260. 只出现一次的数字 III

题目地址（260. 只出现一次的数字 III）

https://leetcode-cn.com/problems/single-number-iii/

标签

- 位运算

难度

- 中等

入选理由

1. 位运算基本是两者题型。 一种是直接考察位运算基础知识和基础 api 操作，另一种是实际应用，这里的应用又以状压压缩为主。因此第一道就是一个位运算的基本题目，另外我总结了好几道基础位运算题目， 推荐大家私底下都看看。明天我们是一个状态压缩应用题。

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：

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。
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?

前置知识

- 位运算
- 数组

- 哈希表

分析

这个题可以很直观的用哈希表来做，遍历一遍数组存入哈希表，再遍历一遍 key，找到 value 为 1 的那两个数就是最后答案，该解决方案的时间复杂度是线性，但是空间复杂度是$O(keys)$，不符合题意。

其实做过该系列的前两道题的应该都知道，本题可以巧用位运算的方式来实现$O(1)$空间复杂度，再具体来说使用到了异或(xor)的性质(这个要多做题总结，而不是第一眼看到题就能想出来用位运算)。

首先来再复习一下 xor 的主要性质

```
0 xor 1 = 1    0 xor 0 = 0
1 xor 1 = 0    1 xor 0 = 1
也就是说当比较的两个bit不同时，xor的结果才为1

a xor b = b xor a  满足交换律
a xor a = 0        与自身异或为0
```

也就是说，如果两个数相同那么必定 xor 没了，而该题说只有两个数出现一次，其他都是两次，再利用 0 xor 1 = 1 这条，可得如下解法：

- 将 nums 中所有数异或起来得到数 x，x 必定不为 0，因为相同的两个数都约掉了，相当于那两个只出现了一次的数进行 xor。
- 随便找一个 x 的 bit 为 1 的位置，为 1 就代表这两个出现一次的数在该位置的 bit 不同。
- 这样就可以根据这个为 1 的 bit 位来将原问题分解为两个子问题，子问题的定义是：给定一个数组，该数组只有一个数出现一次，其他数都出现两次。
- 这样就转换为基本的找出只出现一次数的问题了，直接将这个数组所有元素 xor 起来得到的就是答案。
- 为方便求解，本题使用的是低位最早出现 1 的位置。

代码

代码支持：Java,Python,JS,CPP

Java Code：

```
class Solution {
    public int[] singleNumber(int[] nums) {

        int xor = 0;
```

```
        for (int i : nums)
            xor ^= i;

        int mask = 1;
        while ((mask & xor) == 0)
            mask <<= 1;

        int[] res = new int[2];
        for (int i : nums) {

            if ((i & mask) == 0)
                res[0] ^= i;
            else
                res[1] ^= i;
        }
        return res;
    }
}
```

Python Code：

```
class Solution:
    def singleNumber(self, nums: List[int]) -> List[int]:
        xor = a = b = 0
        right_bit = 1
        length = len(nums)
        for i in nums:
            xor ^= i
        while right_bit & xor == 0:
            right_bit <<= 1
        for i in nums:
            if right_bit & i:
                a ^= i
            else:
                b ^= i
        return [a, b]
```

JS Code:

```
var singleNumber = function (nums) {
  let bitmask = 0;

  for (let n of nums) {
    bitmask ^= n;
  }

  bitmask &= -bitmask;
```

```
  const ret = [0, 0];

  for (let n of nums) {
    if ((n & bitmask) == 0) ret[0] ^= n;
    else ret[1] ^= n;
  }

  return ret;
};
```

CPP Code:

```
class Solution {
public:
    vector<int> singleNumber(vector<int>& nums) {
        int ret = 0;
        for (int n : nums)
            ret ^= n;
        int div = 1;
        while ((div & ret) == 0)
            div <<= 1;
        int a = 0, b = 0;
        for (int n : nums)
            if (div & n)
                a ^= n;
            else
                b ^= n;
        return vector<int>{a, b};
    }
};
```

复杂度分析

设：N个数

时间复杂度：$O(N)$

空间复杂度：$O(1)$

上一页 下一页