浅谈如何解决不平等博弈问题

广东省中山市第一中学 方展鹏

摘要

本文主要介绍如何解决一类游戏双方可选决策不相同的博弈问题。全文分为四个部分。

第一部分：引言，通过寻找博弈问题的区别，提出需要解决的问题。

第二部分：另辟蹊径，介绍如何利用 Surreal Number 解决一类不平等的组合游戏（Partisan Combinatorial Games）。

第三部分：回到起点，利用局面分析的办法，通过迭代、动态规划等方法解决不平等博弈问题。

第四部分：总结全文。

关键字

不平等博弈问题 Surreal Number 局面分析 递推 迭代

目录

正文

一、引言

在信息学竞赛中，博弈问题十分常见，下面就是一个例子：

引例：Green Hackenbush（经典问题）

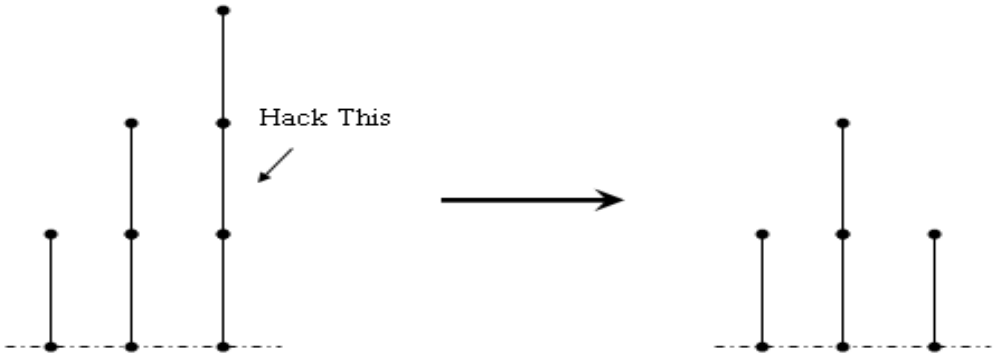
给出 n 棵竹子，高度分别为 $a_1, a_2 \ldots a_n$，玩家 L 和 R 打算在这些竹子上面玩砍竹子游戏，规则如下：

1、两人轮流操作，玩家 L 先手；

2、对于每次操作，先选定一棵高度不为 0 的竹子，然后砍掉该竹子的某一段，并且将与竹子底部不相连的部分也去掉；

3、最先无法进行操作的人输。

假设玩家 L 和 R 都采取最优策略，请问对于给出的局面谁会获胜。



不难看出，一棵高度为 n 的竹子，经过一次操作之后可以变成一棵高度为 0 到 n-1 的竹子。因此，一棵高度为 n 的竹子就等价于一个有 n 颗石子的石子堆（nim pile）。所以，上面的问题就转化为有 n 堆石子，石子数分别为 $a_1, a_2 \ldots a_n$，两个玩家轮流操作，每次操作为选取一堆石子并从中拿走不少于 1 个，最先无法进行操作的人输，问先手必胜还是后手必胜。对于这个问题，根据 The Sprague-Grundy Theorem[1]，我们可以轻松地设计出一个时间复杂度为 O(n)的算法[2]。

我们发现在 Green Hackenbush 中，对于任意局面，玩家 L 和玩家 R 的可选决策都是相同的，也就是如果玩家 L 可以对第 i 棵竹子的第 j 段操作，则玩家 R 也可以。但是，如果不相同的话会怎么样？我们不妨在游戏规则处再多加一条：竹子的每一段都被标上了 L 或者 R，玩家 L 只能砍被标上 L 的段，玩家 R 只能

[1]设 g_i 为游戏 G_i 的 SG 函数，则 $G=G_1+G_2+\ldots G_n$ 的 SG 函数 $g(x_1,x_2,\ldots x_n) = g_1(x_1)$ xor $g_2(x_2)$ … xor $g_n(x_n)$。

[2]详见 2002 年集训队论文《由感性认识到理性认识——透析一类搏弈游戏的解答过程》 张一飞

砍被标上 R 的段。加上这条限制以后，玩家 L 和玩家 R 的可选决策就不相同了，同时我们还发现 The Sprague-Grundy Theorem 在这类问题上也不再成立。

本文所要探讨的正是如何解决这类两个玩家的可选决策集合不相同的博弈问题，也称之为不平等博弈问题（Partizan Games）。

二、另辟蹊径

本节将介绍一个新的工具 Surreal Number，并且通过一道例题介绍如何利用这一工具来分析和解决一类不平等的组合游戏（Partisan Combinatorial Games）。

2.1 Surreal Number 介绍

在数学中，surreal number 是一个包含无穷大数、无穷小数还有实数的域。下面将介绍 surreal number 的定义以及相关性质。由于本文只是将 surreal number 作为一个分析博弈问题的工具引入，所以 surreal number 的一些与博弈问题关系不大的性质将不会在本文被提及，有兴趣的读者请参看参考文献[1]。

2.1.1 Surreal Number 的构造

定义 1 一个 surreal number 由两个集合组成。我们称这两个集合为“左集合”与“右集合”。通常情况下，我们会将 surreal number 写作{ L | R }，其中 L 表示左集合，R 表示右集合。这两个集合中的元素也为 surreal number，且右集合中不存在一个元素 x 使得左集合中存在一个元素 y 满足 $x \le y$。

定义 2 对于 surreal number x = { X_L | X_R }和 y = { Y_L | Y_R }，我们称 $x \le y$ 当且仅当不存在 $x_L \in X_L$ 使得 $y \le x_L$ 以及不存在 $y_R \in Y_R$ 使得 $y_R \le x$。为了表述方便，我们可以将 $y \le x$ 写作 $x \ge y$，将 $x \le y \wedge \neg(y \le x)$ 写作 x<y，将 y<x 写作 x>y，将 $x \le y \wedge y \le x$ 写作 x=y。

