704. 二分查找

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`，如果目标值存在返回下标，否则返回 `-1`。

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
```

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。

In [10]:
```python
from typing import List

class Solution:
    # 我的尝试1：for 循环所有
    def search_for_all(self, nums: List[int], target: int) -> int:
        output = -1

        # search target in nums, if target exists, then return its index
        if target in nums:
            output = 0
            for num in nums:
                if num == target:
                    return output
                else:
                    output += 1

        # Not in nums, return -1
        else:
            return output


    # 我的尝试2：利用二分查找，降低复杂度
    # 每次取查找范围的中点
    def search(self, nums: List[int], target: int) -> int:
        output = -1
        left = 0
        right = len(nums)
        if target in nums:
            while left <= right:
                mid_i = left + (right - left) // 2
                mid_num = nums[mid_i]
                if mid_num == target:
                    return mid_i
                elif mid_num > target:
                    right = mid_i - 1
                elif mid_num < target:
                    left = mid_i + 1

        # Not in nums, return -1
        else:
            return output


    # 去掉target in nums的loop

    def search(self, nums: List[int], target: int) -> int:
        left = 0
        right = len(nums) - 1  # 修正：索引范围是 [0, len(nums) - 1]
        while left <= right:
            mid_i = left + (right - left) // 2
            mid_num = nums[mid_i]
            if mid_num == target:
                return mid_i  # 找到目标值，直接返回索引
            elif mid_num > target:
                right = mid_i - 1  # 目标值在左侧，缩小右边界
            else:
                left = mid_i + 1  # 目标值在右侧，缩小左边界

        return -1  # 未找到目标值，返回 -1
```

In [11]:
```python
def run_tests():
    solution = Solution()  # 创建类的实例

    test_cases = [
```

```
        {"nums": [-1, 0, 3, 5, 9, 12], "target": 9, "expected": 4},
        {"nums": [-1, 0, 3, 5, 9, 12], "target": 2, "expected": -1}
    ]

    # 测试 search 方法
    print("Testing search method:")
    for i, test in enumerate(test_cases):
        nums = test["nums"]
        target = test["target"]
        expected = test["expected"]
        result = solution.search(nums, target)
        print(f"Test case {i + 1}: {'PASSED' if result == expected else 'FAILED'}")
        print(f"Input: nums={nums}, target={target}")
        print(f"Expected Output: {expected}, Actual Output: {result}\n")

    # 测试 search_for_all 方法
    print("Testing search_for_all method:")
    for i, test in enumerate(test_cases):
        nums = test["nums"]
        target = test["target"]
        expected = test["expected"]
        result = solution.search_for_all(nums, target)
        print(f"Test case {i + 1}: {'PASSED' if result == expected else 'FAILED'}")
        print(f"Input: nums={nums}, target={target}")
        print(f"Expected Output: {expected}, Actual Output: {result}\n")

# 运行测试
run_tests()
```

```
Testing search method:
Test case 1: PASSED
Input: nums=[-1, 0, 3, 5, 9, 12], target=9
Expected Output: 4, Actual Output: 4

Test case 2: PASSED
Input: nums=[-1, 0, 3, 5, 9, 12], target=2
Expected Output: -1, Actual Output: -1

Testing search_for_all method:
Test case 1: PASSED
Input: nums=[-1, 0, 3, 5, 9, 12], target=9
Expected Output: 4, Actual Output: 4

Test case 2: PASSED
Input: nums=[-1, 0, 3, 5, 9, 12], target=2
Expected Output: -1, Actual Output: -1
```

703反思

二分查找法： - 直接把边界定义为两个变量 - 更新边界 - 更新的时候跳过中间元素 left/right = mid_i +/- 1 - 考虑左闭右闭更直接

---

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 <= nums.length <= 100 0 <= nums[i] <= 50 0 <= val <= 100

Problem: 27. Remove Element

Problem Statement

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. The order of the elements may be changed. Then return the number of elements in `nums` which are not equal to `val`.

Consider the number of elements in `nums` which are not equal to `val` be `k`. To get accepted, you need to do the following things:

1. Change the array `nums` such that the **first `k` elements** of `nums` contain the elements which are not equal to `val`.
2. The remaining elements of `nums` are not important, as well as the size of `nums`.
3. Return `k`.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int val = ...; // Value to remove
int[] expectedNums = [...]; // The expected answer with correct length.
// It is sorted with no values equaling val.

int k = removeElement(nums, val); // Calls your implementation

assert k == expectedNums.length;
sort(nums, 0, k); // Sort the first k elements of nums
for (int i = 0; i < actualLength; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be accepted.

Examples

Example 1:

Input:

