LeetCode 第 137 号问题：只出现一次的数字 II

> *本文首发于公众号「图解面试算法」，是 图解 LeetCode 系列文章之一。*
>
> *同步博客：https://www.algomooc.com*

题目来源于 LeetCode 上第 137 号问题：只出现一次的数字 II。题目难度为 Medium，目前通过率为 66.7% 。

题目描述

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现了三次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

示例 1:

```
输入: [2,2,3,2]
输出: 3
```

示例 2:

```
输入: [0,1,0,1,0,1,99]
输出: 99
```

题目解析

相比 Single Number，输入数组的条件变了，数组中除了其中的一个元素只出现了一次，其余的元素都出现了 **三** 次，最后的问题还是让你找出这个只出现一次的元素。这道题目，一开始看起来从位运算思考貌似是不可能的，但如果你从集合的角度去思考或许可以想到解法。如果我们遍历数组里面的元素，在遍历的过程中，我们会发现 **对于每个元素来说只有三种情况，出现一次，出现两次，出现三次。**因为我们要找的是出现一次的那个元素，而且最终除了我们要找的元素，其他所有的元素都会出现三次，因此我们需要想办法排除掉出现三次的元素。一开始的时候可以想，我们用两个集合，集合 1 用于存放出现一次的元素，集合 2 用于存放出现两次的元素，于是我们可以发现下面的逻辑对应关系：

```
如果遍历到的元素不在集合 1 中，也不在集合 2 中： 该元素第一次出现，加入集合 1
如果遍历到的元素在集合 1 中，不在集合 2 中： 该元素第二次出现，移出集合 1，加入集合 2
如果遍历到的元素不在集合 1 中，在集合 2 中： 该元素第三次出现，移出集合 2
```

上面的逻辑对应关系你应该很容易理解，但是我想说的是通过位操作可以做到这一点，我们不需要真正的集合，我们只需要用一个整数来代替集合即可。怎么解释呢？假设我们用整数 `ones` 表示集合 1，整数 `twos` 表示集合 2，这两个整数的值初始化均为 0。`ones ^ ele[i]` 表示把元素 `ele[i]` 加入到集合 1 中，如果说下一个元素 `ele[i + 1]` 来了，并且 `ele[i] != ele[i + 1]` ，那么 `ones ^ ele[i] ^ ele[i + 1]` 肯定会产生一个不为零的值，至于这个值是多少，你不用关心。但如果 `ele[i] == ele[i + 1]` ，那么 `ones ^ ele[i] ^ ele[i + 1]` 的结果肯定为 0，到这里，你应该知道通过异或运算，我们已经可以做到，将出现一次的元素加入集合 1，将出现两次的元素移出集合 1。但是这还不够，因为元素还有可能出现三次，如果仅仅是上面的异或表达式，第三次出现的元素还是会被加入到集合 1，我们还需要保证该元素不在集合 2 中，`(ones ^ ele[i]) & (~twos)` 就可以保证这一点。对集合 2 来说也是一样的，`(twos ^ ele[i]) & (~ones)` 保证将不存在于集合 1 中，且不存在集合 2 中的元素加入到集合 2。如果我们先更新集合 1，再更新集合 2，就可以实现我们之前说的逻辑对应关系。说到这里，如果你还是不理解，那么你 **可以尝试把一个元素看作是一堆值为 1 的 bit 位的组合**，比如 12 的二进制是 `0001`

`0100`，如果说 12 出现了三次，那么从右往左数第三位和第五位 bit 的就出现了三次。我们把这个结论放在数组中也是一样的，对于那些出现了 3 的整数倍次的 bits 位我们要进行消除，找到那些出现了 `3 * n + 1` 次的 bit 位，将它们组合在一起就是我们要找的元素，上面的位运算做的就是这个事情，与其说把元素放入集合中，我们也可以说 **将元素的所有值为 1 的 bit 位放入集合中**，这样会更好理解些。

动画演示

代码实现

C++

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int one=0, two=0;
        for(int n:nums)
        {
            one = (one ^ n) & (~two);
            two = (two ^ n) & (~one);
        }
        return one;
    }
};
```

Java

```
class Solution {
    public int singleNumber(int[] nums) {
        int one=0, two=0;
        for(int n:nums)
        {
            one = (one ^ n) & (~two);
            two = (two ^ n) & (~one);
        }
        return one;
    }
}
```

Python

```
class Solution(object):
    def singleNumber(self, nums):
        one = two = 0
        for n in nums:
            one = (one ^ n) & (~two)
```

```
            two = (two ^ n) & (~one)
        return one
```