不难看出上面的两个定义都是基于递归的形式给出的，通过上面的两个定义，我们可以尝试构造 surreal number。

由定义 1 可知，一个 surreal number 由左集合与右集合组成，且这两个集合

的元素也必须为 surreal number。但是我们目前连一个 surreal number 也不知道，因此只能令 L 为空集，R 为空集，得出{Φ | Φ}，通常情况下会简写成{ | }。根据定义 1，我们可以证明{ | }是一个合法的 surreal number。然后，我们令{ | } = 0。同时我们称 0 是在第 0 天出生。

构造出 0 之后，我们尝试利用 0 构造出新的 surreal number，得出{ 0 | }，{ | 0 }以及{0 | 0}。根据定义 2 可知，$0 \le 0$，因此{0 | 0}并不满足定义 1，所以{0 | 0}不是一个 surreal number。

利用定义 2，我们可以证明{ | 0 } < 0 < { 0 | }，因此我们令{ | 0 } = -1，{ 0 | } = 1，同时我们称 1 和-1 是在第 1 天出生的。

利用 0,1,-1 作为左集合与右集合的元素，我们可以构造出 17 个合法的 surreal number，我们称这 17 个 surreal number 是在第 2 天出生的。其中利用定义 2，我们可以证明 1 < { 1 | }和-1 > { | -1}，因此我们令{ 1 | } = 2，{ | -1} = -2。

另外，我们发现 0 < { 0 | 1 } < 1，由于在稍后介绍的加法运算中会有{ 0 | 1 } + { 0 | 1 } = 1，因此我们令{ 0 | 1 } = $^1/_2$。同理我们可以得出{ -1 | 0 } = -$^1/_2$。

因此现在我们有 7 个 surreal number：-2, -1, -$^1/_2$, 0, $^1/_2$, 1, 2。虽然在第 2 天出生的合法的 surreal number 还有 13 个，但是可以证明它们的值是上面这 7 个数的其中一个。例如利用定义 2，我们可以证明{ -1 | 1} = 0。

如此类推，在第 3 天我们可以构造出 $^1/_4$ = { 0 | $^1/_2$}，$^3/_4$ = { $^1/_2$ | 1 }，$^3/_2$ = { 1 | 2 }，3 = { 2 | }以及对应的负数的情况。一直这样下去，我们可以构造出所有形如 $j/2^k$ 的有理数。具体而言，我们可以定义如下函数来建立部分有理数与 surreal number 的对应关系，我们称这个函数为达利函数：

$$\delta(x)=\begin{cases}\{|\}, x=0\\ \{\delta(x-1)\,|\}, x>0\wedge x\in Z\\ \{|\,\delta(x+1)\}, x<0\wedge x\in Z\\ \{\delta(\frac{j-1}{2^k})\,|\,\delta(\frac{j+1}{2^k})\}, x=\frac{j}{2^k}\wedge j,k\in Z\wedge k>0\end{cases}$$

实际上我们还可以构造出所有的实数、无穷大数和无穷小数，但是这与本文的主题关系不大，因此这里就不给出具体的构造方法了，有兴趣的读者可以参看参考文献[1]。

2.1.2 Surreal Number 的一些基本定理

定理 1 对于一个 surreal number x = { L | R }，x 大于 L 中的任意一个元素且小于 R 中的任意一个元素。

定理 2 对于一个 surreal number x = { L | R }，若集合 L 中有最大元素 l_{max}，那么{ l_{max} | R } = x；类似地，若集合 R 中有最小元素 r_{min}，那么{ L | r_{min} } = x 。

举个简单的例子：{ 1, 2, 3 | 4, 5} = { 3 | 4, 5} = { 1, 2, 3 | 4} = {3 | 4 }。

看到这里，或许会有读者提出能否将 surreal number 的定义改为“左集合与右集合最多只能含有一个元素”，实际上这是不行的。注意到定理 2 要求集合 L 中有最大元素，集合 R 中有最小元素，然而当集合 L 与集合 R 的大小为无穷大时，就可能出现不存在最大元素与最小元素的情况，因此 surreal number 的定义不能修改。

定理 3 设 a 到 b 之间最早出生的 surreal number 是在第 i 天出生，那么在 a 到 b 之间且在第 i 天出生的 surreal number 只有一个。

定理 4 如果 a < x < b，且 x 是 a 到 b 之间最早出生的 surreal number，那么{ a | b } = x 。

2.1.3 Surreal Number 的运算法则

下面我们将给出 surreal number 的加法运算法则与减法运算法则，这两个法则也是基于递归的形式给出的。

定义 3 对于 surreal number x = { X_L | X_R }和 y = { Y_L | Y_R }，它们的加法运算被定义为：

$$x+y=\{X_L \mid X_R\}+\{Y_L \mid Y_R\}=\{X_L+y, x+Y_L \mid X_R+y, x+Y_R\},$$

其中对于集合 X 与 surreal number y， $X+y=\{x+y : x\in X\}$。

注意到上面关于加法的定义是基于递归给出的，因此我们还要增加一个结束的条件： $\Phi+n=\Phi$。

定义 4 对于 surreal number x = { X_L | X_R }，x 的相反数为：

$-x=-\{X_L \mid X_R\}=\{-X_R \mid -X_L\}$，其中对于集合 X， $-X=\{-x : x\in X\}$。

由于关于相反数的定义也是基于递归给出的，因此我们还要增加一个结束的

条件：$-0 = -\{|\} = \{|\} = 0$。

定义 5 对于 surreal number x 和 y，x - y = x + (-y)。

通过上面的三个定义，我们可以得出如下定理：

定理 5 对于 surreal number $x = \{ X_L | X_R \}$和 $y = \{ Y_L | Y_R \}$，$x + y = \{ X_L + y, x + Y_L | X_R + y, x + Y_R \}$也是一个合法的 surreal number。

定理 6 surreal number 的加法满足交换率，即 x + y = y + x。

定理 7 surreal number 的加法满足结合率，即(x + y) + z = x + (y + z)。

2.2 Surreal Number 与组合游戏

2.2.1 组合游戏的定义与表示

在介绍 surreal number 在组合游戏中的应用之前，我们先来看看这里所讲的组合游戏的定义：

- 游戏有 2 名参与者，通常称为玩家 L 和玩家 R。
- 游戏过程中的任意时刻有确定的状态。
- 参与者操作时将游戏从当前状态转移到另一状态，规则规定了在任意一个状态时，参与者可以到达的状态集合。
- 参与者轮流进行操作。
- 在游戏处于某状态，当前参与者不能进行操作时，游戏结束。本节只讨论最先不能进行操作者输的情况。
- 无论参与者做出怎样的操作，游戏总会在有限步数之内结束(没有平局)。
- 参与者拥有游戏本身，和游戏过程的所有信息，比如规则、自己和对手之前的操作。

