Problem 523: Continuous Subarray Sum

Problem Information

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

Problem Description

Given an integer array nums and an integer k, return

true

if

nums

has a

good subarray

or

false

otherwise

.

A

good subarray

is a subarray where:

its length is

at least two

, and

the sum of the elements of the subarray is a multiple of

k

.

Note

that:

A

subarray

is a contiguous part of the array.

An integer

x

is a multiple of

k

if there exists an integer

n

such that

x = n * k

.

0

is

always

a multiple of

k

.

Example 1:

Input:

nums = [23,

2,4

,6,7], k = 6

Output:

true

Explanation:

[2, 4] is a continuous subarray of size 2 whose elements sum up to 6.

Example 2:

Input:

nums = [

23,2,6,4,7

], k = 6

Output:

true

Explanation:

[23, 2, 6, 4, 7] is an continuous subarray of size 5 whose elements sum up to 42. 42 is a multiple of 6 because 42 = 7 * 6 and 7 is an integer.

Example 3:

Input:

nums = [23,2,6,4,7], k = 13

Output:

false

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

0 <= sum(nums[i]) <= 2

31

- 1

1 <= k <= 2

31

- 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
bool checkSubarraySum(int* nums, int numsSize, int k) {

}
```

Go:

```
func checkSubarraySum(nums []int, k int) bool {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (check-subarray-sum nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Continuous Subarray Sum
* Difficulty: Medium
```

```
* Tags: array, math, 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 checkSubarraySum(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Continuous Subarray Sum
* Difficulty: Medium
* Tags: array, math, 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 checkSubarraySum(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Continuous Subarray Sum
Difficulty: Medium
Tags: array, math, 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 checkSubarraySum(self, nums: List[int], k: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Continuous Subarray Sum
* Difficulty: Medium
* Tags: array, math, 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 {boolean}
*/
var checkSubarraySum = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Continuous Subarray Sum
* Difficulty: Medium
* Tags: array, math, 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 checkSubarraySum(nums: number[], k: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Continuous Subarray Sum
* Difficulty: Medium
* Tags: array, math, 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 CheckSubarraySum(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Continuous Subarray Sum
* Difficulty: Medium
* Tags: array, math, 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 checkSubarraySum(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Continuous Subarray Sum
// Difficulty: Medium
// Tags: array, math, 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 checkSubarraySum(nums []int, k int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Continuous Subarray Sum
// Difficulty: Medium
// Tags: array, math, 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 check_subarray_sum(nums: Vec<i32>, k: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (check-subarray-sum nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```