宫水三叶的刷题日记

博弈论

Author ：宫水三叶
Date ：2021/10/07
QQ Group：703311589
WeChat ：oaoaya

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

噔噔噔噔，这是公众号「宫水三叶的刷题日记」的原创专题「博弈论」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「博弈论」即可获取最新下载链接。

💡**下面介绍使用本合集的最佳使用实践：**

学习算法：

1. 打开在线目录（Github 版 & Gitee 版）；
2. 从侧边栏的类别目录找到「博弈论」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题‘
4. 拿到题号之后，回到本合集进行检索。

维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流 🍭🍭🍭

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **292. Nim 游戏** ，难度为 **简单**。

Tag : 「博弈论」

你和你的朋友，两个人一起玩 Nim 游戏：

- 桌子上有一堆石头。
- 你们轮流进行自己的回合，你作为先手。
- 每一回合，轮到的人拿掉 1 - 3 块石头。
- 拿掉最后一块石头的人就是获胜者。

假设你们每一步都是最优解。请编写一个函数，来判断你是否可以在给定石头数量为 n 的情况下赢得游戏。如果可以赢，返回 true；否则，返回 false 。

示例 1：

```
输入：n = 4

输出：false

解释：如果堆中有 4 块石头，那么你永远不会赢得比赛；
     因为无论你拿走 1 块、2 块 还是 3 块石头，最后一块石头总是会被你的朋友拿走。
```

示例 2：

```
输入：n = 1

输出：true
```

示例 3：

```
输入：n = 2

输出：true
```

提示：

- 1 <= n <= 2^{31} - 1

博弈论

这是一道 Nim 游戏的简化版。

在不知晓博弈论结论前，可以先通过找规律得到猜想，然后再从「何种情况下，先手会处于必胜态」的角度来进行分析。

根据题意，我们尝试从小范围数据的情况进行讨论：

1. 如果落到先手的局面为「**石子数量为 1 - 3**」的话，那么**先手必胜**；
2. 如果落到先手的局面为「**石子数量为 4**」的话，那么先手决策完（无论何种决策），交到后手的局面为「石子数量为 1 - 3」，即此时后手必胜，对应**先手必败**（到这里我们有一个推论：如果交给先手的局面为 4 的话，那么先手必败）；

3. 如果落到先手的局面为「**石子数量为 5 - 7**」的话，那么先手可以通过控制选择石子的数量，来使得后手处于「石子数量为 4」的局面（此时后手必败），因此**先手必胜**；
4. 如果落到先手的局面为「**石子数量为 8**」的话，由于每次只能选 1 - 3 个石子，因此交由后手的局面为 5 - 7，根据流程 3 我们知道此时**先手必败**；

   ...

到这里，我们猜想 **当起始局面石子数量为 4 的倍数，则先手必败，否则先手必胜（即 `n % 4 != 0` 时，先手必胜）。**

然后我们通过「归纳法」证明一下该猜想的正确性。

在上面的「找规律」分析中，我们分情况讨论了最后一个决胜回合（我们称「剩余石子数量少于等于 4 的局面」为最后回合）的情况：**如果交由先手的石子数量为 4，那么先手必败，否则先手必胜。**

而对于「最后回合」前的任意回合（石子数量大于 4），我们需要证明 **先手可以通过调整所选石子数量，来维持「`n % 4 != 0`」直到最后回合。**

如果起始对先手而言满足「`n % 4 != 0`」，此时先手可以通过选择石子数量为「`n % 4`」来确保交到后手的局面为 4 的倍数。

那么根据推论，此时的原始后手作为下一回合的先手角色，且面临石子数量为 4 的倍数的局面，为必败态。

进一步的解释就是，由于原始后手面临石子数量为 4 的倍数的局面，且只能选 1 - 3 个石子，因此无论如何选择，重新回到原始先手的仍然满足「`n % 4 != 0`」（非 4 的倍数）。

因此 **原始先手只需要确保每次都选择「`x % 4`」个石子（x 为当前石子数量），就可以确保交由自己的局面一直满足「`x % 4 != 0`」，交由对方的局面一直满足「`x % 4 == 0`」，直到最后回合的到来。**

