

如何同时寻找缺失和重复的元素

 Stars 107k  B站 @labuladong 配套PDF和插件 下载 打卡挑战 报名 精品课程 查看




微信搜一搜

Q labuladong公众号

通知： 数据结构精品课 V1.6 持续更新中， 第八期打卡挑战 开始报名。

读完本文，你不仅学会了算法套路，还可以顺便解决如下题目：

牛客	LeetCode	力扣	难度
-	645. Set Mismatch	645. 错误的集合	

今天就聊一道很看起来简单却十分巧妙的问题，寻找缺失和重复的元素。之前的一篇文章 [常用的位操作](#) 中也写过类似的问题，不过这次的和上次的问题使用的技巧不同。

这是力扣第 645 题「[错误的集合](#)」，我来描述一下这个题目：

给一个长度为 N 的数组 `nums`，其中本来装着 $[1..N]$ 这 N 个元素，无序。但是现在出现了一些错误，`nums` 中的一个元素出现了重复，也就同时导致了另一个元素的缺失。请你写一个算法，找到 `nums` 中的重复元素和缺失元素的值。

```
// 返回两个数字，分别是 {dup, missing}
int[] findErrorNums(int[] nums);
```

比如说输入： `nums = [1,2,2,4]`，算法返回 `[2,3]`。

其实很容易解决这个问题，先遍历一次数组，用一个哈希表记录每个数字出现的次数，然后遍历一次 `[1..N]`，看看那个元素重复出现，那个元素没有出现，就 OK 了。

但问题是，这个常规解法需要一个哈希表，也就是 $O(N)$ 的空间复杂度。你看题目给的条件那么巧，在 `[1..N]` 的几个数字中恰好有一个重复，一个缺失，**事出反常必有妖**，对吧。

$O(N)$ 的时间复杂度遍历数组是无法避免的，所以我们可以想想办法如何降低空间复杂度，是否可以在 $O(1)$ 的空间复杂度之下找到重复和缺失的元素呢？

思路分析

这个问题的特点是，每个元素和数组索引有一定的对应关系。

我们现在自己改造下问题，**暂且将 `nums` 中的元素变为 `[0..N-1]`，这样每个元素就和一个数组索引完全对应了，这样方便理解一些。**

如果说 `nums` 中不存在重复元素和缺失元素，那么每个元素就和唯一一个索引值对应，对吧？

现在的问题是，有一个元素重复了，同时导致一个元素缺失了，这会产生什么现象呢？**会导致有两个元素对应到了同一个索引，而且会有一个索引没有元素对应过去。**

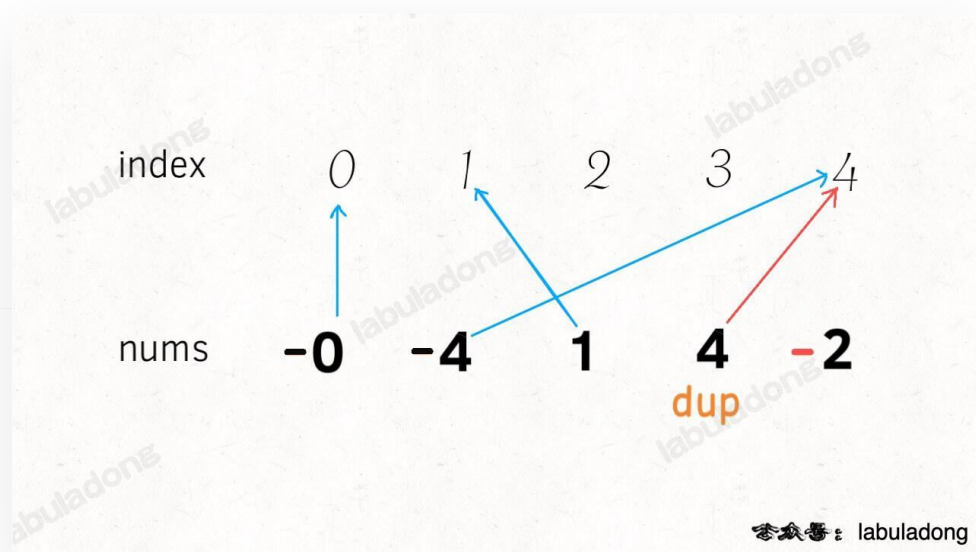
那么，如果我能够通过某些方法，找到这个重复对应的索引，不就是找到了那个重复元素么？找到那个没有元素对应的索引，不就是找到了那个缺失的元素了么？

