Problem 2875: Minimum Size Subarray in Infinite Array

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

and an integer

target

.

A

0-indexed

array

infinite_nums

is generated by infinitely appending the elements of

nums

to itself.

Return

the length of the

shortest

subarray of the array

infinite_nums

with a sum equal to

target

.

If there is no such subarray return

-1

.

Example 1:

Input:

nums = [1,2,3], target = 5

Output:

2

Explanation:

In this example infinite_nums = [1,2,3,1,2,3,1,2,...]. The subarray in the range [1,2], has the sum equal to target = 5 and length = 2. It can be proven that 2 is the shortest length of a subarray with sum equal to target = 5.

Example 2:

Input:

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

Output:

2

Explanation:

In this example infinite_nums = [1,1,1,2,3,1,1,1,2,3,1,1,...]. The subarray in the range [4,5], has the sum equal to target = 4 and length = 2. It can be proven that 2 is the shortest length of a subarray with sum equal to target = 4.

Example 3:

Input:

nums = [2,4,6,8], target = 3

Output:

-1

Explanation:

In this example infinite_nums = [2,4,6,8,2,4,6,8,...]. It can be proven that there is no subarray with sum equal to target = 3.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

1 <= target <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

```
func minSizeSubarray(nums []int, target int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-size-subarray nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Size Subarray in Infinite Array
```

```
* Difficulty: Medium
* Tags: array, 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 minSizeSubarray(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Size Subarray in Infinite Array
* Difficulty: Medium
* Tags: array, 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 minSizeSubarray(int[] nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Size Subarray in Infinite Array
Difficulty: Medium
Tags: array, 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 minSizeSubarray(self, nums: List[int], target: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Size Subarray in Infinite Array
* Difficulty: Medium
* Tags: array, 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 minSizeSubarray = function(nums, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Size Subarray in Infinite Array
* Difficulty: Medium
* Tags: array, 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 minSizeSubarray(nums: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Size Subarray in Infinite Array
* Difficulty: Medium
* Tags: array, 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 MinSizeSubarray(int[] nums, int target) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Size Subarray in Infinite Array
* Difficulty: Medium
* Tags: array, 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 minSizeSubarray(int* nums, int numsSize, int target) {

}
```

Go Solution:

```
// Problem: Minimum Size Subarray in Infinite Array
// Difficulty: Medium
// Tags: array, 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 minSizeSubarray(nums []int, target int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Size Subarray in Infinite Array
// Difficulty: Medium
// Tags: array, 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 min_size_subarray(nums: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-size-subarray nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```