由上面的定义可以看出，我们这里所讲的游戏是由状态组成的。对于一个游戏，如果它当前处于状态 P，玩家 L 可以转移到的状态的集合为 P^L，玩家 R 可以转移到的状态的集合为 P^R，那么我们把这个游戏写作 $P = \{ P^L | P^R \}$。

举个例子，例如当前处于状态 P，玩家 L 可以到达的状态为 A、B、C，玩家 R 可以到达的状态为 D，那么我们可以这样描述当前的游戏：P = { A, B, C | D }。

看到这里，读者可能会觉得这里的游戏与 surreal number 非常相似。事实上

也的确如此，因为所有的 surreal number 都是游戏，但是并非所有的游戏都是 surreal number，因为对于游戏 $P = \{ P^L | P^R \}$，并没有要求集合 P^L 中的任意元素都要小于集合 P^R 中的任意元素。因此我们下面讨论的游戏，都是可以表示为 surreal number 的游戏。

2.2.2 Surreal Number 在组合游戏中的应用

对于一个游戏 G，如果玩家 L 和玩家 R 都采用最优策略进行游戏，且游戏 G 等价于 surreal number x，那么有如下结论：

- 如果 $x > 0$，那么无论先手还是后手，玩家 L 都会获胜。
- 如果 $x < 0$，那么无论先手还是后手，玩家 R 都会获胜。
- 如果 $x = 0$，那么谁后手谁获胜。

上面的结论可以用类似于数学归纳法的思想来证明：

当 $x > 0$ 时，如果是玩家 L 先手，从 x 是 surreal number 可知，玩家 L 总可以将 x 转移到一个大于等于 0 的状态；如果是玩家 R 先手，那么玩家 R 只能将 x 转移到一个大于 0 的状态，因此任意时刻玩家 L 的可选决策集合都不为空集，所以这种情况无论先手还是后手，玩家 L 都会获胜。

当 $x < 0$ 时，玩家 L 只能将 x 转移到一个小于 0 的状态，而玩家 R 总可以将 x 转移到一个小于等于 0 的状态，因此任意时刻玩家 R 的可选决策集合都不为空集，所以无论先手还是后手，玩家 R 都会获胜。

当 $x = 0$ 时，如果是玩家 L 先手，那么他只能将 x 转移到一个小于 0 的状态，这样玩家 R 会获胜；如果是玩家 R 先手，那么他只能将 x 转移到一个大于 0 的状态，这样玩家 L 会获胜。所以，对于 $x = 0$ 的情况，谁后手谁获胜。

很多时候，一个游戏 G 会被分解成 n 个不相交的子游戏 $G_1, G_2, \ldots G_n$，对 G 的每次操作等价于从 n 个子游戏中选取一个来进行操作。这种情况，我们称游戏 G 是游戏 $G_1, G_2, \ldots G_n$ 的和，写作 $G = G_1 + G_2 + \ldots + G_n$。有 n 堆石子的 Nim 游戏就是一个经典的例子。

对于这类可分解的游戏，surreal number 能够对其较好地进行分析，因为有如下定理：

定理 8 如果游戏 G 等价于 surreal number x，游戏 H 等价于 surreal number y，那么游戏 G + H 等价于 surreal number x + y。

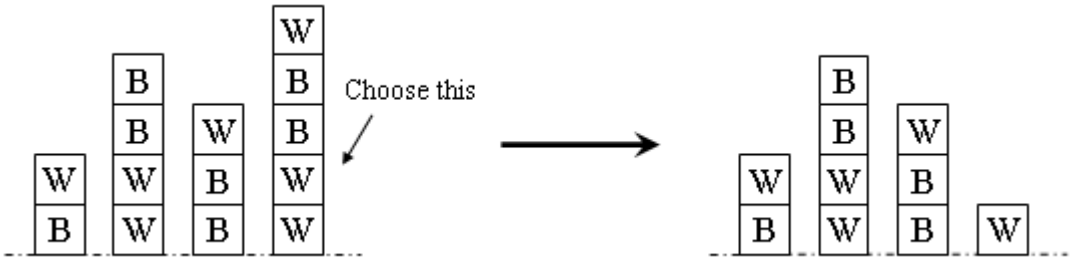
上述定理通过 surreal number 加法运算的定义来证明。通过上面的定理我们可以解决很多 The Sprague-Grundy Theorem 无法解决的不平等博弈问题，下面我们通过一道例题来介绍如何利用 surreal number 以及上述定理来解决一类不平等的组合游戏。

2.3 例一：Procrastination[3]

题目描述

有一个叫 Procrastination 的游戏，规则如下：

- 游戏一开始有四座由正方体叠成的塔，且所有的正方体要么是黑色，要么白色。
- 有两个玩家 L 和 R，游戏开始后，两个玩家轮流进行操作。
- 每次操作，玩家要选定一个正方体，然后拿走该正方体以及位于该正方体上面的所有正方体，并且规定玩家 L 只能选定白色的正方体，玩家 R 只能选定黑色的正方体。
- 最先不能进行操作的人输。



对于一个初始局面，如果玩家 L 无论先手还是后手都能获胜，那么我们称这是一个 W-configuration。另外我们定义子局面 C 表示一个由三座塔组成的局面。一个完整的游戏局面可以由一个子局面 C 以及一座塔 T 构成，写作(C, T)。对于两个子局面 C_1 和 C_2，我们称 C_1 不差于 C_2 当且仅当对于任意的一座塔 T，在(C_2, T)为 W-configuration 时(C_1, T)也为 W-configuration。

