欢迎大家参与本项目，贡献其他语言版本的代码，拥抱开源，让更多学习算法的小伙伴们收益！

27. 移除元素

题目地址：https://leetcode-cn.com/problems/remove-element/

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。

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。

你不需要考虑数组中超出新长度后面的元素。

思路

有的同学可能说了，多余的元素，删掉不就得了。

要知道数组的元素在内存地址中是连续的，不能单独删除数组中的某个元素，只能覆盖。

数组的基础知识可以看这里程序员算法面试中，必须掌握的数组理论知识。

暴力解法

这个题目暴力的解法就是两层for循环，一个for循环遍历数组元素 ，第二个for循环更新数组。

删除过程如下：

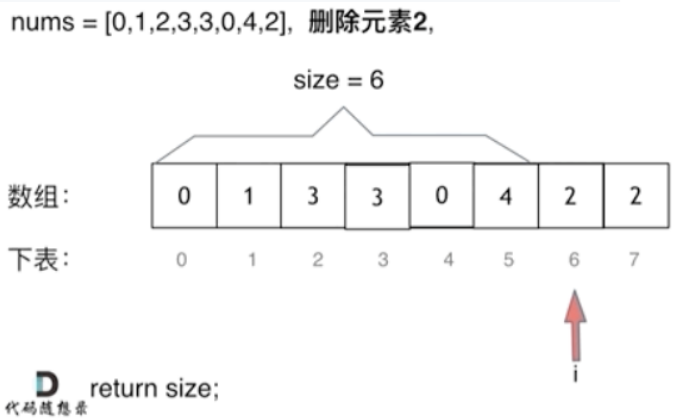
很明显暴力解法的时间复杂度是O(n^2)，这道题目暴力解法在leetcode上是可以过的。

代码如下：

```
// 时间复杂度：O(n^2)
// 空间复杂度：O(1)
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int size = nums.size();
        for (int i = 0; i < size; i++) {
            if (nums[i] == val) { // 发现需要移除的元素，就将数组集体向前移动一位
                for (int j = i + 1; j < size; j++) {
                    nums[j - 1] = nums[j];
                }
                i--; // 因为下表i以后的数值都向前移动了一位，所以i也向前移动一位
                size--; // 此时数组的大小-1
            }
        }
        return size;

    }
};
```

之后i++又回到上次的i的位置



- 时间复杂度：$O(n^2)$
- 空间复杂度：$O(1)$

双指针法

双指针法（快慢指针法）：**通过一个快指针和慢指针在一个for循环下完成两个for循环的工作。**

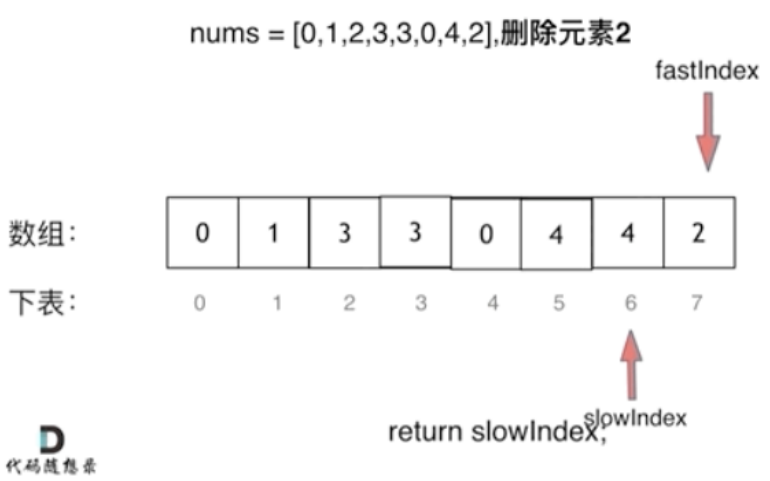
删除过程如下：

双指针法（快慢指针法）在数组和链表的操作中是非常常见的，很多考察数组、链表、字符串等操作的面试题，都使用双指针法。

后序都会一一介绍到，本题代码如下：

```
// 时间复杂度：O(n)
// 空间复杂度：O(1)
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int slowIndex = 0;
```

```
        for (int fastIndex = 0; fastIndex < nums.size(); fastIndex++) {
            if (val != nums[fastIndex]) {
                nums[slowIndex++] = nums[fastIndex];
            }
        }
        return slowIndex;
    }
};
```



注意这些实现方法并没有改变元素的相对位置!

- 时间复杂度：$O(n)$
- 空间复杂度：$O(1)$

旧文链接：数组：就移除个元素很难么？

相关题目推荐

- 26.删除排序数组中的重复项
- 283.移动零
- 844.比较含退格的字符串
- 977.有序数组的平方

其他语言版本

Java：

```
class Solution {
    public int removeElement(int[] nums, int val) {

        // 快慢指针
        int fastIndex = 0;
        int slowIndex;
        for (slowIndex = 0; fastIndex < nums.length; fastIndex++) {
            if (nums[fastIndex] != val) {
                nums[slowIndex] = nums[fastIndex];
                slowIndex++;
            }
        }
        return slowIndex;

    }
}
```

Python：

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        i,n = 0,len(nums)
        for j in range(n):
```

```
            if nums[j] != val:
                nums[i] = nums[j]
                i += 1
        return i
```

Go:

```
func removeElement(nums []int, val int) int {
    length:=len(nums)
    res:=0
    for i:=0;i<length;i++{
        if nums[i]!=val {
            nums[res]=nums[i]
            res++
        }
    }
    return res
}
```

JavaScript:

```
//时间复杂度O(n)
//空间复杂度O(1)
var removeElement = (nums, val) => {
    let k = 0;
    for(let i = 0;i < nums.length;i++){
        if(nums[i] != val){
            nums[k++] = nums[i]
        }
    }
    return k;
};
```

Ruby:

```
def remove_element(nums, val)
    i = 0
    nums.each_index do |j|
        if nums[j] != val
            nums[i] = nums[j]
            i+=1
        end
    end
    i
end
```

Rust:

```
pub fn remove_element(nums: &mut Vec<i32>, val: i32) -> &mut Vec<i32> {
    let mut start: usize = 0;
    while start < nums.len() {
        if nums[start] == val {
```

```
                nums.remove(start);
            }
            start += 1;
        }
        nums
    }
    fn main() {
        let mut nums = vec![5,1,3,5,2,3,4,1];
        println!("{:?}",remove_element(&mut nums, 5));
    }
```

- 作者微信：程序员Carl
- B站视频：代码随想录
- 知识星球：代码随想录