27 搜索旋转排序数组

更新时间：2019-09-11 10:00:58

> 一个不注意小事情的人，永远不会成功大事业。
>
> ——戴尔·卡耐基

刷题内容

> 难度：Medium
>
> 题目链接：https://leetcode-cn.com/problems/search-in-rotated-sorted-array/

题目描述

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

(例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2])。

搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 -1 。

你可以假设数组中不存在重复的元素。

你的算法时间复杂度必须是 O(log n) 级别。

示例 1:

输入: nums = [4,5,6,7,0,1,2], target = 0
输出: 4
示例 2:

输入: nums = [4,5,6,7,0,1,2], target = 3
输出: -1

题目详解

题目中要求我们必须使用 O(log n) 级别的算法，所以我们的时间复杂度就有了限制。想一下，什么有什么方法可以让算法的时间复杂度为 O(log n) 呢？

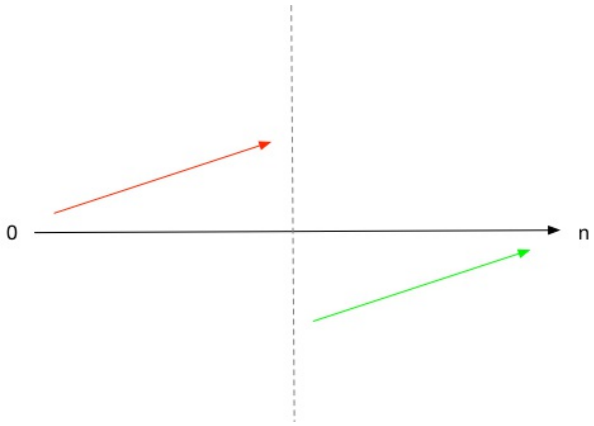
还记得之前的题目中我们用过的 **二分算法** 吗？二分算法其复杂度通常都是 O(log n) 的，二分算法同样适用于这道题，下面我们一起来看一下。

解题方案

思路 1 时间复杂度: O (lgN) 空间复杂度: O (1)

我们可以想一下，题目给的数组可以分成两部分了吧，前面一部分中的最小值都要比后面一部分的最大值还要大

下面是 rotated-array 图解:



所以这里引入一个二分法，那就是我们先搞两个指针，一个叫 l，指向数组第一个元素，一个叫 r，指向数组最后一个元素，然后我们用两个指针中间的那个元素 mid 来进行比较

- 如果 mid 指向的元素就是 target，return mid
- 如果 mid 和 r 同时落在绿线或者红线上
    - 如果 target 在 mid 右边， l 指针右移一位
    - 如果 target 在 mid 左边， r 指针左移一位
- 如果 r 在绿线上，mid 在红线上时
    - 如果 target 在 mid 右边， l 指针右移一位
    - 如果 target 在 mid 左边， r 指针左移一位
- 都没找到，return -1

下面来看具体的代码实现：

python beats 100%

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        l, r = 0, len(nums) - 1 # 左右指针
        while l <= r:
            mid = l + ((r - l) >> 2) # 得到左右指针中间的那个元素idx
            if nums[mid] == target: # 如果中间这个位置的元素就是target，直接返回
                return mid
            if nums[mid] <= nums[r]: # 如果mid和r同时落在绿线或者红线上
                if nums[mid] < target <= nums[r]: # 说明target在mid右边
                    l = mid + 1
                else: # 说明target在mid左边
                    r = mid - 1
            else: # 只有当r在绿线上，mid在红线上时才会有这种情况
                if nums[l] <= target < nums[mid]: # 说明target在mid左边
                    r = mid - 1
                else: # 说明target在mid右边
                    l = mid + 1
        return -1 # 一直没找到，就返回-1
```

c++ beats 100%

```cpp
class Solution {
public:
    //二分法
    int search(vector<int>& nums, int target) {
        int l = 0;
        int r = (int)nums.size() - 1;
        //搜索区间[l,r]

        while (l <= r) {
            //获得区间[l,r]的中点
            int mid = (l + r) / 2;
            if (nums[mid] == target) {
                return mid;
            }
            if (nums[mid] <= nums[r]) {
                //mid和r同时落在绿线或者红线上
                if (nums[mid] < target && nums[r] >= target) {
                    //target在mid的右边
                    l = mid + 1;
                } else {
                    //target在mid的左边
                    r = mid - 1;
                }
            } else {
                //mid在红线，r在绿线的情况
                if (nums[l] <= target && target < nums[mid]) {
                    //target在mid的左边
                    r = mid - 1;
                } else {
                    //target在mid的右边
                    l = mid + 1;
                }
            }
        }
        return -1;
    }
};
```

java beats 100%

```java
class Solution {
    public int search(int[] nums, int target) {
        int l = 0;
        int r = nums.length - 1;
        //搜索区间[l,r]

        while (l <= r) {
            //获得区间[l,r]的中点
            int mid = (l + r) / 2;
            if (nums[mid] == target) {
                return mid;
            }
            if (nums[mid] <= nums[r]) {
                //mid和r同时落在绿线或者红线上
                if (nums[mid] < target && nums[r] >= target) {
                    //target在mid的右边
                    l = mid + 1;
                } else {
                    //target在mid的左边
                    r = mid - 1;
                }
            } else {
                //mid在红线，r在绿线的情况
                if (nums[l] <= target && target < nums[mid]) {
                    //target在mid的左边
                    r = mid - 1;
                } else {
                    //target在mid的右边
                    l = mid + 1;
                }
            }
        }
        return -1;
    }
}
```

go beats 100%

```
func search(nums []int, target int) int {
    l := 0
 r := len(nums) - 1
 //搜索区间[l,r]

 for l <= r {
  //获得区间[l,r]的中点
  mid := (l + r) / 2
  if nums[mid] == target {
   return mid
  }
  if nums[mid] <= nums[r] {
   //mid和r同时落在绿线或者红线上
   if nums[mid] < target && nums[r] >= target {
    //target在mid的右边
    l = mid + 1
   } else {
    //target在mid的左边
    r = mid - 1
   }
  } else {
   //mid在红线，r在绿线的情况
   if nums[l] <= target && target < nums[mid] {
    //target在mid的左边
    r = mid - 1
   } else {
    //target在mid的右边
    l = mid + 1
   }
  }
 }
 return -1
}
```

}

26 下一个排列

28 摆动排序