那么，如何不使用额外空间判断某个索引有多少个元素对应呢？这就是这个问题的精妙之处了：

通过将每个索引对应的元素变成负数，以表示这个索引被对应过一次了，算法过程如下 GIF 所示：

index	0	1	2	3	4
nums	0	4	1	4	2

如果出现重复元素 4，直观结果就是，索引 4 所对应的元素已经是负数了：



对于缺失元素 3，直观结果就是，索引 3 所对应的元素是正数：



对于这个现象，我们就可以翻译成代码了：

```

int[] findErrorNums(int[] nums) {
    int n = nums.length;
    int dup = -1;
    for (int i = 0; i < n; i++) {
        int index = Math.abs(nums[i]);
        // nums[index] 小于 0 则说明重复访问
        if (nums[index] < 0)
            dup = Math.abs(nums[i]);
        else
            nums[index] *= -1;
    }

    int missing = -1;
    for (int i = 0; i < n; i++)
        // nums[i] 大于 0 则说明没有访问
        if (nums[i] > 0)
            missing = i;

    return new int[]{dup, missing};
}

```

这个问题就基本解决了，别忘了我们刚才为了方便分析，假设元素是 `[0..N-1]`，但题目要求是 `[1..N]`，所以只要简单修改两处地方即可得到原题的答案：

```

int[] findErrorNums(int[] nums) {
    int n = nums.length;
    int dup = -1;
    for (int i = 0; i < n; i++) {
        // 现在的元素是从 1 开始的
        int index = Math.abs(nums[i]) - 1;
        if (nums[index] < 0)
            dup = Math.abs(nums[i]);
        else
            nums[index] *= -1;
    }

    int missing = -1;
    for (int i = 0; i < n; i++)
        if (nums[i] > 0)
            // 将索引转换成元素
            missing = i + 1;
}

```

```
    return new int[]{dup, missing};  
}
```

其实，元素从 1 开始是有道理的，也必须从一个非零数开始。因为如果元素从 0 开始，那么 0 的相反数还是自己，所以如果数字 0 出现了重复或者缺失，算法就无法判断 0 是否被访问过。我们之前的假设只是为了简化题目，更通俗易懂。

最后总结

对于这种数组问题，**关键点在于元素和索引是成对儿出现的，常用的方法是排序、异或、映射。**

映射的思路就是我们刚才的分析，将每个索引和元素映射起来，通过正负号记录某个元素是否被映射。

排序的方法也很好理解，对于这个问题，可以想象如果元素都被从小到大排序，如果发现索引对应的元素如果不相符，就可以找到重复和缺失的元素。

异或运算也是常用的，因为异或性质 $a \oplus a = 0$, $a \oplus 0 = a$ ，如果将索引和元素同时异或，就可以消除成对儿的索引和元素，留下的就是重复或者缺失的元素。可以看看前文 [常用的位运算](#)，介绍过这种方法。

《labuladong 的算法小抄》已经出版，关注公众号查看详情；后台回复关键词「进群」可加入算法群；回复「PDF」可获取精华文章 PDF：



微信搜一搜

Q labuladong 公众号

共同维护高质量学习环境，评论礼仪[见这里](#)，违者直接拉黑不解释

6 Comments - powered by [utteranc.es](#)

Joycn2018 commented on Feb 20, 2022