# 题目

给你一个下标从 0 开始长度为 n 的整数数组 nums ，和一个下标从 0 开始长度为 m 的整数数组 pattern ，pattern 数组只包含整数 -1 ，0 和 1 。

大小为 m + 1 的子数组nums[i..j] 如果对于每个元素 pattern[k] 都满足以下条件，那么我们说这个子数组匹配模式数组pattern ：

如果 pattern[k] == 1 ，那么 nums[i + k + 1] > nums[i + k]

如果 pattern[k] == 0 ，那么 nums[i + k + 1] == nums[i + k]

如果 pattern[k] == -1 ，那么 nums[i + k + 1] < nums[i + k]

请你返回匹配 pattern 的 nums 子数组的数目。

示例 1：

输入：nums = [1,2,3,4,5,6], pattern = [1,1]

输出：4

解释：模式 [1,1] 说明我们要找的子数组是长度为 3 且严格上升的。在数组 nums 中，子数组 [1,2,3] ，[2,3,4] ，[3,4,5] 和 [4,5,6] 都匹配这个模式。

所以 nums 中总共有 4 个子数组匹配这个模式。

示例 2：

输入：nums = [1,4,4,1,3,5,5,3], pattern = [1,0,-1]

输出：2

解释：这里，模式数组 [1,0,-1] 说明我们需要找的子数组中，第一个元素小于第二个元素，第二个元素等于第三个元素，第三个元素大于第四个元素。在 nums 中，子数组 [1,4,4,1] 和 [3,5,5,3] 都匹配这个模式。

所以 nums 中总共有 2 个子数组匹配这个模式。

提示：

2 <= n == nums.length <= 100

1 <= nums[i] <= 109

1 <= m == pattern.length < n

-1 <= pattern[i] <= 1

# 分析

## 方法一：暴力法

思路：

对于每个可能的子数组，检查是否与给定的模式数组匹配。

说明：这里使用暴力运行会堆溢出。

代码：

class Solution {

public:

int countMatchingSubarrays(vector<int>& nums, vector<int>& pattern) {

int count = 0;

int n = nums.size();

int m = pattern.size();

for (int i = 0; i <= n - m; ++i) {

if (isMatching(nums, pattern, i)) {

count++;

}

}

return count;

}

private:

bool isMatching(vector<int>& nums, vector<int>& pattern, int start) {

for (int k = 0; k < pattern.size(); ++k) {

int num1 = nums[start + k];

int num2 = nums[start + k + 1];

int p = pattern[k];

if ((p == 1 && num1 >= num2) ||

(p == 0 && num1 != num2) ||

(p == -1 && num1 <= num2)) {

return false;

}

}

return true;

}

};

## 方法二：双指针

思路：

基本步骤如下：

1、遍历数组nums，使用两个指针i和j分别表示子数组的起始和结束位置。

2、对于每个i，从i开始向后遍历数组nums直到nums的倒数第二个元素，同时检查当前子数组是否与模式数组pattern匹配。

3、检查匹配的条件是：

- 如果pattern[k] == 1，则要求nums[i + k] < nums[i + k + 1]；

- 如果pattern[k] == 0，则要求nums[i + k] == nums[i + k + 1]；

- 如果pattern[k] == -1，则要求nums[i + k] > nums[i + k + 1]。

4、如果当前子数组满足模式数组的所有要求，则将匹配的子数组数量增加 1。

5、返回匹配的子数组数量作为最终结果。

代码：

class Solution {

public:

int countMatchingSubarrays(vector<int>& nums, vector<int>& pattern) {

int count = 0;

int n = nums.size();

int m = pattern.size();

for (int i = 0; i <= n - m; ++i) {

int j = i;

while (j < n - 1 && isMatching(nums[j], nums[j + 1], pattern[j - i])) {

if (j - i == m - 1) {

count++;

break;

}

j++;

}

}

return count;

}

private:

bool isMatching(int num1, int num2, int p) {

if ((p == 1 && num1 >= num2) ||

(p == 0 && num1 != num2) ||

(p == -1 && num1 <= num2)) {

return false;

}

return true;

}

};