至此，我们证明了 **如果起始石子数量 n 满足「`n % 4 != 0`」条件，那么先手必胜。**

代码：

```java
class Solution {
    public boolean canWinNim(int n) {
        return n % 4 != 0;
    }
}
```

- 时间复杂度：$O(1)$
- 空间复杂度：$O(1)$

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **810. 黑板异或游戏** ，难度为 **困难**。

Tag : 「博弈论」、「数学」、「异或」

黑板上写着一个非负整数数组 nums[i] 。

Alice 和 Bob 轮流从黑板上擦掉一个数字，Alice 先手。如果擦除一个数字后，剩余的所有数字按位异或运算得出的结果等于 0 的话，当前玩家游戏失败。 (另外，如果只剩一个数字，按位异或运算得到它本身；如果无数字剩余，按位异或运算结果为 0。）

换种说法就是，轮到某个玩家时，如果当前黑板上所有数字按位异或运算结果等于 0，这个玩家获胜。

假设两个玩家每步都使用最优解，当且仅当 Alice 获胜时返回 true。

示例：

```
输入: nums = [1, 1, 2]

输出: false

解释:
Alice 有两个选择: 擦掉数字 1 或 2。
如果擦掉 1, 数组变成 [1, 2]。剩余数字按位异或得到 1 XOR 2 = 3。那么 Bob 可以擦掉任意数字，因为 Alice 会成为擦掉最后一个数
如果 Alice 擦掉 2，那么数组变成[1, 1]。剩余数字按位异或得到 1 XOR 1 = 0。Alice 仍然会输掉游戏。
```

提示：

- 1 <= N <= 1000
- 0 <= nums[i] <= 2^{16}

基本分析

这是一道「博弈论」题。

如果没接触过博弈论，其实很难想到，特别是数据范围为 10^3，很具有迷惑性。

如果接触过博弈论，对于这种「判断先手后手的必胜必败」的题目，博弈论方向是一个优先考虑的方向。

根据题意，如果某位玩家在操作前所有数值异或和为 0，那么该玩家胜利。要我们判断给定序列时，先手是处于「必胜态」还是「必败态」，如果处于「必胜态」返回 `True`，否则返回 `False`。

对于博弈论的题目，通常有两类的思考方式：

1. 经验分析：见过类似的题目，猜一个性质，然后去证明该性质是否可推广。
2. 状态分析：根据题目给定的规则是判断「胜利」还是「失败」来决定优先分析「必胜态」还是「必败态」时具有何种性质，然后证明性质是否可推广。

博弈论

对于本题，给定的是判断「胜利」的规则（在给定序列的情况下，如果所有数值异或和为 0 可立即判断胜利，其他情况无法立即判断胜负），那么我们应该优先判断何为「先手必胜态」，如果不好分析，才考虑分析后手的「必败态」。

接下来是分情况讨论：

1. 如果给定的序列异或和为 0，游戏开始时，先手直接获胜：

由此推导出性质一：**给定序列 `nums` 的异或和为 0，先手处于「必胜态」，返回 `True`。**

2. 如果给定序列异或和不为 0，我们需要分析，先手获胜的话，序列会满足何种性质：

显然如果要先手获胜，则需要满足「**先手去掉一个数，剩余数值异或和必然不为 0；同时后手去掉一个数后，剩余数值异或和必然为 0**」。

换句话说，我们需要分析**什么情况下「经过一次后手操作」后，序列会以上述情况 1 的状态，回到先手的局面**。

也就是反过来分析想要出现「后手必败态」，序列会有何种性质。

假设后手操作前的异或和为 Sum（$Sum \neq 0$），「后手必败态」意味着去掉任意数字后异或和为 0。

同时根据「相同数值异或结果为 0」的特性，我们知道去掉某个数值，等价于在原有异或和的基础上异或上这个值。

则有：

$$Sum' = Sum \oplus nums[i] = 0$$

由于是「后手必败态」，因此 i 取任意一位，都满足上述式子。

则有：

$$Sum \oplus nums[0] = ... = Sum \oplus nums[k] = ... = Sum \oplus nums[n-1] = 0$$

同时根据「任意数值与 0 异或数值不变」的特性，我们将每一项进行异或：

$$(Sum \oplus nums[0]) \oplus ... \oplus (Sum \oplus nums[k]) \oplus ... \oplus (Sum \oplus nums[n-1]) = 0$$

