0560. 和为 K 的子数组

ITCharge 大约 3 分钟

- 标签：数组、哈希表、前缀和
- 难度：中等

题目链接

- 0560. 和为 K 的子数组 - 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：找到该数组中和为 k 的连续子数组的个数。

说明：

- $1 \leq nums.length \leq 2 \times 10^4$。
- $-1000 \leq nums[i] \leq 1000$。 $-10^7 \leq k \leq 10^7$。

示例：

- 示例 1：

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

- 示例 2：

```py
输入：nums = [1,2,3], k = 3
输出：2
```

解题思路

思路 1：枚举算法（超时）

先考虑暴力做法，外层两重循环，遍历所有连续子数组，然后最内层再计算一下子数组的和。部分代码如下：

```py
for i in range(len(nums)):
    for j in range(i + 1):
        sum = countSum(i, j)
```

这样下来时间复杂度就是 $O(n^3)$ 了。下一步是想办法降低时间复杂度。

对于以 i 开头，以 j 结尾（$i \leq j$）的子数组 $nums[i] \dots nums[j]$ 来说，我们可以通过顺序遍历 j，逆序遍历 i 的方式（或者前缀和的方式），从而在 $O(n^2)$ 的时间复杂度内计算出子数组的和，同时使用变量 cnt 统计出和为 k 的子数组个数。

但这样提交上去超时了。

思路 1：代码

```py
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        cnt = 0
        for j in range(len(nums)):
            sum = 0
            for i in range(j, -1, -1):
                sum += nums[i]
                if sum == k:
                    cnt += 1

        return cnt
```

思路 1：复杂度分析

- **时间复杂度**：$O(n^2)$。
- **空间复杂度**：$O(1)$。

思路 2：前缀和 + 哈希表

先用一重循环遍历数组，计算出数组 $nums$ 中前 j 个元素的和（前缀和），保存到一维数组 pre_sum 中，那么对于任意 $nums[i]...nums[j]$ 的子数组的和为 $pre\_sum[j] - pre\_sum[i-1]$。这样计算子数组和的时间复杂度降为了 $O(1)$。总体时间复杂度为 $O(n^2)$。

但是还是超时了。。

由于我们只关心和为 k 出现的次数，不关心具体的解，可以使用哈希表来加速运算。

$pre\_sum[i]$ 的定义是前 i 个元素和，则 $pre\_sum[i]$ 可以由 $pre\_sum[i-1]$ 递推而来，即：$pre\_sum[i] = pre\_sum[i-1] + num[i]$。$[i..j]$ 子数组和为 k 可以转换为：$pre\_sum[j] - pre\_sum[i-1] == k$。

综合一下，可得：$pre\underline{\hspace{0.5em}}sum[i - 1] == pre\underline{\hspace{0.5em}}sum[j] - k $。

所以，当我们考虑以 j 结尾和为 k 的连续子数组个数时，只需要统计有多少个前缀和为 $pre\_sum[j] - k$ （即 $pre\_sum[i-1]$）的个数即可。具体做法如下：

- 使用 pre_sum 变量记录前缀和（代表 $pre\_sum[j]$）。
- 使用哈希表 pre_dic 记录 $pre\_sum[j]$ 出现的次数。键值对为 $pre\_sum[j]$ ：pre_sum_count。
- 从左到右遍历数组，计算当前前缀和 pre_sum。
- 如果 $pre\_sum - k$ 在哈希表中，则答案个数累加上 $pre\_dic[pre\_sum - k]$。
- 如果 pre_sum 在哈希表中，则前缀和个数累加 1，即 $pre\_dic[pre\_sum] += 1$。
- 最后输出答案个数。

思路 2：代码

```py
class Solution:
    def subarraySum(self, nums: List[int], k: int) -> int:
        pre_dic = {0: 1}
        pre_sum = 0
        count = 0
        for num in nums:
            pre_sum += num
            if pre_sum - k in pre_dic:
                count += pre_dic[pre_sum - k]
            if pre_sum in pre_dic:
                pre_dic[pre_sum] += 1
            else:
                pre_dic[pre_sum] = 1
        return count
```

思路 2：复杂度分析

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