0480. 滑动窗口中位数

ITCharge 大约 6 分钟

- 标签：数组、哈希表、滑动窗口、堆（优先队列）
- 难度：困难

题目链接

- [0480. 滑动窗口中位数 - 力扣](#)

题目大意

描述：给定一个数组 $nums$，有一个长度为 k 的窗口从最左端滑动到最右端。窗口中有 k 个数，每次窗口向右移动 1 位。

要求：找出每次窗口移动后得到的新窗口中元素的中位数，并输出由它们组成的数组。

说明：

- **中位数**：有序序列最中间的那个数。如果序列的长度是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。
- 例如：
  - $[2, 3, 4]$，中位数是 3
  - $[2, 3]$，中位数是 $(2 + 3) / 2 = 2.5$。
- 你可以假设 k 始终有效，即：k 始终小于等于输入的非空数组的元素个数。
- 与真实值误差在 10^{-5} 以内的答案将被视作正确答案。

示例：

- 示例 1：

```py
给出 nums = [1,3,-1,-3,5,3,6,7]，以及 k = 3。

窗口位置                      中位数
---------------               -----
[1  3  -1] -3  5  3  6  7       1
 1 [3  -1  -3] 5  3  6  7      -1
```

```
1  3 [-1  -3  5] 3  6  7       -1
1  3  -1 [-3  5  3] 6  7        3
1  3  -1  -3 [5  3  6] 7        5
1  3  -1  -3  5 [3  6  7]       6
因此，返回该滑动窗口的中位数数组 [1,-1,-1,3,5,6]。
```

解题思路

思路 1：小顶堆 + 大顶堆

题目要求动态维护长度为 k 的窗口中元素的中位数。如果对窗口元素进行排序，时间复杂度一般是 $O(k \times \log k)$。如果对每个区间都进行排序，那时间复杂度就更大了，肯定会超时。

我们需要借助一个内部有序的数据结构，来降低取窗口中位数的时间复杂度。Python 可以借助 `heapq` 构建大顶堆和小顶堆。通过 k 的奇偶性和堆顶元素来获取中位数。

接下来还要考虑几个问题：初始化问题、取中位数问题、窗口滑动中元素的添加删除操作。接下来一一解决。

初始化问题：

我们将所有大于中位数的元素放到 $heap\_max$（小顶堆）中，并且元素个数向上取整。然后再将所有小于等于中位数的元素放到 $heap\_min$（大顶堆）中，并且元素个数向下取整。这样当 k 为奇数时，$heap\_max$ 比 $heap\_min$ 多一个元素，中位数就是 $heap\_max$ 堆顶元素。当 k 为偶数时，$heap\_max$ 和 $heap\_min$ 中的元素个数相同，中位数就是 $heap\_min$ 堆顶元素和 $heap\_max$ 堆顶元素的平均数。这个过程操作如下：

- 先将数组中前 k 个元素放到 $heap\_max$ 中。
- 再从 $heap\_max$ 中取出 $k//2$ 个堆顶元素放到 $heap\_min$ 中。

取中位数问题（上边提到过）：

- 当 k 为奇数时，中位数就是 $heap\_max$ 堆顶元素。当 k 为偶数时，中位数就是 $heap\_max$ 堆顶元素和 $heap\_min$ 堆顶元素的平均数。

窗口滑动过程中元素的添加和删除问题：

- 删除：每次滑动将窗口左侧元素删除。由于 `heapq` 没有提供删除中间特定元素相对应的方法。所以我们使用「延迟删除」的方式先把待删除的元素标记上，等到待删除的元素出现在堆顶时，再将其移除。我们使用 $removes$（哈希表）来记录待删除元素个数。
  - 将窗口左侧元素删除的操作为：`removes[nums[left]] += 1`。

