。这样，对每一个 i 而言，数组 LA_i 中的格子把所有白色区域分成两份，一部分与 A 的距离小于 i，另一部分大于 i。

所以，数组 LA_i 中的格子和黑格以及边界一起，形成了一个封闭图形。而且，与 LA_i 中连续的一段格子相邻的 LA_{i+1} 中格子也是相邻的。所以，对于每个 i,j 而言，使得 NP_A[i,j,k]=N 的 k 是连续的。NP_B 也一样。

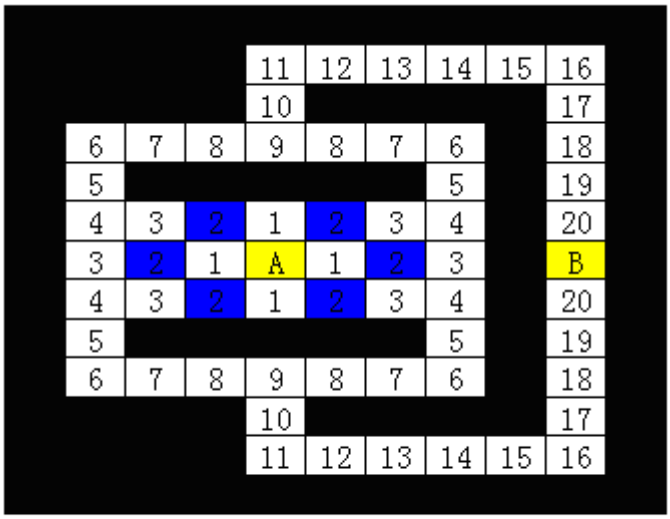

图5-6　LA数组是环型的例子

注意，因为封闭图形是环状的，所以这里所有“连续”的含义都是环状的，即首尾相连的。图 5-6 就是一个例子。

因此，LA_{d-i} 中的格子所形成的环，可以分成两段，一段中的 LA[d-i,k]使得

NP_A[i,j,k]=N，另一段使得 NP_A[i,j,k]=P。

这就是前面提到的状态单调性。像前面第三章第一节的处理方式一样，只需设出 NP 状态的分界处就可以了。这个问题中唯一不同的是，状态是环型的，所以有两个分界点。

用 left[i,j],right[i,j] 表示这两个分界点。设 $NP\_A[i,j,left[i,j]]$ 到 $NP\_A[i,j,right[i,j]]$ 都是 N 状态。

当 left[i,j]<=right[i,j]时表示：当 $k \in [left[i,j], right[i,j]]$ 时 NP_A[i,j,k]都是 N 状态；

当 left[i,j]>right[i,j]时表示当 $k \in [left[i,j], top] \cup [1, right[i,j]]$ 时 NP_A[i,j,k]都是 N 状态。

递推计算 left[i,j]和 right[i,j]的算法是简单的，故总时间复杂度为 $O(n^2)$。（这是因为每个格子最多以 $LA_i[j]$ 的形式出现一次，所以 left[i,j],right[i,j]的总数，不超过各自的总数——n^2）。

5．2．3．小结

BOI 官方解答的时空复杂度的估计的错误，使得可以构造数据使得程序数据越界。但更改程序数组大小后，程序不会超时，这是因为，使得运行时间为 $O(n^{3.5})$ 的构造的时间复杂度的常系数很小，在题目的 $n \le 300$ 限制下，未能体现出很大的劣势。也就是说，这个估计错误的理论价值相比实用价值较大。

另外，尽管 BOI Game 的官方解答的时空复杂度的估计是错误的，但贪心的思想是正确的。正是贪心思想的指引，成就了最终 $O(n^2)$ 的优秀算法。

必须指出，如贪心算法在信息学竞赛中的广泛应用一样，贪心思想解决游戏论问题也是常见的。

六．总结

NP 状态定理和基于它的动态规划是解决游戏有问题的通式方法，它们构建了解决游戏论问题的基本框架。

但对于游戏的分析，以及动态规划的优化手段是因题而异的。尤其是单调性的分析，和对称、贪心等非动态规划的分析相当灵活。

另外，SG 函数作为 NP 状态的增强版，再借助 Nim 和与 Nim 积的运算功能非常强大。但是，使用这些定理往往不是题目的难点，题目的难点一般在于对题目的分析，使得以和复杂的游戏能表达成多个游戏的“和”或“积”的形式。

附录一.“K 倍动态减法游戏”在 k=1 情况下的胜负判断

沿用正文中的符号。

结论：f(m)=lowbit$_2$(m)，其中 lowbit$_2$(m)表示正整数 m 在 2 进制下的最低位，即 $lowbit_2(m)=2^k, 2^k \| m$（$2^k \| m$ 表示恰好整除）。

