

卡码笔记-范泛

代码随想录

数组

《代码随想录》数组：二分查找

LeetCode 704. 二分查找

题目描述 给定一个 n 个元素有序的（升序）整型数组 `nums` 和一个目标值 `target`，写一个函数搜索 `nums` 中的 `target`，如果 `target` 存在返回下标，否则返回 `-1`。

提示：

1. 你可以假设 `nums` 中的所有元素是不重复的。
2. n 将在 $[1, 10000]$ 之间。
3. `nums` 的每个元素都将在 $[-9999, 9999]$ 之间。

示例 1:

```
输入: nums = [-1,0,3,5,9,12], target = 9
输出: 4
解释: 9 出现在 nums 中并且下标为 4
```

示例 2:

```
输入: nums = [-1,0,3,5,9,12], target = 2
输出: -1
解释: 2 不存在 nums 中因此返回 -1
```

核心思路：左闭右闭区间

二分查找（Binary Search）的核心思想是“分而治之”。因为数组是有序的，我们通过比较数组中间的元素与目标值，可以一次性排除掉一半的搜索范围。算法步骤：

1. 定义查找范围 $[left, right]$ ，初始 $left = 0$, $right = \text{len}(\text{nums}) - 1$ 。
2. 当 $left \leq right$ 时（注意这里包含 $=$ ，因为区间内最后一个元素也需要比较）：
 - 计算中间下标 mid 。
 - 如果 $\text{nums}[mid] == \text{target}$ ，查找成功，返回 mid 。
 - 如果 $\text{nums}[mid] > \text{target}$ ，说明目标在左侧，更新 $right = mid - 1$ 。
 - 如果 $\text{nums}[mid] < \text{target}$ ，说明目标在右侧，更新 $left = mid + 1$ 。
3. 循环结束仍未找到，返回 `-1`。

代码实现 (Python)

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left = 0
        right = len(nums) - 1

        # 必须是 <=, 因为我们要处理 left == right 的情况
        while left <= right:
            # 防止溢出的写法 (Python自动处理大数, 但在C++/Java中很重要)
            # mid = left + (right - left) // 2
            mid = (left + right) // 2

            if nums[mid] == target:
                return mid
            elif nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1

        return -1
```

复杂度分析：为什么是 $O(\log n)$ ？

二分查找的时间复杂度是 $O(\log n)$ ，这是因为每次比较都将搜索范围 减半。 数学推导： 假设数组长度为 n 。

- 第 1 次查找：剩余元素 $\frac{n}{2}$
- 第 2 次查找：剩余元素 $\frac{n}{4}$
- ...
- 第 k 次查找：剩余元素 $\frac{n}{2^k}$

最坏的情况是直到剩余 1 个元素才找到（或没找到）。我们需要求解 k 的值，使得：

$$\frac{n}{2^k} = 1$$

即：

$$2^k = n$$

对两边取对数：

$k = \log_2 n$ 在算法复杂度分析中，底数通常省略，记作 $O(\log n)$ 。具体增长量级：

- $n = 100$: $\log_2 100 \approx 6.64$ (约 7 次)
- $n = 10,000$: $\log_2 10000 \approx 13.28$ (约 14 次)
- $n = 1,000,000$: $\log_2 10^6 \approx 20$ (约 20 次)

可以看到，随着数据规模 n 的爆炸式增长，查找次数 k 的增长非常缓慢。

《代码随想录》数组：移除元素

Leetcode 27 移除元素

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。元素的顺序可能发生改变。然后返回 `nums` 中与 `val` 不同的元素的数量。假设 `nums` 中不等于 `val` 的元素数量为 `k`，要通过此题，您需要执行以下操作：

- 更改 `nums` 数组，使 `nums` 的前 `k` 个元素包含不等于 `val` 的元素。`nums` 的其余元素和 `nums` 的大小并不重要。
- 返回 `k`。

用户评测：评测机将使用以下代码测试您的解决方案：

```
int[] nums = [...]; // 输入数组
int val = ...; // 要移除的值
int[] expectedNums = [...]; // 长度正确的预期答案。
// 它以不等于 val 的值排序。

int k = removeElement(nums, val); // 调用你的实现

assert k == expectedNums.length;
sort(nums, 0, k); // 排序 nums 的前 k 个元素
for (int i = 0; i < actualLength; i++) {
    assert nums[i] == expectedNums[i];
}
```

如果所有的断言都通过，你的解决方案将会通过。

示例 1：

```
输入：nums = [3,2,2,3], val = 3
输出：2, nums = [2,2,_,_]
解释：你的函数应该返回 k = 2, 并且 nums 中的前两个元素均为 2。
你在返回的 k 个元素之外留下了什么并不重要（因此它们并不计入评测）。
```

示例 2：