[3] MIT Programming Contest 2005, pku2931

给出两个子局面 C_1 和 C_2，问是否 C_1 不差于 C_2。

数据规模：对于每座给出的塔，它的高度 n 满足 $0 \le n \le 50$。

分析

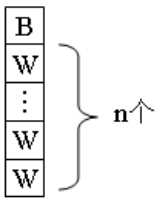
我们不妨先从简单的情况出发，考虑只包含一座塔 T 的局面 G：

如果塔 T 一个正方体也没有，那么无论玩家 L 还是玩家 R 都没有任何可选决策，因此 G = { | }。由 2.1 节的定义可知，G = 0。

如果塔 T 由一个白色正方体叠成，那么玩家 L 有一个转移到状态 0 的决策（选定这个白色正方体进行操作），因此 G = { 0 | } = 1。

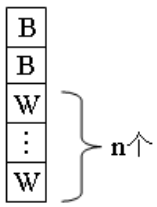
如果塔 T 由两个白色正方体叠成，那么玩家 L 有一个转移到状态 1 的决策（选定上面的正方体）以及一个转移到状态 0 的决策（选定下面的正方体），因此 G = { 0 1 | } = 2。

通过数学归纳法，我们可以得出如果塔 T 由 n 个白色正方体叠成，那么 G = n。同理我们可以得出如果塔 T 由 n 个黑色正方体叠成，有 G = -n。



(1)

下面我们考虑图(1)的情况，即塔 T 由 n(n > 0)个白色正方体加一个黑色正方体叠成。容易得出此时 $G = \{ 0, 1, \ldots n-1 \mid n \} = \{ n-1 \mid n \} = n - 1/2$。



(2)

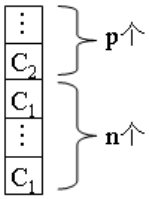
如果塔 T 由 n(n > 0)个白色正方体加上两个黑色正方体叠成，如图(2)所示，则有 $G = \{n-1 \mid n-1/2\} = n - 1/2 - 1/4$。如果在图(2)的基础上在塔 T 的顶端放上一个黑色正方体，可得 $G = n - 1/2 - 1/4 - 1/8$；若在图(2)的基础上在塔 T 的顶端放上一个白色正方体，可得 $G = n - 1/2 - 1/4 + 1/8$。

如此类推，对于任意一个只包含一座塔 T 的局面 G，我们可以用如下算法计算与 G 对应的 surreal number x（为了方便叙述，用伪代码来表示）：

```
SurrealNumber(T)   //T[i]表示塔T从下往上数第i个正方体的颜色
x ← 0
i ← 1
n ← 塔 T 的大小
while i ≤ n and T[i] = T[1]
    if T[i] = 白色 then x ← x + 1 else x ← x - 1
    i ← i + 1
k ← 2
while i ≤ n
    if T[i] = 白色 then x ← x + 1/k else x ← x - 1/k
    i ← i + 1
    k ← k * 2
return x
```

可以看出上述算法分两步进行计算，第一步是计算塔的最下方颜色相同的一“段”正方体，第二步则是计算在该“段”正方体上方的正方体。容易得出上述算法的时间复杂度为 O(n)。

算法第二步的正确性证明如下：



我们可以用数学归纳法证明。考察如右图所示的塔 T，T 最下面的 n 个正方体都是同一种颜色 C_1，自下往上数第 n+1 个正方体的颜色为不同于 C_1 的 C_2，除去最下面的 n 个正方体，还有 p 个正方体：

(1)当 p = 1 时，设塔 T[1..n]对应的 surreal number 为 v。如果塔 T 最上面的正方体为黑色正方体，则 G = { 0, 1, … n – 1 | n } = { n – 1 | n } = n - 1/2 = v - 1/2。如果塔 T 最上面的正方体为白色正方体，则 G = { -n | 0, -1, …-n + 1 } = - n + 1/2 = v + 1/2。因此当 p = 1 时，算法是正确的。

(2)当 $p = m(m > 1 \wedge m \in N)$ 时，假设算法对 p 等于 1 到 m-1 的情况都正确，下面我们来证明 p = m 的时候算法也是正确的。我们先将在塔 T 最上面的 m + 1 个正方体从上往下编号为 m, m – 1, m – 2, … 0，然后分两种情况进行讨论：

①若编号为 m 的正方体为黑色正方体，那么我们自上往下找到第一个白色正方体。设其编号为 k，塔 T[1..n + k]对应的 surreal number 为 v，塔 T[1..n + m-1]对应的 surreal number 为 u。由定理 2 可得：G = $\{v - \frac{1}{2^k} \mid v - \frac{1}{2^{k+1}} - \frac{1}{2^{k+2}} \ldots - \frac{1}{2^{m-1}}\}$

$= \{v-\frac{1}{2^k} \mid v-\frac{1}{2^k}+\frac{1}{2^{m-1}}\} = \{\frac{v*2^m-2^{m-k}}{2^m} \mid \frac{v*2^m-2^{m-k}+2}{2^m}\}$。由达利函数可知 G

$= \frac{v*2^m-2^{m-k}+1}{2^m} = v-\frac{1}{2^k}+\frac{1}{2^{m-1}}-\frac{1}{2^m} = u-\frac{1}{2^m}$。

②若编号为 m 的正方体为白色正方体，那么我们自上往下找到第一个黑色正方体。设其编号为 k，塔 T[1..n + k]对应的 surreal number 为 v，塔 T[1..n + m-1]对应的 surreal number 为 u。由定理 2 可得：G = $\{v+\frac{1}{2^{k+1}}+\frac{1}{2^{k+2}}...+\frac{1}{2^{m-1}} \mid v+\frac{1}{2^k}\}$

$= \{v+\frac{1}{2^k}-\frac{1}{2^{m-1}} \mid v+\frac{1}{2^k}\} = \{\frac{v*2^m+2^{m-k}-2}{2^m} \mid \frac{v*2^m+2^{m-k}}{2^m}\}$。由达利函数可知 G

$= \frac{v*2^m+2^{m-k}-1}{2^m} = v+\frac{1}{2^k}-\frac{1}{2^{m-1}}+\frac{1}{2^m} = u+\frac{1}{2^m}$。

