Problem 3036: Number of Subarrays That Match a Pattern II

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given a

0-indexed

integer array

nums

of size

n

, and a

0-indexed

integer array

pattern

of size

m

consisting of integers

-1

,

0

, and

1

.

A

subarray

nums[i..j]

of size

m + 1

is said to match the

pattern

if the following conditions hold for each element

pattern[k]

:

nums[i + k + 1] > nums[i + k]

if

pattern[k] == 1

.

nums[i + k + 1] == nums[i + k]

if

pattern[k] == 0

.

nums[i + k + 1] < nums[i + k]

if

pattern[k] == -1

.

Return

the

count

of subarrays in

nums

that match the

pattern

.

Example 1:

Input:

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

Output:

4

Explanation:

The pattern [1,1] indicates that we are looking for strictly increasing subarrays of size 3. In the array nums, the subarrays [1,2,3], [2,3,4], [3,4,5], and [4,5,6] match this pattern. Hence, there are 4 subarrays in nums that match the pattern.

Example 2:

Input:

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

Output:

2

Explanation:

Here, the pattern [1,0,-1] indicates that we are looking for a sequence where the first number is smaller than the second, the second is equal to the third, and the third is greater than the fourth. In the array nums, the subarrays [1,4,4,1], and [3,5,5,3] match this pattern. Hence, there are 2 subarrays in nums that match the pattern.

Constraints:

2 <= n == nums.length <= 10

6

1 <= nums[i] <= 10

9

1 <= m == pattern.length < n

-1 <= pattern[i] <= 1

Code Snippets

C++:

```
class Solution {
public:
int countMatchingSubarrays(vector<int>& nums, vector<int>& pattern) {

}
};
```

Java:

```
class Solution {
public int countMatchingSubarrays(int[] nums, int[] pattern) {

}
}
```

Python3:

```
class Solution:
def countMatchingSubarrays(self, nums: List[int], pattern: List[int]) -> int:
```

Python:

```
class Solution(object):
def countMatchingSubarrays(self, nums, pattern):
"""
:type nums: List[int]
:type pattern: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number[]} pattern
* @return {number}
*/
var countMatchingSubarrays = function(nums, pattern) {

};
```

TypeScript:

```
function countMatchingSubarrays(nums: number[], pattern: number[]): number {

};
```

C#:

```
public class Solution {
public int CountMatchingSubarrays(int[] nums, int[] pattern) {

}
}
```

C:

```
int countMatchingSubarrays(int* nums, int numsSize, int* pattern, int
patternSize) {

}
```

Go:

```
func countMatchingSubarrays(nums []int, pattern []int) int {

}
```

Kotlin:

```
class Solution {
fun countMatchingSubarrays(nums: IntArray, pattern: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func countMatchingSubarrays(_ nums: [Int], _ pattern: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_matching_subarrays(nums: Vec<i32>, pattern: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} pattern
# @return {Integer}
def count_matching_subarrays(nums, pattern)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $pattern
* @return Integer
*/
function countMatchingSubarrays($nums, $pattern) {

}
}
```

Dart:

```
class Solution {
int countMatchingSubarrays(List<int> nums, List<int> pattern) {

}
}
```

Scala:

```
object Solution {
def countMatchingSubarrays(nums: Array[Int], pattern: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_matching_subarrays(nums :: [integer], pattern :: [integer]) ::
integer
def count_matching_subarrays(nums, pattern) do

end
end
```

Erlang:

```
-spec count_matching_subarrays(Nums :: [integer()], Pattern :: [integer()])
-> integer().
count_matching_subarrays(Nums, Pattern) ->
.
```

Racket:

```
(define/contract (count-matching-subarrays nums pattern)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Subarrays That Match a Pattern II
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int countMatchingSubarrays(vector<int>& nums, vector<int>& pattern) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Subarrays That Match a Pattern II
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int countMatchingSubarrays(int[] nums, int[] pattern) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Subarrays That Match a Pattern II
Difficulty: Hard
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
```

```
Space Complexity: O(n) for hash map
"""

class Solution:
def countMatchingSubarrays(self, nums: List[int], pattern: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countMatchingSubarrays(self, nums, pattern):
"""
:type nums: List[int]
:type pattern: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Subarrays That Match a Pattern II
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums
* @param {number[]} pattern
* @return {number}
*/
var countMatchingSubarrays = function(nums, pattern) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Subarrays That Match a Pattern II
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function countMatchingSubarrays(nums: number[], pattern: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Subarrays That Match a Pattern II
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int CountMatchingSubarrays(int[] nums, int[] pattern) {

}
}
```

C Solution:

```
/*
* Problem: Number of Subarrays That Match a Pattern II
* Difficulty: Hard
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

int countMatchingSubarrays(int* nums, int numsSize, int* pattern, int
patternSize) {

}
```

Go Solution:

```
// Problem: Number of Subarrays That Match a Pattern II
// Difficulty: Hard
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func countMatchingSubarrays(nums []int, pattern []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countMatchingSubarrays(nums: IntArray, pattern: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func countMatchingSubarrays(_ nums: [Int], _ pattern: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Subarrays That Match a Pattern II
// Difficulty: Hard
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn count_matching_subarrays(nums: Vec<i32>, pattern: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} pattern
# @return {Integer}
def count_matching_subarrays(nums, pattern)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $pattern
* @return Integer
*/
function countMatchingSubarrays($nums, $pattern) {

}
}
```

Dart Solution:

```
class Solution {
int countMatchingSubarrays(List<int> nums, List<int> pattern) {

}
}
```

Scala Solution:

```
object Solution {
def countMatchingSubarrays(nums: Array[Int], pattern: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_matching_subarrays(nums :: [integer], pattern :: [integer]) ::
integer
def count_matching_subarrays(nums, pattern) do

end
end
```

Erlang Solution:

```
-spec count_matching_subarrays(Nums :: [integer()], Pattern :: [integer()])
-> integer().
count_matching_subarrays(Nums, Pattern) ->
.
```

Racket Solution:

```
(define/contract (count-matching-subarrays nums pattern)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```