```
plaintext
nums = [3,2,2,3], val = 3
```

Output:

```
plaintext
2, nums = [2,2,_,_]
```

Explanation:
Your function should return `k = 2`, with the first two elements of `nums` being `2`. It does not matter what you leave beyond the returned `k` (hence they are underscores).

Example 2:

Input:

```
plaintext
nums = [0,1,2,2,3,0,4,2], val = 2
```

Output:

```
plaintext
5, nums = [0,1,4,0,3,_,_,_]
```

Explanation:
Your function should return `k = 5`, with the first five elements of `nums` containing `0, 0, 1, 3, and 4`.
Note that the five elements can be returned in any order.
It does not matter what you leave beyond the returned `k` (hence they are underscores).

Constraints:

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

In [13]:

```
class Solution:
    # 遍历方法
    def removeElement(self, nums: List[int], val: int) -> int:

        current_i = 0
        k = 0
        for num in nums:
            if num != val:
                nums[k] = num #直接覆盖，因为循环不会往回看了
```

```
                k += 1
                current_i += 1
            else:
                current_i += 1
        return k

    # 快慢指针 fast and slow
    # 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。

    # slow 就是 k
    def removeElement(self, nums: List[int], val: int) -> int:
        current_i = 0
        k = 0
        fast = 0
        slow = 0
        size = len(nums)

        while fast < size:
            if nums[fast] != val:
                # slow 不会更新那么频繁， 满足 != val 才更新
                nums[slow] = nums[fast]
                slow += 1
            # 而 fast一直在更新，只要不等于
            fast += 1
        return slow
```

In [14]:

```
def test_remove_element():
    solution = Solution()

    # 测试用例列表
    test_cases = [
        {"nums": [3, 2, 2, 3], "val": 3, "expected_k": 2, "expected_nums": [2, 2]},
        {"nums": [0, 1, 2, 2, 3, 0, 4, 2], "val": 2, "expected_k": 5, "expected_nums": [0, 1, 3, 0, 4]},
        {"nums": [], "val": 1, "expected_k": 0, "expected_nums": []},
        {"nums": [4, 5], "val": 5, "expected_k": 1, "expected_nums": [4]},
        {"nums": [4, 4, 4], "val": 4, "expected_k": 0, "expected_nums": []},
    ]

    print("Testing the traverse method:")
    for i, case in enumerate(test_cases):
        nums = case["nums"][:]
        val = case["val"]
        expected_k = case["expected_k"]
        expected_nums = case["expected_nums"]

        # 调用第一种实现方法
        result_k = solution.removeElement(nums, val)
        assert result_k == expected_k, f"Test case {i+1} failed: k mismatch"
        assert sorted(nums[:result_k]) == sorted(expected_nums), f"Test case {i+1} failed: nums mismatch"
        print(f"Test case {i+1} passed!")

    print("\nTesting the fast and slow pointer method:")
    for i, case in enumerate(test_cases):
        nums = case["nums"][:]
        val = case["val"]
        expected_k = case["expected_k"]
        expected_nums = case["expected_nums"]

        # 调用第二种实现方法
        result_k = solution.removeElement(nums, val)
        assert result_k == expected_k, f"Test case {i+1} failed: k mismatch"
        assert sorted(nums[:result_k]) == sorted(expected_nums), f"Test case {i+1} failed: nums mismatch"
        print(f"Test case {i+1} passed!")

# 运行测试
test_remove_element()
```

```
Testing the traverse method:
Test case 1 passed!
Test case 2 passed!
Test case 3 passed!
Test case 4 passed!
Test case 5 passed!

Testing the fast and slow pointer method:
Test case 1 passed!
Test case 2 passed!
Test case 3 passed!
Test case 4 passed!
Test case 5 passed!
```