由①②可知，当 p = m 时算法也是正确的。因此算法对于 p 为任意正整数的情况都是正确的。证毕。

下面我们来考虑初始局面 G 有 n 座塔的情况。不妨设局面 G 由 n 座塔 T_1, T_2, … T_n 组成，且与 T_1, T_2, … T_n 相对应的 surreal number 为 x_1, x_2, … x_n。由 2.2 节的定理 8 可知，与 G 相对应的 surreal number $x = x_1 + x_2 \ldots + x_n$。

题目要求我们判断子局面 C_1 是否不差于子局面 C_2，也就是要求我们判断是否对于任意的塔 H，当 C_2 + H 大于 0 时，C_1 + H 也大于 0，实际上这就等价于要求我们判断 C_1 是否大于等于 C_2。

因此，我们只需分别计算 C_1、C_2 对应的 surreal number 的大小（即先计算子局面内所有塔对应的 surreal number 的大小再相加），然后再比较大小即可。算法的时间复杂度为 O(n)，是一个非常优秀的算法。

小结

从上面的例子可以看出，利用 surreal number 来分析不平等组合游戏，不仅思路清晰，而且程序的实现也相当简洁，以上面的 procrastination 为例，实现出来的程序大小连 1kb 也不到。因此，对于这类问题的分析，surreal number 是我们的首要选择。

三、回到起点

从上一节的介绍可以看出，surreal number 对于一类特殊的不平等博弈问题，特别是可分解的游戏，具有巨大的优势。但是我们也发现 surreal number 的适用范围较为狭窄，对于一些一般的博弈游戏，例如一些我们平时常见的棋类游戏，显得作用不大。因此对于很多的不平等博弈问题，我们只能回到起点，利用局面分析的办法，通过合理的状态设计以及动态规划、迭代等方法解决该类问题。

3.1 例二：Procrastination[4]

题目描述

见本文 2.3 节。

分析

从 2.3 节的介绍可知，通过 surreal number 可以轻松解决此题，但是除了 surreal number 就没有别的方法了吗？下面我们尝试用局面分析的办法来解这道题目。

我们先来看 W-configuration 的定义：一个局面是 W-configuration，当且仅当玩家 L 无论先手还是后手都能获胜。既涉及到先手又涉及到后手，这个定义给我们的感觉是比较复杂，不好分析，能不能简化一下呢？朝着这方面想，不难得出如下结论：

命题 3.1.1 对于一个局面 G，如果玩家 L 先手必胜，那么玩家 L 后手也是必胜。

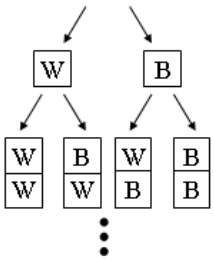
证明：对于一个局面 G，若玩家 L 先手必胜，那么 G 中一定存在一个白色正方体 K，使得玩家 L 对 K 进行操作后将 G 转移到一个玩家 R 先手必败的局面 G_1。因此，对于局面 G，当玩家 R 先手时，如果在整个游戏过程中他都没有对在 K 上方的正方体进行操作，那么他等价于以先手的身份开始局面 G_1，因此玩家 R 必败；若玩家 R 在游戏过程中有对在 K 上方的正方体进行操作，那么玩家 L 可以在玩家 R 进行操作后立刻对 K 进行操作，那么玩家 R 依然等价于以先手的身份开始局面 G_1，因此玩家 R 依然必败。证毕。

[4] MIT Programming Contest 2005, pku2931

有了上面的结论，我们只需关注局面能否使玩家 L 先手必胜就可以了。下面我们来看题目中关于“子局面 C_1 不差于子局面 C_2”的描述，这个描述涉及到任意的塔 T，如果我们可以枚举所有的塔 T，然后再进行判断，那么问题就迎刃而解了。上面的思路听上去或许比较可笑，因为塔有无限个，但是我们可以尝试用枚举来打开突破口。

为了方便叙述，定义函数 g(X)，其中 X 为一个游戏局面，若玩家 L 在局面 X 中先手必胜，则 g(X) = 1，否则 g(X) = 0。定义函数 add(T_1, T_2)，其中 T_1 和 T_2 均为一座塔，函数返回的是在 T_1 上面叠上 T_2 构成的塔 T。定义 T[i]表示塔 T 从下往上数的第 i 个正方体，其中 W 表示白色正方体，B 表示黑色正方体。定义命题 P 为子局面 C_1 不差于子局面 C_2。

现在，考虑如何枚举所有的塔 T，一种比较容易想到的方法就是一开始令 T 一个正方体也没有，然后不断在 T 的上面叠上白色或黑色的正方体，用一个类似于二叉树的形式构造所有的塔，如右图所示。



假设当前枚举到一座塔 T，且有 g((C1, T)) = g((C2, T))，然后我们考虑在塔 T 的上面叠上一个正方体，不难得出如下结论：

命题 3.1.2 若 g((C, T)) = 1，那么对于所有的塔 T_1 满足 $T_1[1]$ = W，都有 g((C, add(T, T_1))) = 1。

证明：不妨设 T_2 = add(T, T_1)，塔 T 的大小为 n，塔 T_2 的大小为 m。若在整个游戏过程中，玩家 R 都不对在正方体 $T_2[n]$上方的正方体进行操作，则局面(C, T_2)等价于局面(C, T)，那么由 g((C, T)) = 1 可知，玩家 L 先手必胜。若玩家 R 对在正方体 $T_2[n]$上方的正方体进行操作，那么玩家 L 可以在玩家 R 进行操作后立刻对 $T_2[n+1]$进行操作，因此初始的游戏局面还是等价于(C, T)，玩家 L 先手必胜。所以 g((C, add(T, T_1))) = 1。证毕。

同理我们可以得出如下结论：

命题 3.1.3 若 g((C, T)) = 0，那么对于所有的塔 T_1 满足 $T_1[1]$ = B，都有 g((C, add(T, T_1))) = 0。

回到我们枚举的想法，如果我们当前枚举到一座塔 T，满足 g((C1, T)) = g((C2,

T)) = 1，那么由命题 3.1.2 可知，我们接下来只需要枚举 T 上面叠一个黑色正方体的情况；如果我们当前枚举到的塔 T 满足 g((C2, T)) = g((C1, T)) = 0，那么由命题 3.1.3 可知，我们接下来只需要枚举 T 上面叠一个白色正方体的情况。这样我们就减少了很多不必要的枚举，将枚举量由幂级别减少到了线性级别。

