如何去除有序数组的重复元素

我们知道对于数组来说，在尾部插入、删除元素是比较高效的，时间复杂度是 O(1)，但是如果在中间或者开头插入、删除元素，就会涉及数据的搬移，时间复杂度为 O(N)，效率较低。

所以对于一般处理数组的算法问题，我们要尽可能只对数组尾部的元素进行操作，以避免额外的时间复杂度。

这篇文章讲讲如何对一个有序数组去重，先看下题目：

> 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。
> ```

显然，由于数组已经排序，所以重复的元素一定连在一起，找出它们并不难，但如果每找到一个重复元素就立即删除它，就是在数组中间进行删除操作，整个时间复杂度是会达到 O(N^2)。而且题目要求我们原地修改，也就是说不能用辅助数组，空间复杂度得是 O(1)。

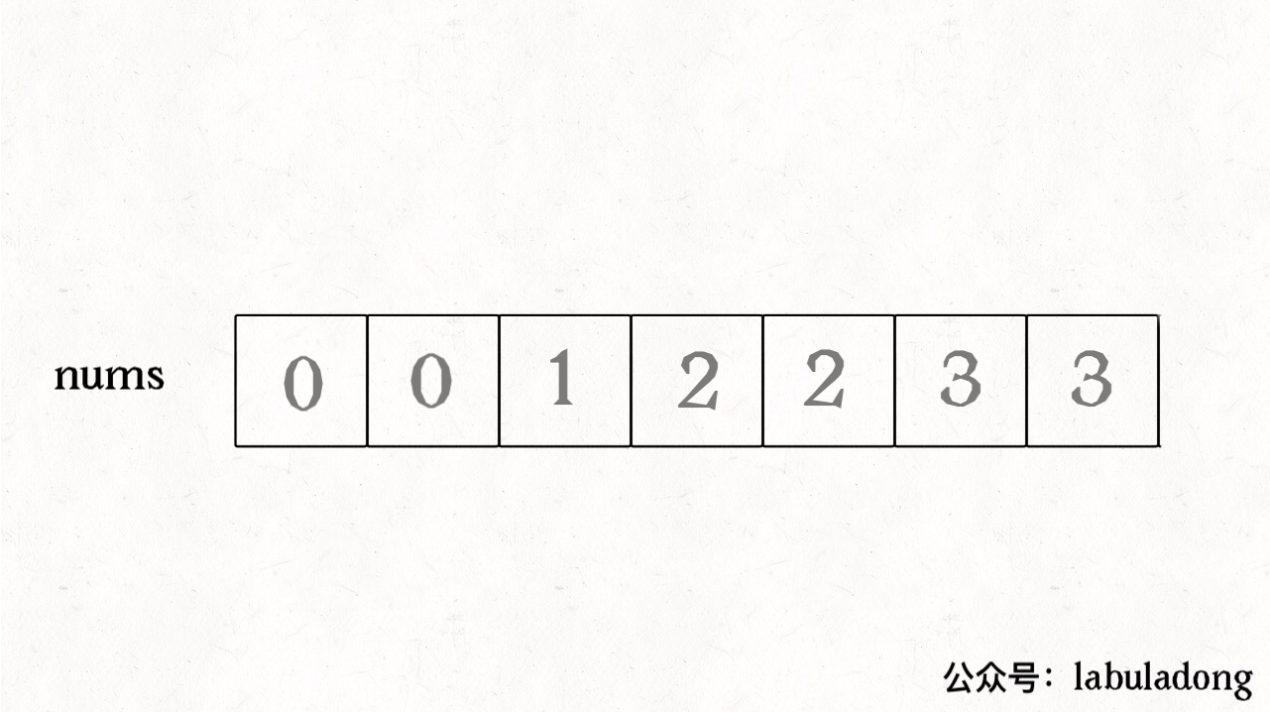
其实，**对于数组相关的算法问题，有一个通用的技巧：要尽量避免在中间删除元素，那我就先想办法把这个元素换到最后去**。这样的话，最终待删除的元素都拖在数组尾部，一个一个 pop 掉就行了，每次操作的时间复杂度也就降低到 O(1) 了。

按照这个思路呢，又可以衍生出解决类似需求的通用方式：双指针技巧。具体一点说，应该是快慢指针。

我们让慢指针 `slow` 走左后面，快指针 `fast` 走在前面探路，找到一个不重复的元素就告诉 `slow` 并让 `slow` 前进一步。这样当 `fast` 指针遍历完整个数组 `nums` 后，**`nums[0..slow]` 就是不重复元素，之后的所有元素都是重复元素**。

```
int removeDuplicates(int[] nums) {
    int n = nums.length;
    if (n == 0) return 0;
    int slow = 0, fast = 1;
    while (fast < n) {
        if (nums[fast] != nums[slow]) {
            slow++;
            // 维护 nums[0..slow] 无重复
            nums[slow] = nums[fast];
        }
        fast++;
    }
    // 长度为索引 + 1
    return slow + 1;
}
```

