Problem 2395: Find Subarrays With Equal Sum

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

, determine whether there exist

two

subarrays of length

2

with

equal

sum. Note that the two subarrays must begin at

different

indices.

Return

true

if these subarrays exist, and

false

otherwise.

A

subarray

is a contiguous non-empty sequence of elements within an array.

Example 1:

Input:

nums = [4,2,4]

Output:

true

Explanation:

The subarrays with elements [4,2] and [2,4] have the same sum of 6.

Example 2:

Input:

nums = [1,2,3,4,5]

Output:

false

Explanation:

No two subarrays of size 2 have the same sum.

Example 3:

Input:

nums = [0,0,0]

Output:

true

Explanation:

The subarrays [nums[0],nums[1]] and [nums[1],nums[2]] have the same sum of 0. Note that even though the subarrays have the same content, the two subarrays are considered different because they are in different positions in the original array.

Constraints:

2 <= nums.length <= 1000

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
```

```
bool findSubarrays(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public boolean findSubarrays(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findSubarrays(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool FindSubarrays(int[] nums) {

}
}
```

C:

```
bool findSubarrays(int* nums, int numsSize) {

}
```

Go:

```
func findSubarrays(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun findSubarrays(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool findSubarrays(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (find-subarrays nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Subarrays With Equal Sum
* Difficulty: Easy
* 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:
bool findSubarrays(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find Subarrays With Equal Sum
* Difficulty: Easy
* 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 boolean findSubarrays(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Subarrays With Equal Sum
Difficulty: Easy
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 findSubarrays(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Subarrays With Equal Sum
* Difficulty: Easy
* 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
* @return {boolean}
*/
var findSubarrays = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Subarrays With Equal Sum
* Difficulty: Easy
* 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 findSubarrays(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Find Subarrays With Equal Sum
* Difficulty: Easy
* 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 bool FindSubarrays(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find Subarrays With Equal Sum
* Difficulty: Easy
* 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
*/

bool findSubarrays(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find Subarrays With Equal Sum
// Difficulty: Easy
// 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 findSubarrays(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun findSubarrays(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Subarrays With Equal Sum
// Difficulty: Easy
// 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 find_subarrays(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $nums
* @return Boolean
*/
function findSubarrays($nums) {

}
}
```

Dart Solution:

```
class Solution {
bool findSubarrays(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-subarrays nums)
(-> (listof exact-integer?) boolean?)
)
```