11 删除排序数组中的重复项

更新时间：2019-08-19 09:57:03

> 人的影响短暂而微弱，书的影响则广泛而深远。
>
> ——普希金

刷题内容

> 难度：**Easy**
>
> 原题链接：https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array/。

给定一个排序数组，你需要在原地删除重复出现的元素，使得每个元素只出现一次，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。

示例 1:
给定数组 nums = [1,1,2],
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。
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 = [0,0,1,1,1,2,2,3,3,4],
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。
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]);
}
```

在这里我举一个生活中的例子：假如我们有一场作文评选大赛，每一位作者可以提交数篇自己的作品到举办方，举办方要给位于前100名获奖作品的作者颁发一份奖品。但是每位作者限领一份奖品，那如果一个作者有两篇作品入围了怎么办呢？这样的话我们就需要移除重复的作者。

从上面的例子我们可以知道， 100 篇获奖作品的作者是一个已经排好序的数组，每一个作者都是数组中的一个元素。

我们希望in-place地移除这个数组中所有的重复元素，即在原数组上直接做完这个操作，而不是另外创建一个新数组，然后放入所有的不重复作者，我们应该怎么做呢？

> 因为已经说了是排好序的数组，所以只需要不停判断当前位置值和下一位置值是否相等。 若相等则`pop`掉当前值，否则`move`到下一位置继续做判断。

下面我们来看下具体的代码实现：

python版本

```python
class Solution(object):
    def removeDuplicates(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        idx = 0
        while idx < len(nums) - 1:
            if nums[idx] == nums[idx+1]: # 若相等则pop掉当前值
                nums.pop(idx)
            else: # 否则move到下一位置继续做判断
                idx += 1
        return len(nums)
```

Java版本

```java
class Solution {
    public int removeDuplicates(int[] nums) {
        // len 初始化为 nums 的长度，当发现重复的元素则会递减
        int len = nums.length;
        int i = 0;
        while (i < len - 1) {
            if (nums[i] == nums[i + 1]) {
                for (int j = i; j < len - 1; j++) {
                    // 发现重复的元素则会将后面的元素依次前移一位
                    nums[j] = nums[j + 1];
                }
                len--;
            }else {
                i++;
            }
        }
        return len;
    }
}
```

Go版本

```go
func removeDuplicates(nums []int) int {
    idx := 0
    for idx < len(nums) - 1 {
        if nums[idx] == nums[idx+1] {
            nums = append(nums[:idx], nums[idx+1:]...) // 若相等则pop掉当前值
        } else { // 否则move到下一位置继续做判断
            idx += 1
        }
    }
    return len(nums)
}
```

但是上面的方式有一个缺陷：我们的时间复杂度是 O(N^2)，因为每一次pop操作我们都需要将被删除元素后面的所有元素向前移动一格。那么该怎么优化一下这个算法呢？

其实我们只需要不停地向下遍历，同时自增地分配不重复的值给前面的位置即可。

python版本

```python
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        idx = 0
        for num in nums:
            # 如果是第一位自然不可能重复（同时为了保证数组不越界）
            if idx < 1 or num != nums[idx-1]:
                nums[idx] = num # 如果当前元素不是重复值，我们就将其分配到目前不重复元素到达的index
                idx += 1
        return idx
```

Java版本

```java
class Solution {
    public int removeDuplicates(int[] nums) {
        // 第一个数肯定不是重复的数，并且作为一个指针来计算不重复的数的个数
        int i = 0;
        for (int j = 1; j < nums.length; j++) {
            // 如果不是重复的数，就将指针 i 后移,同时将不重复的元素分配到指针 i 目前到达的位置
            if (nums[j] != nums[i]) {
                i++;
                nums[i] = nums[j];
            }
        }
        // i+1 是因为需要加上第一个不重复的数
        return i + 1;
    }
}
```

Go版本

```go
func removeDuplicates(nums []int) int {
    idx := 0
    for _, num := range nums {
        // 如果是第一位自然不可能重复（同时为了保证数组不越界）
        if (idx < 1) || (num != nums[idx-1]) {
            nums[idx] = num // 如果当前元素不是重复值，我们就将其分配到目前不重复元素到达的index
            idx += 1
        }
    }
    return idx
}
```

这样的话我们的时间复杂度就是O(N)了。

那么继续深入问一下，如果我们允许一个作品作者最多可以得到两份奖品，我们该怎么办呢？毫无疑问，此时我们就可以允许有两个重复值，超过两次重复就不行了。

那么我们只要稍微改一下我们的代码就可以了(其实就是1改成2)，见下方：

python版本

```python
class Solution:
    def removeDuplicates(self, nums: List[int]) -> int:
        idx = 0
        for num in nums:
            # 如果是前两位自然不可能重复两次以上（同时为了保证数组不越界）
            if idx < 2 or num != nums[idx-2]:
                nums[idx] = num # 如果当前元素没有重复两次以上，我们就将其assign到目前不重复元素到达的index
                idx += 1
        return idx
```

Java版本