证明：用归纳法证明。

(1)列表，知结论对 m=1,2,3 或 4 成立。

(2)若结论对 1,2,3…m-1 成立。则对于 m：

注意到，对于任意 $0<n<lowbit_2(m)$，由 2 进制下的退位减法法则知：$lowbit_2(m-n)=lowbit_2(n)<lowbit_2(m)$。

```
  m      1...1000
m-n       ...0010
  n       ...0110
```

2进制下退位减法的图式

而且，由 n>0 知：$lowbit_2(n)\le n$。

所以 $f(m-n)=lowbit_2(m-n)=lowbit_2(n)\le n=kn$。即，$f(m)\ge lowbit_2(m)$。

要证 $f(m-lowbit_2(m))>lowbit_2(m)$ 分两种情况：

1. $m=lowbit_2(m)$ 时，$f(m-lowbit_2(m))=f(0)=+\infty>lowbit_2(m)$

2. $m>lowbit_2(m)$ 时，不难发现，一个数减去了它的 2 进制最低位后，他的二进制最低位。也就是，把 m 的 2 进制下最靠右的 1 改成 0 以后，m 的 2 进制下最靠右的 1 比之前更左边了。

```
        m          1010010
m-lowbit(m)        1010000
```

把m的最右侧的1改成0前后

所以，$f(m-lowbit_2(m))>lowbit_2(m)$ 恒成立。

所以，由 f(m)=min{n|f(m-n)>kn}知，$f(m)=lowbit_2(m)$。

综上：$f(m)=lowbit_2(m)$ 对一切正整数 m 成立。

推论 1：当且仅当起始值 S(>1)为 2 的幂次时，先手必败，否则先手必胜。

这是因为当且仅当 S 为 2 的幂次，$S=f(S)$。

推论 2：当(m,n)为 N 状态时（即 $n\geq lowbit_2(m)$），在 S 上减去 $lowbit_2(m)$ 是必胜策略。

由于不考虑高精度计算的情况下，计算 $lowbit_2(m)$ 只需 O(1)的时间。所以，不考虑高精度计算的情况下，判定胜负的时间复杂度是 O(1)；交互型的游戏算法的时间复杂度为 O(1)-O(1)的。

附录二.“K 倍动态减法游戏”在 k=2 情况下的胜负判断

沿用正文中的符号。且记 F_n 表示费波纳切数列，即 $F_0 = F_1$ =1,对于 n>1, $F_n = F_{n-1} + F_{n-2}$。

引理：对于任意正整数 m，m 可以被唯一的表示成一些费波纳切数的和，且这些费波纳切数任意两个不相等，任意两个不相邻。

即，对于给定的 m，满足下述条件的 k,$i_1, i_2, i_3 \ldots i_k$ 取值唯一：

$$m = \sum_{j=1}^{k} F_{i_j}$$ 且对于任意 j=1,2…k-1 有 $i_j + 2 \le i_{j+1}$。

引理的证明：对 m 归纳：

(1)当 m=1,2,3,4,5，下面表达方式显然唯一：1=1,2=2,3=3,4=1+3,5=5

(2)当 m>6，若结论对 1,2,3…m-1 成立，则：

设 $F_t \le m < F_{t+1}$。显然：$i_k \le t$。

假如 $i_k < t$，则 $\sum_{j=1}^{k-1} F_{i_j} = m - F_{i_k} \ge F_t - F_{t-1} = F_{t-2}$。

但事实上，$\sum_{j=1}^{k-1} F_{i_j} \le F_{i_k-2} + F_{i_k-4} + \ldots \le F_{t-3} + F_{t-5} + F_{t-7} + \ldots$

$= (F_{t-2} - F_{t-4}) + (F_{t-4} - F_{t-6}) + \ldots < F_{t-2}$，矛盾！

所以，i_k 取值唯一。

而另一方面，由归纳假设，$m - F_t$ 可以被唯一的表达成符合要求的和。且根据 $m - F_t < F_{t+!} - F_t = F_{t-1}$，表示 $m - F_t$ 的和的最大项不超过 F_{t-2}，不与 F_t 相邻。也就是说，$i_1, i_2, i_3 \ldots i_{k-1}$ 的合法表达也存在，且唯一！

综上，引理获证。

根据引理，可以引入符号 $lowbit_F(m)$，表示正整数 m 表示成上述和时的最小项。

结论：“k 被动态减法游戏”中，若 k=2，则 $f(m) = lowbit_F(m)$。

证明：对 m 归纳：

(1)列表，知结论对 m=1,2,3,4 或 5 成立。

(2)若结论对 1,2,3…m-1 成立。则对于 m：

注意到，若设 m 可唯一表达为 $m=\sum_{j=1}^{k}F_{i_j}$，则对于任意 $0<n<lowbit_F(m)=F_{i_1}$，有：$m-n=\sum_{j=1}^{k}F_{i_j}-n=(F_{i_1}-n)+\sum_{j=2}^{k}F_{i_j}$。所以 $0<f(m-n)=lowbit_F(m-n)=lowbit_F(F_{i_1}-n)=lowbit_F(lowbit_F(m)-n)$。