- 添加：每次滑动在窗口右侧添加元素。需要根据上一步删除的结果来判断需要添加到哪一个堆上。我们用 $banlance$ 记录 $heap\_max$ 和 $heap\_min$ 元素个数的差值。
  - 如果窗口左边界 $nums[left]$小于等于 $heap\_max$ 堆顶元素 ，则说明上一步删除的元素在 $heap\_min$ 上，则让 `banlance -= 1` 。
  - 如果窗口左边界 $nums[left]$ 大于 $heap\_max$ 堆顶元素，则说明上一步删除的元素在 $heap\_max$ 上，则上 `banlance += 1` 。
  - 如果窗口右边界 $nums[right]$ 小于等于 $heap\_max$ 堆顶元素，则说明待添加元素需要添加到 $heap\_min$ 上，则让 `banlance += 1` 。
  - 如果窗口右边界 $nums[right]$ 大于 $heap\_max$ 堆顶元素，则说明待添加元素需要添加到 $heap\_max$ 上，则让 `banlance -= 1` 。
- 经过上述操作，$banlance$ 的取值为 0、-2、2 中的一种。需要经过调整使得 $banlance == 0$。
  - 如果 $banlance == 0$，已经平衡，不需要再做操作。
  - 如果 $banlance == -2$，则说明 $heap\_min$ 比 $heap\_max$ 的元素多了两个。则从 $heap\_min$ 中取出堆顶元素添加到 $heap\_max$ 中。
  - 如果 $banlance == 2$，则说明 $heap\_max$ 比 $heap\_min$ 的元素多了两个。则从 $heap\_max$ 中取出堆顶元素添加到 $heap\_min$ 中。
- 调整完之后，分别检查 $heap\_max$ 和 $heap\_min$ 的堆顶元素。
  - 如果 $heap\_max$ 堆顶元素恰好为待删除元素，即 $removes[-heap\_max[0]] > 0$，则弹出 $heap\_max$ 堆顶元素。
  - 如果 $heap\_min$ 堆顶元素恰好为待删除元素，即 $removes[heap\_min[0]] > 0$，则弹出 $heap\_min$ 堆顶元素。
- 最后取中位数放入答案数组中，然后继续滑动窗口。

思路 1：代码

```py
import collections
import heapq

class Solution:
    def median(self, heap_max, heap_min, k):
        if k % 2 == 1:
            return -heap_max[0]
        else:
            return (-heap_max[0] + heap_min[0]) / 2

    def medianSlidingWindow(self, nums: List[int], k: int) -> List[float]:
        heap_max, heap_min = [], []
        removes = collections.Counter()
```

```python
        for i in range(k):
            heapq.heappush(heap_max, -nums[i])
        for i in range(k // 2):
            heapq.heappush(heap_min, -heapq.heappop(heap_max))

        res = [self.median(heap_max, heap_min, k)]

        for i in range(k, len(nums)):
            banlance = 0
            left, right = i - k, i
            removes[nums[left]] += 1
            if heap_max and nums[left] <= -heap_max[0]:
                banlance -= 1
            else:
                banlance += 1

            if heap_max and nums[right] <= -heap_max[0]:
                heapq.heappush(heap_max, -nums[i])
                banlance += 1
            else:
                banlance -= 1
                heapq.heappush(heap_min, nums[i])

            if banlance == -2:
                heapq.heappush(heap_max, -heapq.heappop(heap_min))
            if banlance == 2:
                heapq.heappush(heap_min, -heapq.heappop(heap_max))

            while heap_max and removes[-heap_max[0]] > 0:
                removes[-heapq.heappop(heap_max)] -= 1
            while heap_min and removes[heap_min[0]] > 0:
                removes[heapq.heappop(heap_min)] -= 1
            res.append(self.median(heap_max, heap_min, k))

        return res
```

思路 1：复杂度分析

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

参考资料

- 【题解】《风 险 对 冲》：双堆对顶，大堆小堆同时维护，44ms - 滑动窗口中位数 - 力扣