```
class Solution {
    public int removeDuplicates(int[] nums) {
        // 设置一个指针 i 来标志满足要求的元素个数
        int i = 0;
        for (int j = 0; j < nums.length; j++) {
            // 前两个元素肯定不会重复两次以上
            if (i < 2 || nums[j] != nums[i - 2]) {
                nums[i] = nums[j];
                i++;
            }
        }
        return i;
    }
}
```

Go版本

```
func removeDuplicates(nums []int) int {
    idx := 0
    for _, num := range nums {
        // 如果是前两位自然不可能重复两次以上（同时为了保证数组不越界）
        if (idx < 2) || (num != nums[idx-2]) {
            nums[idx] = num // 如果当前元素没有重复两次以上，我们就将其assign到目前不重复元素到达的index
            idx += 1
        }
    }
    return idx
}
```

上面的问题对应 LeetCode 第26题，LeetCode 第80题。

从排序链表中移除重复元素

相信看完从排序数组中移除重复元素，大家也想看看如果换成排序链表的话会有什么不同，下面就让我们来看一看：

python版本

```
class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        dummy = head
        while head:
            while head.next and head.next.val == head.val:
                head.next = head.next.next  # 删掉重复元素
            head = head.next    # 当前元素没重复，进行下一次判断
        return dummy
```

Java版本

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode dummy = head;
        while (head != null) {
            // 当下一个元素不为空，并且当前元素的值与下一个元素的值相等
            while (head.next != null && head.val == head.next.val) {
                // 则将当前元素指向下一个元素的指针指向下一个元素的再下一个元素
                head.next = head.next.next;
            }
            // 当前元素没重复，进行下一次判断
            head = head.next;
        }
        return dummy;
    }
}
```

Go版本

```
func deleteDuplicates(head *ListNode) *ListNode {
    dummy := head
    for head != nil {
        for head.Next != nil && head.Next.Val == head.Val {
            head.Next = head.Next.Next // 删掉重复元素
        }
        head = head.Next // 当前元素没重复，进行下一次判断
    }
    return dummy
}
```

相信大家也看到了，这里我们巧妙运用了一个`dummy`节点来做我们的操作，这个方法可以很有效地避免一些边界问题，因此希望大家能够养成一个习惯，只要做到链表类的问题，就可以无脑使用一个`dummy`节点。

现在又来了一个新问题，如果我们想让最终的链表中只留下之前没有重复的元素，该怎么做？

这个时候我们就要用到一个`prev`指针指向前面的元素，方便我们进行比较啦。

python版本

```
class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        dummy = prev = cur = ListNode(None)
        while head:
            # 如果当前是元素是重复元素
            while head and ((head.val == prev.val) or (head.next and head.next.val == head.val)):
                prev = head
                head = head.next # 不停跳过重复元素
            cur.next = head # cur永远指向我们的不重复元素
            cur = cur.next
            if head: # head也要往后走，但是注意head可能已经走到None了，因此要判断一下
                head = head.next

        return dummy.next
```

Java版本

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }
        // dummy 指向不重复的 head 结点
        ListNode dummy  = new ListNode(0);
        // cur 指向当前结点
        ListNode cur  = dummy;
        // prev 指向前一个结点
        ListNode prev  = null;
        while (head != null) {
            if ((head.next == null || head.val != head.next.val) && (prev == null || prev.val != head.val)){
                cur.next = head;
                cur = cur.next;
            }
            prev = head;
            head = head.next;
        }
        cur.next = null;
        return dummy.next;
    }
}
```

Go版本

```
func deleteDuplicates(head *ListNode) *ListNode {
    dummy := &ListNode{Val: math.MaxInt32+1} // 这里是为了不与后面的元素重复
    prev, cur := dummy, dummy
    for head != nil {
        // 如果当前是元素是重复元素
        for head != nil && ((head.Val == prev.Val) || (head.Next != nil && head.Next.Val == head.Val)) {
            prev = head
            head = head.Next // 不停跳过重复元素
        }
        cur.Next = head // cur永远指向我们的不重复元素
        cur = cur.Next
        if head != nil { // head也要往后走，但是注意head可能已经走到None了，因此要判断一下
            head = head.Next
        }
    }
    return dummy.Next
}
```

上面的问题对应 LeetCode 第83题，LeetCode 第82题。

小结

1. 解决完一个问题之后，我们还要多问自己几个问题来加强认识。比如我们已经知道，怎么处理从数组中移除重复元素的时候，我们可以问下自己如果允许出现一次重复呢？ 再问自己如果换成链表呢？有一个科学实验证明，重复一样的操作不如稍微改变一点的训练。因此我们做完任务后稍微改变一下再做一次，这样记得更牢，理解也更深；
2. 遇到一个问题的时候，我们要善于将它们归纳关联到已知的知识上。比如问你链表中移除重复元素怎么处理，我们就要往我们已知的领域，即数组移除重复元素上靠；
3. 善于总结自己的模版，同时牢记自己遇到过的坑。比如对于数组问题，我们需要注意越界问题；对于链表问题，加一个`dummy`，对我们的处理会更加方便。

}

10 合并两个有序链表

12 移除数组中的指定元素