0167. 两数之和 II - 输入有序数组

ITCharge 大约 3 分钟

- 标签：数组、双指针、二分查找
- 难度：中等

题目链接

- 0167. 两数之和 II - 输入有序数组 - 力扣

题目大意

描述：给定一个下标从 1 开始计数、升序排列的整数数组：$numbers$ 和一个目标值 $target$。

要求：从数组中找出满足相加之和等于 $target$ 的两个数，并返回两个数在数组中下的标值。

说明：

- $2 \leq numbers.length \leq 3 \times 10^4$。
- $-1000 \leq numbers[i] \leq 1000$。
- $numbers$ 按非递减顺序排列。
- $-1000 \leq target \leq 1000$。
- 仅存在一个有效答案。

示例：

- 示例 1：

```py
输入：numbers = [2,7,11,15], target = 9
输出：[1,2]
解释：2 与 7 之和等于目标数 9。因此 index1 = 1, index2 = 2。返回 [1, 2]。
```

- 示例 2：

```py
输入：numbers = [2,3,4], target = 6
输出：[1,3]
解释：2 与 4 之和等于目标数 6。因此 index1 = 1, index2 = 3。返回 [1, 3]。
```

解题思路

这道题如果暴力遍历数组，从中找到相加之和等于 $target$ 的两个数，时间复杂度为 $O(n^2)$，可以尝试一下。

```py
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        size = len(numbers)
        for i in range(size):
            for j in range(i + 1, size):
                if numbers[i] + numbers[j] == target:
                    return [i + 1, j + 1]
        return [-1, -1]
```

结果不出意外的超时了。所以我们要想办法降低时间复杂度。

思路 1：二分查找

因为数组是有序的，可以考虑使用二分查找来减少时间复杂度。具体做法如下：

1. 使用一重循环遍历数组，先固定第一个数，即 $numsbers[i]$。
2. 然后使用二分查找的方法寻找符合要求的第二个数。
3. 使用两个指针 $left$，$right$。$left$ 指向数组第一个数的下一个数，$right$ 指向数组值最大元素位置。
4. 判断第一个数 $numsbers[i]$ 和两个指针中间元素 $numbers[mid]$ 的和与目标值的关系。
    1. 如果 $numbers[mid] + numbers[i] < target$，排除掉不可能区间 $[left, mid]$，在 $[mid + 1, right]$ 中继续搜索。
    2. 如果 $numbers[mid] + numbers[i] \geq target$，则第二个数可能在 $[left, mid]$ 中，则在 $[left, mid]$ 中继续搜索。
5. 直到 $left$ 和 $right$ 移动到相同位置停止检测。如果 $numbers[left] + numbers[i] == target$，则返回两个元素位置 $[left + 1, i + 1]$（下标从 1 开始计数）。
6. 如果最终仍没找到，则返回 $[-1, -1]$。

思路 1：代码

```py
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        for i in range(len(numbers)):
            left, right = i + 1, len(numbers) - 1
            while left < right:
                mid = left + (right - left) // 2
                if numbers[mid] + numbers[i] < target:
                    left = mid + 1
                else:
                    right = mid
            if numbers[left] + numbers[i] == target:
                return [i + 1, left + 1]

        return [-1, -1]
```

思路 1：复杂度分析

- **时间复杂度**：$O(n \times \log n)$。
- **空间复杂度**：$O(1)$。

思路 2：对撞指针

可以考虑使用对撞指针来减少时间复杂度。具体做法如下：

1. 使用两个指针 $left$，$right$。$left$ 指向数组第一个值最小的元素位置，$right$ 指向数组值最大元素位置。
2. 判断两个位置上的元素的和与目标值的关系。
    1. 如果元素和等于目标值，则返回两个元素位置。
    2. 如果元素和大于目标值，则让 $right$ 左移，继续检测。
    3. 如果元素和小于目标值，则让 $left$ 右移，继续检测。
3. 直到 $left$ 和 $right$ 移动到相同位置停止检测。
4. 如果最终仍没找到，则返回 $[-1, -1]$。

思路 2：代码

```py
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        left = 0
        right = len(numbers) - 1
        while left < right:
            total = numbers[left] + numbers[right]
            if total == target:
                return [left + 1, right + 1]
            elif total < target:
                left += 1
            else:
                right -= 1
        return [-1, -1]
```

思路 2：复杂度分析

- **时间复杂度**：$O(n)$。
- **空间复杂度**：$O(1)$。只用到了常数空间存放若干变量。