0081. 搜索旋转排序数组 II

ITCharge 大约 3 分钟

- 标签：数组、二分查找
- 难度：中等

题目链接

- 0081. 搜索旋转排序数组 II - 力扣

题目大意

描述：一个按照升序排列的整数数组 $nums$，在位置的某个下标 k 处进行了旋转操作。（例如：$[0, 1, 2, 5, 6, 8]$ 可能变为 $[5, 6, 8, 0, 1, 2]$）。

现在给定旋转后的数组 $nums$ 和一个整数 $target$。

要求：编写一个函数来判断给定的 $target$ 是否存在与数组中。如果存在则返回 `True`，否则返回 `False`。

说明：

- $1 \leq nums.length \leq 5000$。
- $-10^4 \leq nums[i] \leq 10^4$。
- 题目数据保证 $nums$ 在预先未知的某个下标上进行了旋转。
- $-10^4 \leq target \leq 10^4$。

示例：

- 示例 1：

```py
输入：nums = [2,5,6,0,0,1,2], target = 0
输出：true
```

- 示例 2：

```py
输入：nums = [2,5,6,0,0,1,2], target = 3
输出：false
```

解题思路

思路 1：二分查找

这道题算是「0033. 搜索旋转排序数组 」的变形，只不过输出变为了判断。

原本为升序排列的数组 nums 经过「旋转」之后，会有两种情况，第一种就是原先的升序序列，另一种是两段升序的序列。

```text
          *
        *
      *
    *
  *
*
```

```text
    *
  *
*
          *
        *
      *
```

最直接的办法就是遍历一遍，找到目标值 target。但是还可以有更好的方法。考虑用二分查找来降低算法的时间复杂度。

我们将旋转后的数组看成左右两个升序部分：左半部分和右半部分。

有人会说第一种情况不是只有一个部分吗？其实我们可以把第一种情况中的整个数组看做是左半部分，然后右半部分为空数组。

然后创建两个指针 $left$、$right$，分别指向数组首尾。让后计算出两个指针中间值 mid。将 mid 与两个指针做比较，并考虑与 $target$ 的关系。

- 如果 $nums[mid] > nums[left]$，则 mid 在左半部分（因为右半部分值都比 $nums[left]$ 小）。
  - 如果 $nums[mid] \geq target$，并且 $target \geq nums[left]$，则 $target$ 在左半部分，并且在 mid 左侧，此时应将 $right$ 左移到 $mid - 1$ 位置。
  - 否则如果 $nums[mid] < target$，则 $target$ 在左半部分，并且在 mid 右侧，此时应将 $left$ 右移到 $mid + 1$。
  - 否则如果 $nums[left] > target$，则 $target$ 在右半部分，应将 $left$ 移动到 $mid + 1$ 位置。
- 如果 $nums[mid] < nums[left]$，则 mid 在右半部分（因为右半部分值都比 $nums[left]$ 小）。
  - 如果 $nums[mid] < target$，并且 $target \leq nums[right]$，则 $target$ 在右半部分，并且在 mid 右侧，此时应将 $left$ 右移到 $mid + 1$ 位置。
  - 否则如果 $nums[mid] \geq target$，则 $target$ 在右半部分，并且在 mid 左侧，此时应将 $right$ 左移到 $mid - 1$ 位置。
  - 否则如果 $nums[right] < target$，则 $target$ 在左半部分，应将 $right$ 左移到 $mid - 1$ 位置。
- 最终判断 $nums[left]$ 是否等于 $target$，如果等于，则返回 `True`，否则返回 `False`。

思路 1：代码

```py
class Solution:
    def search(self, nums: List[int], target: int) -> bool:
        n = len(nums)
        if n == 0:
            return False

        left = 0
        right = len(nums) - 1
        while left < right:
            mid = left + (right - left) // 2

            if nums[mid] > nums[left]:
                if nums[left] <= target and target <= nums[mid]:
                    right = mid
                else:
                    left = mid + 1
            elif nums[mid] < nums[left]:
```

```
                if nums[mid] < target and target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid
            else:
                if nums[mid] == target:
                    return True
                else:
                    left = left + 1

        return nums[left] == target
```

思路 1：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 是数组 $nums$ 的长度。最坏情况下数组元素均相等且不为 $target$，我们需要访问所有位置才能得出结果。
- **空间复杂度**：$O(1)$。