LeetCode 第 260 号问题：只出现一次的数字 III

> *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 260 号问题：只出现一次的数字 III。题目难度为 Medium，目前通过率为 72.1% 。

题目描述

给定一个整数数组 `nums`，其中恰好有两个元素只出现一次，其余所有元素均出现两次。 找出只出现一次的那两个元素。

示例：

```
输入：[1,2,1,3,2,5]
输出：[3,5]
```

注意：

1. 结果输出的顺序并不重要，对于上面的例子， [5, 3] 也是正确答案。
2. 你的算法应该具有线性时间复杂度。你能否仅使用常数空间复杂度来实现？

题目解析

第三道题目，和第一道题目只变化了一点，就是输入数组中除了 **两** 个元素出现了一次，其余的都出现了两次。我们依然可以从第一道题目的解法去思考这道题，如果我们还是按照第一题的做法，最后我们得到的答案将会是 `ele1 ^ ele2` 的结果，我们需要思考如何从这个结果出发，得到 `ele1` 和 `ele2`。首先思考一个问题 `ele1 ^ ele2` 的结果具体是什么，或者说里面有什么样的信息，异或操作是将相同的 bit 位置 0，相异的置 1，也就是说 **`ele1` 和 `ele2` 异或的结果中为 1 的 bit 位是两个元素相异的 bit 位，再进一步讲，我们可以用这个 bit 位来区分两个元素。**于是在第一题的基础之上，用一个 bit 位作为判断条件，来决定当前遍历到的元素和那个值进行异或，因为这时我们要求的值有两个。

从上面这些题目中你可以看到位运算的强大，三道系列题目的时间复杂度均为 `O(n)`，但是位运算是更加底层的运算，实际时间消耗会比正常操作要更快一些。在理解位运算的时候，**试着把 bit 作为最小单位去思考，或许会有不一样的发现。**

代码实现

```java
public int[] singleNumber(int[] nums) {
    if (nums == null || nums.length == 0) {
        return new int[2];
    }

    int different = 0;
    for (int i : nums) {
        different ^= i;
    }
```

```
    // 这个操作是取 different 从左往右最后一个为 1 的 bit 位
    different &= -different;
    int[] ans = {0, 0};
    for (int i : nums) {
        if ((different & i) == 0) {
            ans[0] ^= i;
        } else {
            ans[1] ^= i;
        }
    }

    return ans;
}
```

动画演示