根据交换律进行变换：

$$(Sum \oplus Sum \oplus ... \oplus Sum) \oplus (nums[0] \oplus ... \oplus nums[k] \oplus ... \oplus nums[n-1]) = 0$$

再结合 Sum 为原序列的异或和可得：

$$(Sum \oplus Sum \oplus ... \oplus Sum) \oplus Sum = 0, Sum \neq 0$$

至此，我们分析出当处于「后手必败态」时，去掉任意一个数值会满足上述式子。

根据「相同数值偶数次异或结果为 0」的特性，可推导出「后手必败态」会导致交回到先手的序

列个数为偶数，由此推导后手操作前序列个数为奇数，后手操作前一个回合为偶数。

到这一步，我们推导出想要出现「后手必败态」，先手操作前的序列个数应当为偶数。

那么根据先手操作前序列个数为偶数（且异或和不为 0），是否能够推导出必然出现「后手必败态」呢？

显然是可以的，因为如果不出现「后手必败态」，会与我们前面分析过程矛盾。

假设先手操作前异或和为 Xor（序列数量为偶数，同时 $Xor \neq 0$），如果最终不出现「后手必败态」的话，也就是先手会输掉的话，那么意味着有 $Xor \oplus nums[i] = 0$，其中 i 为序列的任意位置。利用此性质，像上述分析那样，将每一项进行展开异或，会得到奇数个 Xor 异或结果为 0，这与开始的 $Xor \neq 0$ 矛盾。

由此推导出性质二：**只需要保证先手操作前序列个数为偶数时就会出现「后手必败态」，从而确保先手必胜。**

综上，如果序列 `nums` 本身异或和为 0，天然符合「先手必胜态」的条件，答案返回 `True` ；如果序列 `nums` 异或和不为 0，但序列长度为偶数，那么最终会出现「后手必败态」，推导出先手必胜，答案返回 `True` 。

代码：

```
class Solution {
    public boolean xorGame(int[] nums) {
        int sum = 0;
        for (int i : nums) sum ^= i;
        return sum == 0 || nums.length % 2 == 0;
    }
}
```

- 时间复杂度：$O(n)$
- 空间复杂度：$O(1)$

总结

事实上，在做题的时候，我也是采取「先假定奇偶性，再证明」的做法，因为这样比较快。

但「假定奇偶性」这一步是比较具有跳跃性的，这有点像我前面说到的「经验分析解法」，而本题解证明没有做任何的前置假定，单纯从「先手必胜态」和「后手必败态」进行推导，最终推导出「先手序列偶数必胜」的性质 ，更符合前面说到的「状态分析解法」。

两种做法殊途同归，在某些博弈论问题上，「经验分析解法」可以通过「归纳」&「反证」很好分析出来，但这要求选手本身具有一定的博弈论基础；而「状态分析解法」则对选手的题量要求低些，逻辑推理能力高些。

两种方法并无优劣之分，都是科学严谨的做法。

我十分建议大家将此题解与 官方题解 一同阅读，体会两种分析方法的区别。

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **877. 石子游戏** ，难度为 **中等**。

Tag : 「区间 DP」、「博弈论」

亚历克斯和李用几堆石子在做游戏。偶数堆石子排成一行，每堆都有正整数颗石子 piles[i] 。

游戏以谁手中的石子最多来决出胜负。石子的总数是奇数，所以没有平局。

亚历克斯和李轮流进行，亚历克斯先开始。 每回合，玩家从行的开始或结束处取走整堆石头。这种情况一直持续到没有更多的石子堆为止，此时手中石子最多的玩家获胜。

假设亚历克斯和李都发挥出最佳水平，当亚历克斯赢得比赛时返回 true ，当李赢得比赛时返回 false 。

示例：

```
输入：[5,3,4,5]
输出：true
解释：
亚历克斯先开始，只能拿前 5 颗或后 5 颗石子 。
假设他取了前 5 颗，这一行就变成了 [3,4,5] 。
如果李拿走前 3 颗，那么剩下的是 [4,5]，亚历克斯拿走后 5 颗赢得 10 分。
如果李拿走后 5 颗，那么剩下的是 [3,4]，亚历克斯拿走后 4 颗赢得 9 分。
这表明，取前 5 颗石子对亚历克斯来说是一个胜利的举动，所以我们返回 true 。
```

