Problem 446: Arithmetic Slices II - Subsequence

Problem Information

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

Problem Description

Given an integer array

nums

, return

the number of all the

arithmetic subsequences

of

nums

.

A sequence of numbers is called arithmetic if it consists of

at least three elements

and if the difference between any two consecutive elements is the same.

For example,

[1, 3, 5, 7, 9]

,

[7, 7, 7, 7]

, and

[3, -1, -5, -9]

are arithmetic sequences.

For example,

[1, 1, 2, 5, 7]

is not an arithmetic sequence.

A

subsequence

of an array is a sequence that can be formed by removing some elements (possibly none) of the array.

For example,

[2,5,10]

is a subsequence of

[1,2,1,

2

,4,1,

5

,

10

]

.

The test cases are generated so that the answer fits in

32-bit

integer.

Example 1:

Input:

nums = [2,4,6,8,10]

Output:

7

Explanation:

All arithmetic subsequence slices are: [2,4,6] [4,6,8] [6,8,10] [2,4,6,8] [4,6,8,10] [2,4,6,8,10] [2,6,10]

Example 2:

Input:

nums = [7,7,7,7,7]

Output:

16

Explanation:

Any subsequence of this array is arithmetic.

Constraints:

1 <= nums.length <= 1000

-2

31

<= nums[i] <= 2

31

- 1

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int numberOfArithmeticSlices(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int NumberOfArithmeticSlices(int[] nums) {

}
}
```

C:

```
int numberOfArithmeticSlices(int* nums, int numsSize) {

}
```

Go:

```
func numberOfArithmeticSlices(nums []int) int {
```

```
}
```

Kotlin:

```
class Solution {
fun numberOfArithmeticSlices(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
```

```
function numberOfArithmeticSlices($nums) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Arithmetic Slices II - Subsequence
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int numberOfArithmeticSlices(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Arithmetic Slices II - Subsequence
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numberOfArithmeticSlices(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Arithmetic Slices II - Subsequence
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Arithmetic Slices II - Subsequence
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

```
};
```

TypeScript Solution:

```
/**
* Problem: Arithmetic Slices II - Subsequence
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numberOfArithmeticSlices(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Arithmetic Slices II - Subsequence
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int NumberOfArithmeticSlices(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Arithmetic Slices II - Subsequence
* Difficulty: Hard
```

```
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numberOfArithmeticSlices(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Arithmetic Slices II - Subsequence
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numberOfArithmeticSlices(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfArithmeticSlices(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Arithmetic Slices II - Subsequence
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn number_of_arithmetic_slices(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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