Problem 1477: Find Two Non-overlapping Sub-arrays Each With Target Sum

Problem Information

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

Problem Description

You are given an array of integers

arr

and an integer

target

.

You have to find

two non-overlapping sub-arrays

of

arr

each with a sum equal

target

. There can be multiple answers so you have to find an answer where the sum of the lengths of the two sub-arrays is

minimum

.

Return

the minimum sum of the lengths

of the two required sub-arrays, or return

-1

if you cannot find such two sub-arrays.

Example 1:

Input:

arr = [3,2,2,4,3], target = 3

Output:

2

Explanation:

Only two sub-arrays have sum = 3 ([3] and [3]). The sum of their lengths is 2.

Example 2:

Input:

arr = [7,3,4,7], target = 7

Output:

2

Explanation:

Although we have three non-overlapping sub-arrays of sum = 7 ([7], [3,4] and [7]), but we will choose the first and third sub-arrays as the sum of their lengths is 2.

Example 3:

Input:

arr = [4,3,2,6,2,3,4], target = 6

Output:

-1

Explanation:

We have only one sub-array of sum = 6.

Constraints:

1 <= arr.length <= 10

5

1 <= arr[i] <= 1000

1 <= target <= 10

8

Code Snippets

C++:

```
class Solution {
public:
int minSumOfLengths(vector<int>& arr, int target) {

}
```

```
};
```

Java:

```
class Solution {
public int minSumOfLengths(int[] arr, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minSumOfLengths(arr: number[], target: number): number {

};
```

C#:

```
public class Solution {
public int MinSumOfLengths(int[] arr, int target) {

}
}
```

C:

```
int minSumOfLengths(int* arr, int arrSize, int target) {

}
```

Go:

```
func minSumOfLengths(arr []int, target int) int {

}
```

Kotlin:

```
class Solution {
fun minSumOfLengths(arr: IntArray, target: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @param {Integer} target
# @return {Integer}
def min_sum_of_lengths(arr, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $target
* @return Integer
*/
function minSumOfLengths($arr, $target) {

}
}
```

Dart:

```
class Solution {
int minSumOfLengths(List<int> arr, int target) {

}
}
```

Scala:

```
object Solution {
def minSumOfLengths(arr: Array[Int], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_sum_of_lengths(arr :: [integer], target :: integer) :: integer
```

```
def min_sum_of_lengths(arr, target) do

end
end
```

Erlang:

```
-spec min_sum_of_lengths(Arr :: [integer()], Target :: integer()) ->
integer().
min_sum_of_lengths(Arr, Target) ->
.
```

Racket:

```
(define/contract (min-sum-of-lengths arr target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Two Non-overlapping Sub-arrays Each With Target Sum
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* 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 minSumOfLengths(vector<int>& arr, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Find Two Non-overlapping Sub-arrays Each With Target Sum
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* 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 minSumOfLengths(int[] arr, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Two Non-overlapping Sub-arrays Each With Target Sum
Difficulty: Medium
Tags: array, dp, hash, search

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 minSumOfLengths(self, arr: List[int], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Two Non-overlapping Sub-arrays Each With Target Sum
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* 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[]} arr
* @param {number} target
* @return {number}
*/
var minSumOfLengths = function(arr, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Two Non-overlapping Sub-arrays Each With Target Sum
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* 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 minSumOfLengths(arr: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Find Two Non-overlapping Sub-arrays Each With Target Sum
* Difficulty: Medium
```

```
* Tags: array, dp, hash, search
*
* 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 MinSumOfLengths(int[] arr, int target) {

}
}
```

C Solution:

```
/*
* Problem: Find Two Non-overlapping Sub-arrays Each With Target Sum
* Difficulty: Medium
* Tags: array, dp, hash, search
*
* 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 minSumOfLengths(int* arr, int arrSize, int target) {

}
```

Go Solution:

```
// Problem: Find Two Non-overlapping Sub-arrays Each With Target Sum
// Difficulty: Medium
// Tags: array, dp, hash, search
//
// 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 minSumOfLengths(arr []int, target int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun minSumOfLengths(arr: IntArray, target: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Two Non-overlapping Sub-arrays Each With Target Sum
// Difficulty: Medium
// Tags: array, dp, hash, search
//
// 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 min_sum_of_lengths(arr: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} target
# @return {Integer}
def min_sum_of_lengths(arr, target)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $target
* @return Integer
*/
function minSumOfLengths($arr, $target) {

}
}
```

Dart Solution:

```
class Solution {
int minSumOfLengths(List<int> arr, int target) {

}
}
```

Scala Solution:

```
object Solution {
def minSumOfLengths(arr: Array[Int], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_sum_of_lengths(arr :: [integer], target :: integer) :: integer
def min_sum_of_lengths(arr, target) do

end
end
```

Erlang Solution:

```
-spec min_sum_of_lengths(Arr :: [integer()], Target :: integer()) ->
integer().
min_sum_of_lengths(Arr, Target) ->
.
```

Racket Solution:

```
(define/contract (min-sum-of-lengths arr target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```