Problem 325: Maximum Size Subarray Sum Equals k

Problem Information

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

Problem Description

Given an integer array

nums

and an integer

k

, return

the maximum length of a

subarray

that sums to

k

. If there is not one, return

0

instead.

Example 1:

Input:

nums = [1,-1,5,-2,3], k = 3

Output:

4

Explanation:

The subarray [1, -1, 5, -2] sums to 3 and is the longest.

Example 2:

Input:

nums = [-2,-1,2,1], k = 1

Output:

2

Explanation:

The subarray [-1, 2] sums to 1 and is the longest.

Constraints:

1 <= nums.length <= 2 * 10

5

-10

4

<= nums[i] <= 10

4

-10

9

<= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxSubArrayLen(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int maxSubArrayLen(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

```
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maxSubArrayLen(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxSubArrayLen(int[] nums, int k) {

}
}
```

C:

```
int maxSubArrayLen(int* nums, int numsSize, int k) {

}
```

Go:

```
func maxSubArrayLen(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxSubArrayLen(nums: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def max_sub_array_len(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxSubArrayLen($nums, $k) {

}
}
```

Dart:

```
class Solution {
int maxSubArrayLen(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def maxSubArrayLen(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_sub_array_len(nums :: [integer], k :: integer) :: integer
def max_sub_array_len(nums, k) do

end
end
```

Erlang:

```
-spec max_sub_array_len(Nums :: [integer()], K :: integer()) -> integer().
max_sub_array_len(Nums, K) ->
.
```

Racket:

```
(define/contract (max-sub-array-len nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Size Subarray Sum Equals k
* 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 maxSubArrayLen(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Size Subarray Sum Equals k
* 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 maxSubArrayLen(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Size Subarray Sum Equals k
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 maxSubArrayLen(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Size Subarray Sum Equals k
* 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} k
* @return {number}
*/
var maxSubArrayLen = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Size Subarray Sum Equals k
* 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 maxSubArrayLen(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Size Subarray Sum Equals k
* 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 MaxSubArrayLen(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Size Subarray Sum Equals k
* 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 maxSubArrayLen(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Size Subarray Sum Equals k
// 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 maxSubArrayLen(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSubArrayLen(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Size Subarray Sum Equals k
// 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 max_sub_array_len(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def max_sub_array_len(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxSubArrayLen($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxSubArrayLen(List<int> nums, int k) {
}
}
```

Scala Solution:

```
object Solution {
def maxSubArrayLen(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_sub_array_len(nums :: [integer], k :: integer) :: integer
def max_sub_array_len(nums, k) do

end
end
```

Erlang Solution:

```
-spec max_sub_array_len(Nums :: [integer()], K :: integer()) -> integer().
max_sub_array_len(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (max-sub-array-len nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```