Problem 3404: Count Special Subsequences

Problem Information

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

Problem Description

You are given an array

nums

consisting of positive integers.

A

special subsequence

is defined as a

subsequence

of length 4, represented by indices

(p, q, r, s)

, where

p < q < r < s

. This subsequence

must

satisfy the following conditions:

nums[p] * nums[r] == nums[q] * nums[s]

There must be

at least

one

element between each pair of indices. In other words,

q - p > 1

,

r - q > 1

and

s - r > 1

.

Return the

number

of different

special

subsequences

in

nums

.

Example 1:

Input:

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

Output:

1

Explanation:

There is one special subsequence in

nums

.

(p, q, r, s) = (0, 2, 4, 6)

:

This corresponds to elements

(1, 3, 3, 1)

.

nums[p] * nums[r] = nums[0] * nums[4] = 1 * 3 = 3

nums[q] * nums[s] = nums[2] * nums[6] = 3 * 1 = 3

Example 2:

Input:

nums = [3,4,3,4,3,4,3,4]

Output:

3

Explanation:

There are three special subsequences in

nums

.

(p, q, r, s) = (0, 2, 4, 6)

:

This corresponds to elements

(3, 3, 3, 3)

.

nums[p] * nums[r] = nums[0] * nums[4] = 3 * 3 = 9

nums[q] * nums[s] = nums[2] * nums[6] = 3 * 3 = 9

(p, q, r, s) = (1, 3, 5, 7)

:

This corresponds to elements

(4, 4, 4, 4)

.

nums[p] * nums[r] = nums[1] * nums[5] = 4 * 4 = 16

nums[q] * nums[s] = nums[3] * nums[7] = 4 * 4 = 16

(p, q, r, s) = (0, 2, 5, 7)

:

This corresponds to elements

(3, 3, 4, 4)

.

nums[p] * nums[r] = nums[0] * nums[5] = 3 * 4 = 12

nums[q] * nums[s] = nums[2] * nums[7] = 3 * 4 = 12

Constraints:

7 <= nums.length <= 1000

1 <= nums[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
long long numberOfSubsequences(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public long numberOfSubsequences(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numberOfSubsequences(nums: number[]): number {

};
```

C#:

```
public class Solution {
public long NumberOfSubsequences(int[] nums) {

}
}
```

C:

```
long long numberOfSubsequences(int* nums, int numsSize) {

}
```

Go:

```
func numberOfSubsequences(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun numberOfSubsequences(nums: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func numberOfSubsequences(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_subsequences(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def number_of_subsequences(nums)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[] $nums
* @return Integer
*/
function numberOfSubsequences($nums) {

}
}
```

Dart:

```
class Solution {
int numberOfSubsequences(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def numberOfSubsequences(nums: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_subsequences(nums :: [integer]) :: integer
def number_of_subsequences(nums) do

end
end
```

Erlang:

```
-spec number_of_subsequences(Nums :: [integer()]) -> integer().
number_of_subsequences(Nums) ->
.
```

Racket:

```
(define/contract (number-of-subsequences nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Special Subsequences
* Difficulty: Medium
* Tags: array, math, 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:
long long numberOfSubsequences(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Count Special Subsequences
* Difficulty: Medium
* Tags: array, math, 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 long numberOfSubsequences(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Count Special Subsequences
Difficulty: Medium
Tags: array, math, 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 numberOfSubsequences(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Special Subsequences
* Difficulty: Medium
* Tags: array, math, 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
* @return {number}
*/
var numberOfSubsequences = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Special Subsequences
* Difficulty: Medium
* Tags: array, math, 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 numberOfSubsequences(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Special Subsequences
* Difficulty: Medium
* Tags: array, math, 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 long NumberOfSubsequences(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Special Subsequences
* Difficulty: Medium
* Tags: array, math, 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
*/

long long numberOfSubsequences(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Special Subsequences
// Difficulty: Medium
// Tags: array, math, 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 numberOfSubsequences(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfSubsequences(nums: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func numberOfSubsequences(_ nums: [Int]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Count Special Subsequences
// Difficulty: Medium
// Tags: array, math, 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 number_of_subsequences(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def number_of_subsequences(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function numberOfSubsequences($nums) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfSubsequences(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfSubsequences(nums: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_subsequences(nums :: [integer]) :: integer
def number_of_subsequences(nums) do

end
end
```

Erlang Solution:

```
-spec number_of_subsequences(Nums :: [integer()]) -> integer().
number_of_subsequences(Nums) ->
.
```

Racket Solution:

```
(define/contract (number-of-subsequences nums)
(-> (listof exact-integer?) exact-integer?)
)
```