提示：

- 2 <= piles.length <= 500
- piles.length 是偶数。
- 1 <= piles[i] <= 500
- sum(piles) 是奇数。

动态规划

定义 $f[l][r]$ 为考虑区间 $[l, r]$，在双方都做最好选择的情况下，先手与后手的最大得分差值为多少。

那么 $f[1][n]$ 为考虑所有石子，先手与后手的得分差值：

- $f[1][n] > 0$，则先手必胜，返回 `True`
- $f[1][n] < 0$，则先手必败，返回 `False`

不失一般性的考虑 $f[l][r]$ 如何转移。根据题意，只能从两端取石子（令 $piles$ 下标从 1 开始），共两种情况：

- 从左端取石子，价值为 $piles[l - 1]$；取完石子后，原来的后手变为先手，从 $[l + 1, r]$ 区间做最优决策，所得价值为 $f[l + 1][r]$。**因此本次先手从左端点取石子的话，双方差值为：**

$$piles[l-1] - f[l+1][r]$$

- 从右端取石子，价值为 $piles[r - 1]$；取完石子后，原来的后手变为先手，从 $[l, r - 1]$ 区间做最优决策，所得价值为 $f[l][r - 1]$。**因此本次先手从右端点取石子的话，双方差值为：**

$$piles[r-1] - f[l][r-1]$$

双方都想赢，都会做最优决策（即使自己与对方分差最大）。因此 $f[l][r]$ 为上述两种情况中的最大值。

根据状态转移方程，我们发现大区间的状态值依赖于小区间的状态值，典型的区间 DP 问题。

按照从小到大「枚举区间长度」和「区间左端点」的常规做法进行求解即可。

代码：

```
class Solution {
    public boolean stoneGame(int[] ps) {
        int n = ps.length;
        int[][] f = new int[n + 2][n + 2];
        for (int len = 1; len <= n; len++) { // 枚举区间长度
            for (int l = 1; l + len - 1 <= n; l++) { // 枚举左端点
                int r = l + len - 1; // 计算右端点
                int a = ps[l - 1] - f[l + 1][r];
                int b = ps[r - 1] - f[l][r - 1];
                f[l][r] = Math.max(a, b);
            }
        }
        return f[1][n] > 0;
    }
}
```

- 时间复杂度：$O(n^2)$
- 空间复杂度：$O(n^2)$

博弈论

事实上，这还是一道很经典的博弈论问题，也是最简单的一类博弈论问题。

为了方便，我们称「石子序列」为石子在原排序中的编号，下标从 1 开始。

由于石子的堆数为偶数，且只能从两端取石子。**因此先手后手所能选择的石子序列，完全取决于先手每一次决定。**

由于石子的堆数为偶数，对于先手而言：**每一次的决策局面，都能「自由地」选择奇数还是偶数**

的序列，从而限制后手下一次「只能」奇数还是偶数石子。

具体的，对于本题，**由于石子堆数为偶数，因此先手的最开始局面必然是** $[奇数, 偶数]$**，即必然是「奇偶性不同的局面」；当先手决策完之后，交到给后手的要么是** $[奇数, 奇数]$ **或者** $[偶数, 偶数]$**，即必然是「奇偶性相同的局面」；后手决策完后，又恢复「奇偶性不同的局面」交回到先手** ...

不难归纳推理，这个边界是可以应用到每一个回合。

因此先手只需要在进行第一次操作前计算原序列中「奇数总和」和「偶数总和」哪个大，然后每一次决策都「限制」对方只能选择「最优奇偶性序列」的对立面即可。

同时又由于所有石子总和为奇数，堆数为偶数，即没有平局，所以先手必胜。

代码：

```
class Solution {
    public boolean stoneGame(int[] piles) {
        return true;
    }
}
```

- 时间复杂度：$O(1)$
- 空间复杂度：$O(1)$

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

💡**更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。**

最新专题合集资料下载，可关注公众号「宫水三叶的刷题日记」，回台回复「博弈论」获取下载链接。

觉得专题不错，可以请作者吃糖 🍭🍭🍭 ：


“给作者手机充个电”
YOLO 的赞赏码