```
输入：nums = [0,1,2,2,3,0,4,2], val = 2
输出：5, nums = [0,1,4,0,3,_,_,_]
解释：你的函数应该返回 k = 5, 并且 nums 中的前五个元素为 0,0,1,3,4。
注意这五个元素可以任意顺序返回。
你在返回的 k 个元素之外留下了什么并不重要（因此它们并不计入评测）。
```

提示：

- $0 \leq \text{nums.length} \leq 100$
- $0 \leq \text{nums}[i] \leq 50$
- $0 \leq \text{val} \leq 100$

核心思路：快慢双指针

由于题目要求原地删除，我们不能新建数组。数组在内存中是连续的，删除中间的元素往往需要移动后续所有

元素，导致 $O(n^2)$ 的复杂度。利用双指针法可以将时间复杂度优化到 $O(n)$ 。指针定义：

- 快指针 (fast)：寻找新数组的元素（即不包含 val 的元素）。
- 慢指针 (slow)：指向新数组下标的位置。

算法流程：

- fast 和 slow 都从 0 开始。
- fast 遍历数组：
 - 如果 `nums[fast] != val`：说明这是个“好元素”，把它填入 `nums[slow]` 的位置，同时 slow 加 1。
 - 如果 `nums[fast] == val`：说明这是“坏元素”，跳过它（不做任何赋值），slow 保持不动。
- 循环结束后，slow 的值就是新数组的长度。

图解演示 假设 `nums = [0, 1, 2, 2, 3, 0]`, `val = 2`

步骤	fast 指向的值	是否等于 val?	操作	数组状态 (前 slow 位)	slow	fast
初始	0	否	赋值 <code>nums[0]=0</code>	<code>[0, ...]</code>	1	1
2	1	否	赋值 <code>nums[1]=1</code>	<code>[0, 1, ...]</code>	2	2
3	2	是	跳过	<code>[0, 1, ...]</code>	2	3
4	2	是	跳过	<code>[0, 1, ...]</code>	2	4
5	3	否	赋值 <code>nums[2]=3</code>	<code>[0, 1, 3, ...]</code>	3	5
6	0	否	赋值 <code>nums[3]=0</code>	<code>[0, 1, 3, 0, ...]</code>	4	6

最终返回 `slow = 4`，数组前 4 位为 `[0, 1, 3, 0]`。

代码实现 (Python)

Python

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        slow = 0
        fast = 0

        while fast < len(nums):
            # 只有当 fast 指向的值不是目标值时，才进行收集
            if nums[fast] != val:
                nums[slow] = nums[fast]
                slow += 1
            # fast 无论如何都要一直向前探索
            fast += 1

        return slow
```

复杂度分析

- 时间复杂度: $O(n)$
 - fast 指针遍历数组一次, slow 指针最多遍历一次。总共有 n 个元素, 操作次数为线性。
- 空间复杂度: $O(1)$
 - 我们没有使用额外的数组空间, 只是利用了几个变量进行指针操作。

《代码随想录》数组：有序数组的平方

有序数组的平方

给你一个按 非递减顺序 排序的整数数组 `nums`, 返回 每个数字的平方 组成的新数组, 要求也按 非递减顺序 排序。

示例 1:

```
输入: nums = [-4,-1,0,3,10]
输出: [0,1,9,16,100]
解释: 平方后, 数组变为 [16,1,0,9,100]
排序后, 数组变为 [0,1,9,16,100]
```**示例 2: **
```

输入: `nums = [-7,-3,2,3,11]` 输出: `[4,9,9,49,121]`

```
**提示: **
- `1 <= nums.length <= 104`
- `-104 <= nums[i] <= 104`
- `nums` 已按 **非递减顺序** 排序
```

进阶:

- 请你设计时间复杂度为  $O(n)$  的算法解决本问题

有两种简单方法

写法一 Python3 Java C++ C Go JavaScript Rust class Solution: def sortedSquares(self, nums: List[int]) -> List[int]: n = len(nums) ans = [0] \* n i, j = 0, n - 1 for p in range(n - 1, -1, -1): x = nums[i] \* nums[i] y = nums[j] \* nums[j] if x > y: # 更大的数放右边 ans[p] = x i += 1 else: ans[p] = y j -= 1 return ans 写法二 改成比较  $-nums[i]$  和  $nums[j]$  的大小:

如果  $-nums[i] > nums[j]$ , 那么填入  $ans[p] = nums[i]^2$ 。如果  $-nums[i] \leq nums[j]$ , 那么填入  $ans[p] = nums[j]^2$ 。为什么这样做是对的?

如果 `nums` 中的元素均为负数, 那么  $-nums[i] > 0 > nums[j]$  恒成立, 我们按照 (平方后) 从大到小的顺序填入答案。如果 `nums` 中的元素均为非负数, 那么  $-nums[i] \leq 0 \leq nums[j]$  恒成立, 我们按照 (平方后) 从大到小的顺序填入答案。否则,  $nums[i] < 0$  且  $nums[j] \geq 0$ , 那么  $-nums[i] > nums[j]$  等价于  $nums[i]^2 > nums[j]^2$

$nums[j]^2$ , 反之亦然。填入答案, 移动指针后, 转换成这三种情况之一。所以我们仍然按照 (平方后) 从大到小的顺序填入答案。注: 这种写法每次循环只需要计算一次乘法。

Python3 Java C++ C Go JavaScript Rust class Solution: def sortedSquares(self, nums: List[int]) ->

```
List[int]: n = len(nums) ans = [0] * n i, j = 0, n - 1 for p in range(n - 1, -1, -1): x, y = nums[i], nums[j] if -x > y: ans[p] = x * x i += 1 else: ans[p] = y * y j -= 1 return ans
```

复杂度分析 时间复杂度：O(n)，其中 n 是 nums 的长度。空间复杂度：O(1)。返回值不计入。

作者：灵茶山艾府

链接：<https://leetcode.cn/problems/squares-of-a-sorted-array/solutions/2806253/xiang-xiang-shuang-zhi-zhen-cong-da-dao-bld6/>

来源：力扣（LeetCode） 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。