0862. 和至少为 K 的最短子数组

ITCharge 大约 4 分钟

- 标签：队列、数组、二分查找、前缀和、滑动窗口、单调队列、堆（优先队列）
- 难度：困难

题目链接

- 0862. 和至少为 K 的最短子数组 - 力扣

题目大意

描述：给定一个整数数组 $nums$ 和一个整数 k。

要求：找出 $nums$ 中和至少为 k 的最短非空子数组，并返回该子数组的长度。如果不存在这样的子数组，返回 -1。

说明：

- **子数组**：数组中连续的一部分。
- $1 \le nums.length \le 10^5$。
- $-10^5 \le nums[i] \le 10^5$。
- $1 \le k \le 10^9$。

示例：

- 示例 1：

```py
输入：nums = [1], k = 1
输出：1
```

- 示例 2：

```py
输入：nums = [1,2], k = 4
输出：-1
```

解题思路

思路 1：前缀和 + 单调队列

题目要求得到满足和至少为 k 的子数组的最短长度。

先来考虑暴力做法。如果使用两重循环分别遍历子数组的开始和结束位置，则可以直接求出所有满足条件的子数组，以及对应长度。但是这种做法的时间复杂度为 $O(n^2)$。我们需要对其进行优化。

1. 前缀和优化

首先对于子数组和，我们可以使用「前缀和」的方式，方便快速的得到某个子数组的和。

对于区间 $[left, right]$，通过 $pre\_sum[right + 1] - prefix\_cnts[left]$ 即可快速求解出区间 $[left, right]$ 的子数组和。

此时问题就转变为：是否能找到满足 i，j 且 $pre\_sum[i] - pre\_sum[j] \geq k$ 两个条件的子数组 $[j, i)$？如果能找到，则找出 $i - j$ 值最小的作为答案。

2. 单调队列优化

对于区间 $[j, i)$ 来说，我们应该尽可能的减少不成立的区间枚举。

1. 对于某个区间 $[j, i)$ 来说，如果 $pre\_sum[i] - pre\_sum[j] \geq k$，那么大于 i 的索引值就不用再进行枚举了，不可能比 $i - j$ 的差值更优了。此时我们应该尽可能的向右移动 j，从而使得 $i - j$ 更小。
2. 对于某个区间 $[j, i)$ 来说，如果 $pre\_sum[j] \geq pre\_sum[i]$，对于任何大于等于 i 的索引值 r 来说，$pre\_sum[r] - pre\_sum[i]$ 一定比 $pre\_sum[i] - pre\_sum[j]$ 更小且长度更小，此时 $pre\_sum[j]$ 可以直接忽略掉。

因此，我们可以使用单调队列来维护单调递增的前缀数组 pre_sum。其中存放了下标 x：$x_0, x_1, \ldots$，满足 $pre\_sum[x_0] < pre\_sum[x_1] < \ldots$ 单调递增。

1. 使用一重循环遍历位置 i，将当前位置 i 存入倒掉队列中。
2. 对于每一个位置 i，如果单调队列不为空，则可以判断其之前存入在单调队列中的 $pre\_sum[j]$ 值，如果 $pre\_sum[i] - pre\_sum[j] \geq k$，则更新答案，并将 j 从队头位置

弹出。直到不再满足 $pre\_sum[i] - pre\_sum[j] \geq k$ 时为止（即 $pre\_sum[i] - pre\_sum[j] < k$）。
3. 如果队尾 $pre\_sum[j] \geq pre\_sum[i]$，那么说明以后无论如何都不会再考虑 $pre\_sum[j]$ 了，则将其从队尾弹出。
4. 最后遍历完返回答案。

思路 1：代码

```Python
class Solution:
    def shortestSubarray(self, nums: List[int], k: int) -> int:
        size = len(nums)

        # 优化 1
        pre_sum = [0 for _ in range(size + 1)]
        for i in range(size):
            pre_sum[i + 1] = pre_sum[i] + nums[i]

        ans = float('inf')
        queue = collections.deque()

        for i in range(size + 1)
            # 优化 2
            while queue and pre_sum[i] - pre_sum[queue[0]] >= k:
                ans = min(ans, i - queue.popleft())
            while queue and pre_sum[queue[-1]] >= pre_sum[i]:
                queue.pop()
            queue.append(i)

        if ans == float('inf'):
            return -1
        return ans
```

思路 1：复杂度分析

- **时间复杂度**：$O(n)$，其中 n 为数组 $nums$ 的长度。
- **空间复杂度**：$O(n)$。