26. 删除有序数组中的重复项

地址：

删除有序数组中的重复项

题目：

- 中文

26. 删除有序数组中的重复项

难度 简单 2115 收藏 分享 切换为英文 接收动态 反馈

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

- English

26. Remove Duplicates from Sorted Array

难度 简单 2115 收藏 分享 切换为中文 接收动态 反馈

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates in-place such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the final result. It does not matter what you leave beyond the first `k` elements.

Return `k` *after placing the final result in the first* `k` *slots of* `nums`.

Do **not** allocate extra space for another array. You must do this by **modifying the input array** in-place with O(1) extra memory.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Example 1:

```
Input: nums = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the first two
elements of nums being 1 and 2 respectively.
It does not matter what you leave beyond the returned k (hence they are
underscores).
```

Example 2:

```
Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
Explanation: Your function should return k = 5, with the first five
elements of nums being 0, 1, 2, 3, and 4 respectively.
It does not matter what you leave beyond the returned k (hence they are
underscores).
```

Constraints:

- `0 <= nums.length <= 3 * 10^4`
- `-100 <= nums[i] <= 100`
- `nums` is sorted in **non-decreasing** order.

审题

- 原地修改；
- nums 数组是升序.

思路 1 ： 快慢指针

1. 创建 慢指针 slow，指向数组第一个元素；
2. 创建 快指针 fast，指向数组第二个元素；
3. 遍历数组,若 nums[fast] 与 nums[slow] 不相等:
    a. 将 slow 递增 1;
    b. 将不同的 nums[fast] 值 赋给 slow 递增1后的 nums[slow]；
4. 因为最初的 slow 等于 0 的元素未统计，返回无重复元素数组的长度 slow 需要 +1.

复杂度分析

- 时间复杂度：O(n)；

- 空间复杂度：O(1).

代码

```java
// Java
// Time : 2021 - 07 - 18

 public int removeDuplicates(int[] nums) {

        // 数组长度为 0，返回 0
        if (nums.length == 0)
            return 0;

        int slow = 0; // 慢指针指向数组第一个元素
         int fast = 1; // 快指针指向数组第二个元素

        // 循环找重复
        for (; fast < nums.length; fast++) {

            // 当快指针元素于慢指针元素不同时，将快指针元素复制到慢指针下一位
            if (nums[fast] != nums[slow]) {
                slow++;
                nums[slow] = nums[fast];
            }
        }

        // 返回无重复数组长度
        return slow + 1;
    }
```

#Leetcode/Array#