宫水三叶的刷题日记

移除元素问题

Author : 宫水三叶
Date : 2021/10/07
QQ Group: 703311589
WeChat : oaoaya

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

噔噔噔噔，这是公众号「宫水三叶的刷题日记」的原创专题「移除元素问题」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「移除元素问题」即可获取最新下载链接。

💡下面介绍使用本合集的最佳使用实践：

学习算法：

1. 打开在线目录（Github 版 & Gitee 版）；
2. 从侧边栏的类别目录找到「移除元素问题」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题‘
4. 拿到题号之后，回到本合集进行检索。

维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流 🍭🍭🍭

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **26. 删除有序数组中的重复项** ，难度为 **简单**。

Tag：「数组」、「双指针」、「数组移除元素问题」

给你一个有序数组 nums ，请你 原地 删除重复出现的元素，使每个元素 只出现一次 ，返回删除后数组的新长度。

不要使用额外的数组空间，你必须在 原地 修改输入数组 并在使用 O(1) 额外空间的条件下完成。

说明：

为什么返回数值是整数，但输出的答案是数组呢?

请注意，输入数组是以「引用」方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。

你可以想象内部操作如下:

```
// nums 是以“引用”方式传递的。也就是说，不对实参做任何拷贝
int len = removeDuplicates(nums);

// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度, 它会打印出数组中 该长度范围内 的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

示例 1：

```
输入：nums = [1,1,2]

输出：2, nums = [1,2]

解释：函数应该返回新的长度 2 ，并且原数组 nums 的前两个元素被修改为 1, 2 。不需要考虑数组中超出新长度后面的元素。
```

示例 2：

```
输入：nums = [0,0,1,1,1,2,2,3,3,4]

输出：5, nums = [0,1,2,3,4]

解释：函数应该返回新的长度 5 ， 并且原数组 nums 的前五个元素被修改为 0, 1, 2, 3, 4 。不需要考虑数组中超出新长度后面的元素
```

提示：

- 0 <= nums.length <= 3 * 10^4
- -10^4 <= nums[i] <= 10^4
- nums 已按升序排列

双指针解法

一个指针 `i` 进行数组遍历，另外一个指针 `j` 指向有效数组的最后一个位置。

只有当 `i` 所指向的值和 `j` 不一致（不重复），才将 `i` 的值添加到 `j` 的下一位置。

代码：

```
class Solution {
    public int removeDuplicates(int[] nums) {
        int n = nums.length;
        int j = 0;
        for (int i = 0; i < n; i++) {
            if (nums[i] != nums[j]) {
                nums[++j] = nums[i];
            }
        }
        return j + 1;
    }
}
```

- 时间复杂度：$O(n)$
- 空间复杂度：$O(1)$

通用解法

为了让解法更具有一般性，我们将原问题的「最多保留 1 位」修改为「最多保留 k 位」。

对于此类问题，我们应该进行如下考虑：

- 由于是保留 `k` 个相同数字，**对于前 `k` 个数字，我们可以直接保留。**
- 对于后面的任意数字，能够保留的前提是：**与当前写入的位置前面的第 `k` 个元素进行比较，不相同则保留。**

举个🌰，我们令 `k=1`，假设有样例：`[3,3,3,3,4,4,4,5,5,5]`

0. 设定变量 `idx`，指向待插入位置。`idx` 初始值为 `0`，目标数组为 `[]`
1. 首先我们先让第 `1` 位直接保留（性质 1）。`idx` 变为 `1`，目标数组为 `[3]`
2. 继续往后遍历，能够保留的前提是与 `idx` 的前面 `1` 位元素不同（性质 2），因此我们会跳过剩余的 `3`，将第一个 `4` 追加进去。`idx` 变为 `2`，目标数组为 `[3,4]`
3. 继续这个过程，跳过剩余的 `4`，将第一个 `5` 追加进去。`idx` 变为 `3`，目标数组为 `[3,4,5]`

4. 当整个数组被扫描完，最终我们得到了目标数组 `[3,4,5]` 和 答案 `idx` 为 `3`。

代码：

```
class Solution {
    public int removeDuplicates(int[] nums) {
        return process(nums, 1);
    }
    int process(int[] nums, int k) {
        int idx = 0;
        for (int x : nums) {
            if (idx < k || nums[idx - k] != x) nums[idx++] = x;
        }
        return idx;
    }
}
```

- 时间复杂度：$O(n)$
- 空间复杂度：$O(1)$

基于上述解法我们还能做一点小剪枝：**利用目标数组的最后一个元素必然与原数组的最后一个元素相同进行剪枝，从而确保当数组有超过 `k` 个最大值时，数组不会被完整扫描。**

但需要注意这种「剪枝」同时会让我们单次循环的常数变大，所以仅作为简单拓展。

代码：

