

12 移除数组中的指定元素

更新时间：2019-08-20 11:07:25



“

知识是一种快乐，而好奇则是知识的萌芽。

——培根

”

刷题内容

难度: **Easy**

原题链接: <https://leetcode-cn.com/problems/remove-element/>。

内容描述

给定一个数组 **nums** 和一个值 **val**，你需要原地移除所有数值等于 **val** 的元素，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。

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。

示例 1:

给定 **nums** = [3,2,2,3], **val** = 3,
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。

示例 2:

给定 **nums** = [0,1,2,2,3,0,4,2], **val** = 2,
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。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]);
}
```

题目详解

题目中给出了一个数组，我们要在原地删除所有值为 **val** 的元素，然后返回删除掉所有值为 **val** 的元素的数组长度。****注意：******题目同样要求要在原地进行删除，也就是说我们不能使用额外的空间。否则我们直接用一个新的数组来装所有旧数组中值不等于 **val** 的元素就行了。

解题方案

思路 1：时间复杂度： **$O(N^2)$** 空间复杂度： **$O(1)$**

其实我们可以非常暴力地看待这个问题。只要数组 **nums** 中还存在着值为 **val** 的元素，我们就把这个元素删除掉。

下面来看具体的代码：

Python beats 95.56%

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        while val in nums: # 只要值为val的元素还存在
            nums.remove(val)
        return len(nums)
```

c++ beats 80.41%

```

class Solution {
public:
    //模拟python的remove方法
    int removeIndex(vector<int>& nums, int len, int index) {
        for (int i = index + 1; i < len; i++) {
            nums[i - 1] = nums[i];
        }
        //python这里会调用一个resize的方法，把数组的长度变量修改成len-1，但数组实际上的长度还是不变
        return len - 1;
    }
    int removeElement(vector<int>& nums, int val) {
        int len = nums.size();
        while (1) {
            bool find = false;
            //再次羡慕python有in这个操作符
            for (int i = 0; i < len; i++) {
                if (nums[i] == val) {
                    len = removeIndex(nums, len, i);
                    find = true;
                    break;
                }
            }
            if (!find) {
                break;
            }
        }
        return len;
    }
};

```

Java beats 100%

```

class Solution {
    //模拟python的remove方法
    private int removeIndex(int[] nums, int len, int index) {
        for (int i = index + 1; i < len; i++) {
            nums[i - 1] = nums[i];
        }
        //python这里会调用一个resize的方法，把数组的长度变量修改成len-1，但数组实际上的长度还是不变
        return len - 1;
    }
    public int removeElement(int[] nums, int val) {
        int len = nums.length;
        while (true) {
            boolean find = false;
            for (int i = 0; i < len; i++) {
                if (nums[i] == val) {
                    len = removeIndex(nums, len, i);
                    find = true;
                    break;
                }
            }
            if (!find) {
                break;
            }
        }
        return len;
    }
}

```

go beats 100%

```
//模拟python的remove方法
func removeIndex(nums []int, ret int, index int) int {
    for i := index + 1; i < ret; i++ {
        nums[i - 1] = nums[i]
    }
    //python这里会调用一个resize的方法，把数组的长度变量修改成ret-1，但数组实际上的长度还是不变
    return ret - 1
}

func removeElement(nums []int, val int) int {
    ret := len(nums)
    for {
        find := false
        for i := 0; i < ret; i++ {
            if nums[i] == val {
                ret = removeIndex(nums, ret, i)
                find = true
                break
            }
        }
        if !find {
            break;
        }
    }
    return ret
}
```

但是这样会出现一个问题：那就是当数组 `nums` 中值为 `val` 的元素过多的时候，我们每次都要遍历整个数组去查看数组中是否还存在着值为 `val` 的元素，最坏的情况就是整个数组中都是值为 `val` 的元素，这样的话算法的时间复杂度就是 $O(N^2)$ 。既然我们知道问题的所在，那有什么办法可以优化一下吗？

思路 2：时间复杂度： $O(N)$ 空间复杂度： $O(1)$

其实我们可以使用一个小技巧：如果当前元素的值等于 `val`，就把当前元素的值换成数组的最后一个元素的值，然后删除掉数组的最后一个元素，这样就相当于把当前这个等于 `val` 的元素删除掉了，删除掉这个元素的同时我们也没有放弃数组的最后一个元素。这样做数组中的每个元素最多需要校验一次就可以，时间复杂度就是 $O(N)$ 了。

下面我们来看下具体的代码实现：

Python beats 100%

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        idx = 0
        while idx < len(nums):
            if nums[idx] == val:
                nums[idx] = nums[-1]
                del nums[-1]
            else:
                idx += 1
        return len(nums)
```

补充一下，开始我还以为 `nums = nums[:-1]` 和 `del nums[-1]` 都可以达到相同的效果，但是这道题的 `nums` 是通过引用传递，所以我们切片的时候 `nums` 地址变化了，这样是不对的，因此这里只能用 `del nums[-1]`。

c++ beats 100%

```

class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int len = 0;
        //把nums[0..len]当做新数组，不等于val的往里面插入
        for (int i = 0; i < nums.size(); i++) {
            if (nums[i] != val) {
                nums[len++] = nums[i];
            }
        }
        return len;
    }
};

```

java beats 100%

```

class Solution {
    public int removeElement(int[] nums, int val) {
        int len = 0;
        //把nums[0..len]当做新数组，不等于val的往里面插入
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != val) {
                nums[len++] = nums[i];
            }
        }
        return len;
    }
}

```

go beats 100%

```

func removeElement(nums []int, val int) int {
    ret := 0
    //把nums[0..ret]当做新数组，不等于val的往里面插入
    for i := 0; i < len(nums); i++ {
        if nums[i] != val {
            nums[ret] = nums[i]
            ret++
        }
    }
    return ret
}

```

这里也发现，思路 2 确实比思路 1 更快了一些。

小结

其实在学习算法的过程中，我们总是在一直总结各种模板和小技巧。一旦发现这样便于解题的小技巧，就在自己的小本本上记录下来。以后碰上类似题目的时候，就可以直接用上了。

}