0137. 只出现一次的数字 II

ITCharge 大约 2 分钟

- 标签：位运算、数组
- 难度：中等

题目链接

- 0137. 只出现一次的数字 II - 力扣

题目大意

描述：给定一个整数数组 $nums$，除了某个元素仅出现一次外，其余每个元素恰好出现三次。

要求：找到并返回那个只出现了一次的元素。

说明：

- $1 \leq nums.length \leq 3 * 10^4$。
- $-2^{31} \leq nums[i] \leq 2^{31} - 1$。
- $nums$ 中，除某个元素仅出现一次外，其余每个元素都恰出现三次。

示例：

- 示例 1：

```py
输入：nums = [2,2,3,2]
输出：3
```

- 示例 2：

```py
输入：nums = [0,1,0,1,0,1,99]
输出：99
```

解题思路

思路 1：哈希表

1. 利用哈希表统计出每个元素的出现次数。
2. 再遍历一次哈希表，找到仅出现一次的元素。

思路 1：代码

```py
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        nums_dict = dict()
        for num in nums:
            if num in nums_dict:
                nums_dict[num] += 1
            else:
                nums_dict[num] =
        for key in nums_dict:
            value = nums_dict[key]
            if value == 1:
                return key
        return 0
```

思路 1：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 是数组 $nums$ 的元素个数。
- **空间复杂度**：$O(n)$。

思路 2：位运算

将出现三次的元素换成二进制形式放在一起，其二进制对应位置上，出现 1 的个数一定是 3 的倍数（包括 0）。此时，如果在放进来只出现一次的元素，则某些二进制位置上出现 1 的个数就不是 3 的倍数了。

将这些二进制位置上出现 1 的个数不是 3 的倍数位置值置为 1，是 3 的倍数则置为 0。这样对应下来的二进制就是答案所求。

注意：因为 Python 的整数没有位数限制，所以不能通过最高位确定正负。所以 Python 中负整数的补码会被当做正整数。所以在遍历到最后 31 位时进行 $ans-=(1<<31)$ 操作，目的是将负数的补码转换为「负号 + 原码」的形式。这样就可以正常识别二进制下的负数。参考：Two's Complement Binary in Python? - Stack Overflow

思路 2：代码

```py
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        ans = 0
        for i in range(32):
            count = 0
            for j in range(len(nums)):
                count += (nums[j] >> i) & 1
            if count % 3 != 0:
                if i == 31:
                    ans -= (1 << 31)
                else:
                    ans = ans |      i
        return ans
```

思路 2：复杂度分析

- **时间复杂度**：$O(n \log m)$，其中 n 是数组 $nums$ 的长度，m 是数据范围，本题中 $m = 32$。
- **空间复杂度**：$O(1)$。