027 Remove Element

2018年3月29日 12:58

Question:

Given an array and a value, remove all instances of that value in-place and return the new length.
Do not allocate extra space for another array, you must do this by **modifying the input array in-place** with O(1) extra memory.
The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Example:

Given **nums** = [3,2,2,3], **val** = 3,
Your function should return length = 2, with the first two elements of *nums* being 2.
来自 <https://leetcode.com/problems/remove-element/description/>

给定一个数组和一个值，在这个数组中原地移除指定值和返回移除后新的数组长度。
不要为其他数组分配额外空间，你必须使用 O(1) 的额外内存原地修改这个输入数组。
元素的顺序可以改变。超过返回的新的数组长度以外的数据无论是什么都没关系。

示例:

给定 **nums** = [3,2,2,3]， **val** = 3，

你的函数应该返回 长度 = 2，数组的前两个元素是 2。

Solution for Python3:

```
class Solution:
    def removeElement(self, nums, val):
        """
        :type nums: List[int]
        :type val: int
        :rtype: int
        """
        n = 0
        for i in range(0, len(nums)):
            if nums[i] != val:
                nums[n] = nums[i]
                n += 1
        return n
Complexity analysis:
    Time complexity: O(n).Assume the array has a total of n elements,
                    both i and j traverse at most 2n steps.
    Space complexity: O(1)


class Solution:
    def removeElement(self, nums, val):
        """
        :type nums: List[int]
        :type val: int
        :rtype: int
        """
        i, n = 0, len(nums)
        while i < n:
            if nums[i] == val:
                nums[i] = nums[n - 1];
```

```
                #通过把数组最后元素前移替代当前要删除元素来减小数组规模
                n -= 1
            else:
                i += 1
        return n
Complexity analysis:
    Time complexity: O(n). Both i and n traverse at most n steps.
                    In this approach, the number of assignment operation
                    is equal to the number of elements to remove. So it is
                    more efficient if elements to remove are rare.
    Space complexity: O(1)
```

Solution for C++:

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int n = 0;
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] != val) {
                nums[n++] = nums[i];
            }
        }
        return n;
    }
};

class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
       int i = 0, n = nums.size();
        while(i < n)
           if(nums[i] == val) {
               nums[i] = nums[n - 1];
               n--;
           } else {
               i++;
           }
        return n;
    }
};
```

Appendix:每种版本的第二种方法更加高效，在遍历同时缩短数组长度。因为数组的顺序可以改变，所以可以把数组最后的元素提前来判断。