Problem 1546: Maximum Number of Non-Overlapping Subarrays With Sum Equals Target

Problem Information

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

Problem Description

Given an array

nums

and an integer

target

, return

the maximum number of

non-empty

non-overlapping

subarrays such that the sum of values in each subarray is equal to

target

.

Example 1:

Input:

nums = [1,1,1,1,1], target = 2

Output:

2

Explanation:

There are 2 non-overlapping subarrays [

1,1

,1,

1,1

] with sum equals to target(2).

Example 2:

Input:

nums = [-1,3,5,1,4,2,-9], target = 6

Output:

2

Explanation:

There are 3 subarrays with sum equal to 6. ([5,1], [4,2], [3,5,1,4,2,-9]) but only the first 2 are non-overlapping.

Constraints:

1 <= nums.length <= 10

5

-10

4

<= nums[i] <= 10

4

0 <= target <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int maxNonOverlapping(vector<int>& nums, int target) {

}
};
```

Java:

```
class Solution {
public int maxNonOverlapping(int[] nums, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxNonOverlapping(nums: number[], target: number): number {

};
```

C#:

```
public class Solution {
public int MaxNonOverlapping(int[] nums, int target) {

}
}
```

C:

```
int maxNonOverlapping(int* nums, int numsSize, int target) {

}
```

Go:

```
func maxNonOverlapping(nums []int, target int) int {
}
```

Kotlin:

```
class Solution {
fun maxNonOverlapping(nums: IntArray, target: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def max_non_overlapping(nums, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function maxNonOverlapping($nums, $target) {

}
}
```

Dart:

```
class Solution {
int maxNonOverlapping(List<int> nums, int target) {

}
}
```

Scala:

```
object Solution {
def maxNonOverlapping(nums: Array[Int], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_non_overlapping(nums :: [integer], target :: integer) :: integer
def max_non_overlapping(nums, target) do

end
end
```

Erlang:

```
-spec max_non_overlapping(Nums :: [integer()], Target :: integer()) ->
integer().
max_non_overlapping(Nums, Target) ->
.
```

Racket:

```
(define/contract (max-non-overlapping nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Non-Overlapping Subarrays With Sum Equals Target
* Difficulty: Medium
* Tags: array, greedy, 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 maxNonOverlapping(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Non-Overlapping Subarrays With Sum Equals Target
* Difficulty: Medium
* Tags: array, greedy, 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 maxNonOverlapping(int[] nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Non-Overlapping Subarrays With Sum Equals Target
Difficulty: Medium
Tags: array, greedy, 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 maxNonOverlapping(self, nums: List[int], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Non-Overlapping Subarrays With Sum Equals Target
* Difficulty: Medium
* Tags: array, greedy, 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} target
* @return {number}
*/
var maxNonOverlapping = function(nums, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Non-Overlapping Subarrays With Sum Equals Target
* Difficulty: Medium
* Tags: array, greedy, 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 maxNonOverlapping(nums: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Non-Overlapping Subarrays With Sum Equals Target
* Difficulty: Medium
* Tags: array, greedy, 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 MaxNonOverlapping(int[] nums, int target) {

}

}
```

C Solution:

```
/*
* Problem: Maximum Number of Non-Overlapping Subarrays With Sum Equals Target
* Difficulty: Medium
* Tags: array, greedy, 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 maxNonOverlapping(int* nums, int numsSize, int target) {

}
```

Go Solution:

```
// Problem: Maximum Number of Non-Overlapping Subarrays With Sum Equals
Target
// Difficulty: Medium
// Tags: array, greedy, 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 maxNonOverlapping(nums []int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxNonOverlapping(nums: IntArray, target: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Non-Overlapping Subarrays With Sum Equals
Target
// Difficulty: Medium
// Tags: array, greedy, 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 max_non_overlapping(nums: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def max_non_overlapping(nums, target)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function maxNonOverlapping($nums, $target) {

}
}
```

Dart Solution:

```
class Solution {
int maxNonOverlapping(List<int> nums, int target) {

}
}
```

Scala Solution:

```
object Solution {
def maxNonOverlapping(nums: Array[Int], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_non_overlapping(nums :: [integer], target :: integer) :: integer
def max_non_overlapping(nums, target) do

end
end
```

Erlang Solution:

```
-spec max_non_overlapping(Nums :: [integer()], Target :: integer()) ->
integer().
max_non_overlapping(Nums, Target) ->
.
```

Racket Solution:

```
(define/contract (max-non-overlapping nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```