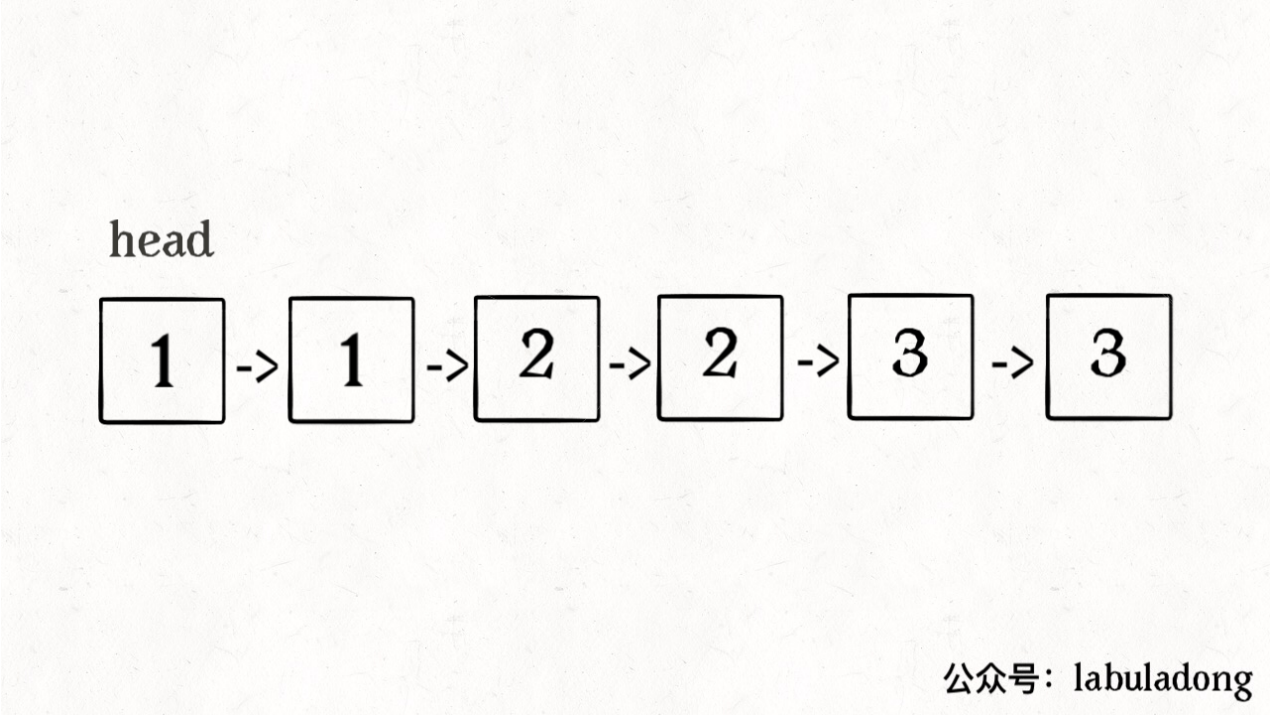
看下算法执行的过程：



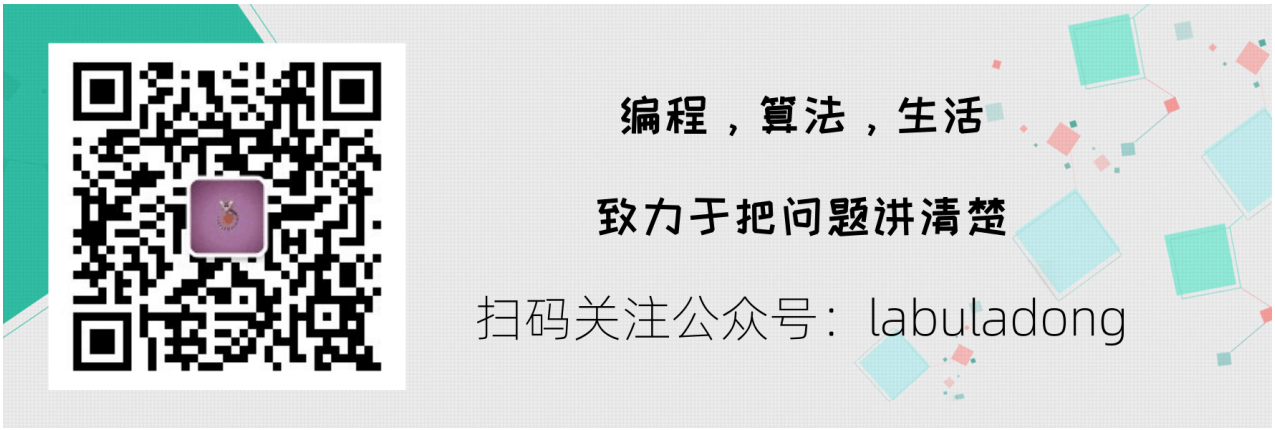
再简单扩展一下，如果给你一个有序链表，如何去重呢？其实和数组是一模一样的，唯一的区别是把数组赋值操作变成操作指针而已：

```
ListNode deleteDuplicates(ListNode head) {
    if (head == null) return null;
    ListNode slow = head, fast = head.next;
    while (fast != null) {
        if (fast.val != slow.val) {
            // nums[slow] = nums[fast];
            slow.next = fast;
            // slow++;
            slow = slow.next;
```

```
            }
            // fast++
            fast = fast.next;
        }
        // 断开与后面重复元素的连接
        slow.next = null;
        return head;
    }
```



致力于把算法讲清楚！欢迎关注我的微信公众号 labuladong，查看更多通俗易懂的文章：



[eric wang](#) 提供有序数组 Python3 代码

```
def removeDuplicates(self, nums: List[int]) -> int:
    n = len(nums)

    if n == 0:
        return 0

    slow, fast = 0, 1
```

```
    while fast < n:
        if nums[fast] != nums[slow]:
            slow += 1
            nums[slow] = nums[fast]

        fast += 1

    return slow + 1
```

eric wang 提供有序链表 Python3 代码

```
def deleteDuplicates(self, head: ListNode) -> ListNode:
    if not head:
        return head

    slow, fast = head, head.next

    while fast:
        if fast.val != slow.val:
            slow.next = fast
            slow = slow.next

        fast = fast.next

    # 断开与后面重复元素的连接
    slow.next = None
    return head
```

==其他语言代码==