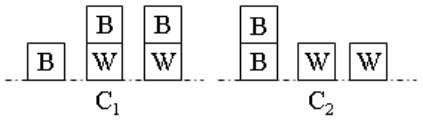
如果我们当前枚举到一座塔 T，满足 g((C1, T)) = 1, g((C2, T)) = 0，那么我们可以得出如下结论：

命题 3.1.4 若 g((C1, T)) = 1, g((C2, T)) = 0，那么对于所有的塔 T1 满足 g((C2, add(T, T1))) = 1 都有 g((C1, add(T, T1))) = 1。

证明：因为 g((C2, T)) = 0，所以如果塔 T1 满足 g((C2, add(T, T1))) = 1，由命题 3.1.3 可知，T1[1] = W，又因为 g((C1, T)) = 1，因此由命题 3.1.2 可得 g((C1, add(T, T1))) = 1。证毕。

因此，如果在我们枚举的过程中，遇到一座塔 T 满足 g((C1, T)) = 1, g((C2, T)) = 0，那么由命题 3.1.4 可知，对于任意的塔 T 满足 g((C2, T)) = 1 都有 g((C1, T)) = 1，这样就可以得出命题 P 为真；同理如果枚举的过程中遇到一座塔 T 满足 g((C1, T)) = 0, g((C2, T)) = 1，那么我们可以得出命题 P 为假。

通过以上的讨论，我们已经基本定下了解题的框架，下面还需要解决一些细节上的问题。首先就是枚举什么时候停止的问题，因为我们不难发现存在一些子局面 C1、C2 满足对于任意的塔 T 都有 g((C1, T)) = g((C2, T))，例如 C1 等于 C2 的时候或者如右图的情况。因此考虑枚举什么时候停止的问题是有必要的。由于给出的子局面包含的塔的大小都不超过 50，因此我们考虑是否当被枚举到的塔 T 的高度达到一定程度后就没有再枚举的必要。朝着这方面想，可以得出如下结论：



命题 3.1.5 设子局面 C 包含的三座塔的高度为 n1, n2, n3，则 g((C, T))只与 T 最下面的 n1 + n2 + n3 + 1 个正方体有关。

由于给出的子局面包含的塔的高度都不超过 50，因此由命题 3.1.5 可知，当我们枚举到塔 T 的高度大于 151 后，若 g((C1, T))依然等于 g((C2, T))，那么我们可以得出对于任意的塔 T，都有 g((C1, T)) = g((C2, T))，因此命题 P 为真。

接下来要考虑的就是函数 g 的实现，对于一个由四座高度分别为 n_1, n_2, n_3, n_4 的塔组成的局面 C，我们可以定义状态 d[i, j, k, l, cur]表示组成局面 C 的四座塔的高度减为 i, j, k, l，且轮到玩家 cur 进行操作的状态是必胜还是必败。然后我们可以利用动态规划的方法，自底向上地计算每个状态。对于一个状态 S，我们用 O(n)的代价来枚举状态 S 可以到达的所有状态 S'（也就是枚举选定哪个正方体进行操作），然后根据 S'的情况来确定状态 S 是必胜还是必败[5]。由于状态的数目为 n^4，计算每个状态的代价为 O(n)，因此计算一个新的局面的代价为 $O(n^5)$。

显然，上述计算方法的时间复杂度我们是无法接受的，因此我们能否对其进行改进？答案是肯定的，通过观察，我们可以得出如下结论：

命题 3.1.6 对于一个局面 G，如果选定正方体 k 进行操作对于当前玩家是一个必胜的决策，那么选定在 k 上方的任意与 k 同色的正方体进行操作对于当前玩家都是必胜决策。

证明：不妨设当前玩家为 L 且选定正方体 k 进行操作是一个必胜决策，如果我们选定在 k 上方的白色正方体 k'进行操作，那么在接下来的游戏过程中，如果玩家 R 不对在 k 上方的黑色正方体进行操作，那么等价于玩家 L 一开始就选定了 k 进行操作，因此玩家 L 必胜；如果玩家 R 对在 k 上方的黑色正方体进行操作，则玩家 L 可以在玩家 R 进行操作后立刻对 k 进行操作，那么还是等价于玩家 L 一开始就选定了 k 进行操作，所以玩家 L 还是必胜。因此选定 k'进行操作也是一个必胜决策。证毕。

由命题 3.1.6 可知，我们在计算状态的时候，并不需要枚举所有的决策，而只需要考虑在每座塔最上方的白色（或黑色）正方体。这样计算一个状态的代价就降到了 O(1)，计算一个局面的代价也降到了 $O(n^4)$。

具体实现的时候，在枚举的过程中，由于新的塔是在原来的塔的基础上添加一个正方体得来，因此对于一座新的塔，我们没有必要将所有的状态都重新计算一遍，而只需要计算新增加的 n^3 个状态，又因为被枚举的塔的最大高度也是 n 阶的，所以算法的时间复杂度为 $O(n^4)$。由于 $n \le 50$，因此上述算法已经可以通过本题。

[5] 详见 2002 年集训队论文《浅析解 “对策问题” 的两种思路——从“取石子”问题谈起》骆骥

小结

回顾整个解题过程，我们只是分析了一些特殊的局面以及用到了动态规划来计算一个局面的胜负状态，并没有用到任何高深的东西，因此可以说这个方法在知识上的要求是比较低的。然而与 2.3 节介绍的 surreal number 的解法相比，这个方法不仅要分析较多的情况，而且实现起来也没有前者简洁清晰。但是当我们面对一些新的博弈问题，在没有现成的工具可以用来帮助分析的情况下，回到起点，从基本局面开始分析的办法往往可以帮我们打开解题的突破口。因此上面的解题思路还是很有掌握的必要。

3.2 例三：The Easy Chase[6]