```
class Solution {
    public int removeDuplicates(int[] nums) {
        int n = nums.length;
        if (n <= 1) return n;
        return process(nums, 1, nums[n - 1]);
    }
    int process(int[] nums, int k, int max) {
        int idx = 0;
        for (int x : nums) {
            if (idx < k || nums[idx - k] != x) nums[idx++] = x;
            if (idx - k >= 0 && nums[idx - k] == max) break;
        }
        return idx;
    }
}
```

- 时间复杂度：$O(n)$
- 空间复杂度：$O(1)$

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **27. 移除元素** ，难度为 **简单**。

Tag：「数组」、「双指针」、「数组移除元素问题」

给你一个数组 nums 和一个值 val，你需要「原地」移除所有数值等于 val 的元素，并返回移除后数组的新长度。

不要使用额外的数组空间，你必须仅使用 $O(1)$ 额外空间并「原地」修改输入数组。

元素的顺序可以改变。

你不需要考虑数组中超出新长度后面的元素。

说明:

为什么返回数值是整数，但输出的答案是数组呢?

请注意，输入数组是以「引用」方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。

你可以想象内部操作如下:

```
// nums 是以“引用”方式传递的。也就是说，不对实参作任何拷贝
int len = removeElement(nums, val);

// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度, 它会打印出数组中 该长度范围内 的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

示例 1：

```
输入：nums = [3,2,2,3], val = 3

输出：2, nums = [2,2]

解释：函数应该返回新的长度 2, 并且 nums 中的前两个元素均为 2。你不需要考虑数组中超出新长度后面的元素。例如，函数返回的新长度
```

示例 2：

```
输入：nums = [0,1,2,2,3,0,4,2], val = 2

输出：5, nums = [0,1,4,0,3]

解释：函数应该返回新的长度 5, 并且 nums 中的前五个元素为 0, 1, 3, 0, 4。注意这五个元素可为任意顺序。你不需要考虑数组中超出
```

提示：

- 0 <= nums.length <= 100
- 0 <= nums[i] <= 50
- 0 <= val <= 100

双指针解法

本解法的思路与 【题解】26. 删除排序数组中的重复项 中的「双指针解法」类似。

根据题意，我们可以将数组分成「前后」两段：

- 前半段是有效部分，存储的是不等于 `val` 的元素。
- 后半段是无效部分，存储的是等于 `val` 的元素。

最终答案返回有效部分的结尾下标。

代码：

```java
class Solution {
    public int removeElement(int[] nums, int val) {
        int j = nums.length - 1;
        for (int i = 0; i <= j; i++) {
            if (nums[i] == val) {
                swap(nums, i--, j--);
            }
        }
        return j + 1;
    }
    void swap(int[] nums, int i, int j) {
        int tmp = nums[i];
        nums[i] = nums[j];
        nums[j] = tmp;
    }
}
```

- 时间复杂度：$O(n)$
- 空间复杂度：$O(1)$

通用解法

本解法的思路与 【题解】26. 删除排序数组中的重复项 中的「通用解法」类似。

先设定变量 `idx`，指向待插入位置。`idx` 初始值为 0

然后从题目的「要求/保留逻辑」出发，来决定当遍历到任意元素 `x` 时，应该做何种决策：

- 如果当前元素 `x` 与移除元素 `val` 相同，那么跳过该元素。
- 如果当前元素 `x` 与移除元素 `val` 不同，那么我们将其放到下标 `idx` 的位置，并让 `idx` 自增右移。

最终得到的 `idx` 即是答案。

代码：

```java
class Solution {
    public int removeElement(int[] nums, int val) {
        int idx = 0;
        for (int x : nums) {
            if (x != val) nums[idx++] = x;
        }
        return idx;
    }
}
```

- 时间复杂度：$O(n)$
- 空间复杂度：$O(1)$

总结

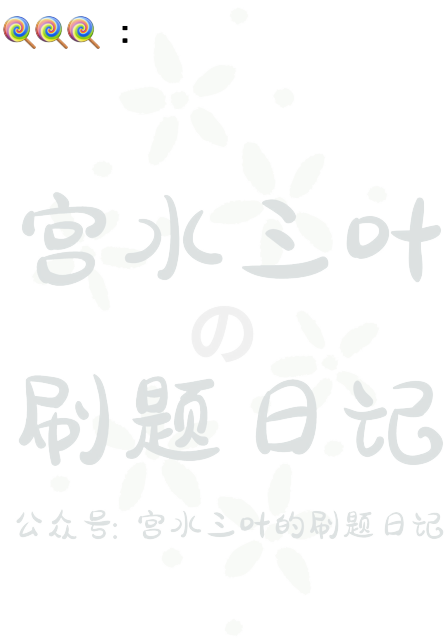
对于诸如「相同元素最多保留 `k` 位元素」或者「移除特定元素」的问题，更好的做法是从题目本身性质出发，利用题目给定的要求提炼出具体的「保留逻辑」，将「保留逻辑」应用到我们的遍历到的每一个位置。

> **🍭更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

💡更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。

最新专题合集资料下载，可关注公众号「宫水三叶的刷题日记」，回台回复「移除元素问题」获取下载链接。

觉得专题不错，可以请作者吃糖🍭🍭🍭 ：



"给作者手机充个电"

YOLO 的赞赏码