又注意到 $lowbit_F(m)-n$ 在按费波纳切数列唯一分解时的最小项，最多与 n 的最小项相差一位。

也就是说，若 $lowbit_F(n)=F_i$，则 $lowbit_F(lowbit_F(m)-n)\le F_{i+1}$。

所以，$lowbit_F(lowbit_F(m)-n)\le F_{i+1}=F_i+F_{i-1}\le 2F_i=2lowbit_F(n)$

所以，$0<f(m-n)\le 2lowbit_F(n)\le 2n$。

另一方面，与 k=1 的情况类似，若 $lowbit_F(m)=F_i$，则 $lowbit_F(m-F_i)\ge F_{i+2}$。而 $F_{i+2}=F_{i+1}+F_i>2F_i$，所以，$lowbit_F(m-F_i)>2F_i$。

故，综上：$f(m)=lowbit_F(m)$。

结论获证。

推论 1：当且仅当起始值 S(>1)为费波纳切数时，先手必败，否则先手必胜。

这是因为当且仅当 S 为费波纳切数，$S=f(S)$。

推论 2：当(m,n)为 N 状态时（即 $n\ge lowbit_F(m)$），在 S 上减去 $lowbit_F(m)$ 是必胜策略。

由于不考虑高精度计算的情况下，计算 $lowbit_F(m)$ 只需 O(logm)的时间。所以，不考虑高精度计算的情况下，判定胜负的时间复杂度是 O(logm)；交互型的游戏算法的时间复杂度为 O(logm)-O(logm)的。

附录三．Nim 游戏的变种

变种一：共有 n 级楼梯，每一级楼梯上有若干颗棋子，双方轮流操作，每次操作可以将某一级楼梯上若干颗棋子一道下一级楼梯，即从第 i 级到第 i-1 级。第 0 级表示平地。不能操作者负，即，使得所有棋子到达平地者胜。

变种二：共有 n 个格子排成一排，编号 0,1,2…n-1。现有若干颗棋子，放在某些格子中。双方轮流操作，每次操作可以将一颗棋子移到比当前标号小的格子中。起始终态以及中间过程中允许两颗或多颗棋子在同一个格子中。无法操作者负。即使得所有棋子移进 0 号格者胜。

变种三：有一个矩形方格棋盘，每行中恰有一黑色棋子和一白色棋子。任意两颗棋子不同格。双方 AB 轮流操作，每次操作可以将一颗棋子在同行中移动，但 A 只能以白棋，B 只能移黑棋，而且，移动的过程不能越过其他棋子。无法操作者负。

变种四：有 k 堆石子，各包含 $x_1,x_2…x_k$ 颗石子，满足 $x_1 \le x_2 \le ... \le x_n$，且在整个游戏过程中必须维护这一性质。双方玩家轮流操作，每次操作选择其中非空的一堆，取走其中的若干颗石子（1 颗，2 颗…甚至整堆），保持上述性质不变（堆与堆之间不能交换位置），取走最后一颗石子的玩家获胜。

变种五：有一颗有根树。双方玩家轮流操作，每次操作，删除树种的一条边。而且，每次操作后，原树会被分成不联通的两部分，与根不相连的部分的节点和边也被全部一起删除。删去最后一条边的玩家胜。

附录四．SG 定理的证明

SG 定理：设 $G(V,E)=G_1+G_2+..+G_n$，$x=(x_1,x_2...x_n)\in V$，则 $SG(x)=SG(x_1)\oplus SG(x_2)\oplus...\oplus SG(x_n)$。

对于 n=2 的情况：对照⊕的运算法则表，不难发现：

$x\oplus y=mex\{a\oplus x,b\oplus y\}$其中，a<y,b<x。

所以：$SG(x)=SG(x_1)\oplus SG(x_2)$。

进一步，对于一般的 n：$SG(x)=SG\left(\sum_{i=1}^{n}x_i\right)=SG\left(\sum_{i=1}^{n-1}x_i\right)\oplus SG(x_n)=$

$SG\left(\sum_{i=1}^{n-2}x_i\right)\oplus SG(x_{n-1})\oplus SG(x_n)=...=SG(x_1)\oplus SG(x_2)\oplus...\oplus SG(x_n)$。定理获证。

参考文献

Theodore L. Turocy, Texas A&M University ***Game Theory***

Erik D. Demaine, Robert A. Hearn, ***Algorithmic Combinatorial Game Theory***

Thomas S. Ferguson ***GAME THEORY***

E. R. Berlekamp, J. H. Conway and R. K. Guy ***WinningWays for your mathematical plays,***

刘汝佳、黄亮 *算法艺术与信息学竞赛*