题目描述

玩家 L 与玩家 R 很喜欢玩一个双人的棋类游戏，游戏规则如下：

在一个大小为 n*n 的棋盘上，有一个白色的棋子，初始位置为(wx, wy)，与一个黑色的棋子，初始位置为(bx, by)。玩家 L 执白先行，玩家 R 执黑后行，两人交替行棋。

如果当前是玩家 L 行棋，玩家 L 可以在上下左右四个方向中选一个并让他的棋子在该方向前进一格；如果当前是玩家 R 行棋，玩家 R 可以在上下左右四个方向中选一个并让他的棋子在该方向前进一格或两格（均不能走出棋盘）。一个人取得胜利当且仅当他的棋子走到了对方的棋子当前所在的位置。

玩家 L 与玩家 R 都采取同样的策略行棋：如果一方能赢，一定会用尽量少的步数去赢；如果一方会输，一定会拖尽量多的步数才输。

假设玩家 L 与玩家 R 都绝顶聪明，行棋中途均不犯错误，你能提前预测最终的胜者以及棋局持续的步数吗？

数据规模：$2 \le n \le 20$

分析

[6] TopCoder Algorithm SRM 423, Div 1, 500。

这是一类典型的不平等博弈问题，我们平时常见的中国象棋、国际象棋等棋类游戏实际上就是上述问题的加强版。下面我们考虑如何解决这个例题。

首先，由于棋盘上只有两个棋子，且棋盘也不算很大($1 \le n \le 20$)，所以我们考虑将游戏中所有可能出现的局面都表示出来，然后再计算每个局面的胜负情况。

考虑如何表示一个局面，我们不难得出可以用一个五元组(x_1, y_1, x_2, y_2, cur)来刻画一个局面，其中(x_1, y_1)表示白色棋子当前的位置，(x_2, y_2)表示黑色棋子当前的位置，cur 表示当前轮到玩家 L 行棋还是玩家 R 行棋。接下来要考虑的就是如何描述一个局面的胜负情况，由于题目不仅要求确定胜者而且还要求确定棋局持续的步数，因此对于一个局面，我们可以用两个量 winner 和 move 分别描述局面的胜者和持续的步数。但是这样描述不仅显得繁琐而且在状态转移的时候也显得很不方便，因此我们考虑用如下方式来进行描述：

对于一个局面 G，我们用函数 f(G)来描述 G 的胜负情况。定义 infinite 为一个很大的正整数，不妨设为 10^8。如果局面 G 的胜者为玩家 L 且棋局持续 x 步，则 f(G) = infinite – x；如果局面 G 的胜者为玩家 R 且棋局持续 x 步，则 f(G) = -infinite + x。

举个例子，如果一个局面 H 的胜者为玩家 L 且棋局将持续 5 步，则 f(H) = infinite – 5；如果 f(H) = -infinite + 3，则局面 H 的胜者为玩家 R 且棋局将持续 3 步。

设 G_L = (wx, wy, bx, by, L)，那么我们的目标就是确定 f(G_L)的值。由于局面与局面之间的拓扑关系并不清晰，因此我们考虑先确定一些终止局面对应的 f 函数值，然后再用逆推的方法得出 f(G_L)。

对于终止局面，不难得出只有两种情况：(x, y, x, y, L)和(x, y, x, y, R)。且容易得出 f(x, y, x, y, L) = -infinite，f(x, y, x, y, R) = infinite。

下面我们尝试列出状态转移方程。因为题目里面说到两个玩家采取的策略都是能赢的话用尽量少的步数赢，输的话拖尽量多的步数才输，因此对于一个局面 G，玩家 L 的目标就是使 f(G)尽可能大，玩家 R 的目标就是使 f(G)尽可能小，所以我们可以得出如下状态转移方程：

$$f(x_1, y_1, x_2, y_2, L) = max\{ f(x_1', y_1', x_2, y_2, R) - sign(f(x_1', y_1', x_2, y_2, R)) \}$$

$$f(x_1, y_1, x_2, y_2, R) = min\{ f(x_1, y_1, x_2', y_2', L) - sign(f(x_1, y_1, x_2', y_2', L)) \}$$

其中$(x_1, y_1)\neq(x_2, y_2)$，(x_1', y_1')为白色棋子在(x_1, y_1)时走一步可以到达的位置，(x_2', y_2')为黑色棋子在(x_2, y_2)时走一步可以到达的位置，$sign(x)=\begin{cases}1, x>0\\0, x=0\\-1, x<0\end{cases}$。

到这里，本题似乎已经可以解决了，但是仔细分析，不难发现局面的拓扑关系并不清晰，我们并不能确定一个计算局面的顺序，那么应该如何解决这个问题？注意到局面与局面之间虽然会有互推的关系，但是并不可能存在一个局面的值最后又推回到自身，即类似于求最短路时出现的负权回路的情况，因此我们可以用类似于 SPFA 的迭代算法来解决局面的计算顺序问题，算法描述如下：

```
Count(G')
初始化 f，对于所有的局面 G，令 f(G) = 0
枚举所有的终止局面 Ge，确定 f(Ge)的值，并将 Ge 放入队列 q
while q 不为空
    取出队首元素，并令其为 Y
    for 每个可以一步到达局面 Y 的局面 X
        tmp ← f(X)
        根据状态转移方程重新计算 f(X)
        if tmp ≠ f(X) and X 不在队列 q 中 then 将 X 放入队列 q
return f(G')
```

下面我们分析一下上述算法的时间复杂度，局面的数目为 n^4，而转移的数目与局面的数目是同阶的，因此算法的时间复杂度为 $O(kn^4)$，在平均情况下 k 为一个比较小的常数。由于 $n \le 20$，因此上述算法已经能够很好地解决本题。

扩展

上述的算法不仅清晰、简洁，而且我们还发现这个算法并非只适用于本题，对于很多局面不多的游戏我们都可以用上述算法来解决。当局面与局面之间的拓扑关系清晰的时候，我们往往可以采用动态规划算法按拓扑顺序计算局面的胜负情况，但是当局面与局面之间不存在明显的拓扑关系，无法或者很难确定局面计算顺序的时候，上述的迭代算法不失为一个解决问题的有力工具。

四、总结

要解决好不平等博弈问题，需要灵活运用各种分析方法，并融入各种深刻的算法思想，从而高效地解决问题。随着信息学竞赛的发展，各种各样灵活多变的博弈问题层出不穷。希望本文的思想能为大家打开一扇窗，在遇到博弈问题的时候多一种思路。而要在竞赛非常有限的时间内解决好问题，必须多思考多实践。正所谓“纸上得来终觉浅，绝知此事要躬行”，希望读者在阅此拙文后，能够有所启发。

感谢

感谢唐文斌教练和胡伟栋教练的指导。

感谢阮志远老师和万杰老师多年的训练与指导。

感谢中山一中的黄子畦同学的帮助。

参考文献

1、On Numbers and Games: J. H. Conway 著

2、Surreal Numbers - An Introduction: Claus Tøndering 著

3、Game Theory: Thomas S. Ferguson著

4、国家集训队 2002 年论文：张一飞《由感性认识到理性认识——透析一类搏弈游戏的解答过程》

5、国家集训队 2002 年论文：骆骥《浅析解 “对策问题” 的两种思路——从“取石子”问题谈起》

6、国家集训队 2007 年论文：王晓珂《解析一类组合游戏》

附录

附录 1 论文原题

Procrastination 题目描述

Once upon a time, there were two graduate students that were best friends.

During their short breaks from research (usually, not longer than several hours), the two students liked to play the game of Procrastination.

The game of Procrastination is for two players (black and white), who take turns moving. The game involves removing cubes from towers. Each cube is either black or white. At the start of the game, these cubes are arranged into 4 towers: each tower is a stack of several cubes. On a player's turn, he can remove any cube that matches his color (the white player removes only white cubes, and the black player only removes black cubes). All the cubes above the chosen cube are also removed from the tower, irrespective of color. For example, suppose a tower is composed of the following cubes (from bottom to top): black, white, black, white. Then, if black removes the bottom-most black cube, he removes the entire tower, black can also take the 3rd cube, removing the 4th cube with it. If white removes the 2nd cube, then only one black cube will remain, white can also take the 4th cube. If a player cannot remove any cubes, he loses.

Having already been trained in the intricacies of the game during their undergraduate years, the two students learned to play the game perfectly, i.e., if a player had a winning strategy, then that player would win the game. However, at some time, they discovered that, for most starting configurations, one of the players has the winning strategy irrespective of which player moves first. They called a configuration a W-configuration if white has a winning strategy irrespective of who moves first, and a B-configuration if black has a winning strategy irrespective of who moves first.

Moreover, the friends noted that some partial configurations are at least as favorable for one player as other configurations. A partial configuration C is defined as a set of 3 towers, note that a partial configuration C together with a 4th tower T forms a complete game configuration, which we denote as (C, T). A formal definition of the notion "at least as favorable", is as follows. A partial configuration C1 (composed of 3 towers) is at least as favorable for white as another partial configuration C2 (also composed of 3 towers) if and only if for any 4th tower T, if (C2, T) is a W-configuration then (C1, T) is also a W-configuration. In other words,

there does not exist a 4th tower T such that (C1, T) is not a W-configuration and (C2, T) is a W-configuration.

Given two partial configurations C1 and C2, you are to check whether C1 is at least as favorable for white as the partial configuration C2.

Input

The first line of the input contains an integer, the number of test cases. A test case includes one line with Test N, where N is the current test case number followed by eight lines, specifying the two partial configurations C1 and C2 in this order. Each configuration is specified by four lines.

The first line of the partial configuration contains three numbers: n1, n2, n3 denoting the heights of the three towers of the partial configuration ($0 \leqslant n1, n2, n3 \leqslant 50$). The second line contains n1 letters (B or W) separated by spaces describing the first tower. The third line contains n2 letters separated by spaces describing the second tower. The fourth line contains n3 letters separated by spaces describing the third tower. A letter W denotes a white cube and the letter B denotes a black cube. Each tower is described in the bottom-to-top order.

Output

For each test case, print on a separate line the test case number and Yes if C1 is at least as favorable for white as the partial configuration C2, and No otherwise.

Sample Input

2

Test 1

3 3 1

W B B

W B W

B

3 3 3

B W W

B W W

W B B

Test 2

3 3 2

W B B

W B W

B B

3 3 3

B W W

B W W

W B B

Sample Output

Test 1: Yes

Test 2: No

The Easy Chase 题目描述

Two players play a simple game on a n x n board. The first player has a single white checker which is initially located at (rowWhite, colWhite). The second player has a single black checker which is initially located at (rowBlack, colBlack). All coordinates are 1-based. The two players alternate turns, and the first player moves first.

When it is the first player's turn, he chooses one of four directions (up, down, left or right) and moves his checker one cell in the chosen direction. When it is the second player's turn, he also chooses one of those four directions and moves his checker one or two cells in the chosen direction. A player wins the game when his move puts his checker in the cell occupied by his opponent's checker.

Both players use an optimal game strategy. If the player can win, he will follow the strategy that minimizes the number of moves in the game. If the player cannot win,

he will follow the strategy that maximizes the number of moves in the game.

If the first player will win, return "WHITE x", and if the second player will win, return "BLACK x", where x is the number of moves in the game (all quotes for clarity).

Definition

Class: TheEasyChase

Method: winner

Parameters: int, int, int, int, int

Returns: String

Method signature: String winner(int n, int rowWhite, int colWhite, int rowBlack, int colBlack)

(be sure your method is public)

Constraints

- n will be between 2 and 20, inclusive.
- rowWhite will be between 1 and n, inclusive.
- colWhite will be between 1 and n, inclusive.
- rowBlack will be between 1 and n, inclusive.
- colBlack will be between 1 and n, inclusive.
- (rowWhite, colWhite) and (rowBlack, colBlack) will represent different cells.

Examples

0)

2

1

1

2

2

Returns: "BLACK 2"

There are two possible moves for the first player. But he will lose the game anyway.

1)

2

2

2

1

2

Returns: "WHITE 1"

Just one move in this game.

2)

3

1

1

3

3

Returns: "BLACK 6"

附录 2 参考程序

procrastination_1

例一程序：

procrastination_2

例二程序：

TheEasyChase

例三程序：