Problem 2025: Maximum Number of Ways to Partition an Array

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

of length

n

. The number of ways to

partition

nums

is the number of

pivot

indices that satisfy both conditions:

1 <= pivot < n

nums[0] + nums[1] + ... + nums[pivot - 1] == nums[pivot] + nums[pivot + 1] + ... + nums[n - 1]

You are also given an integer

k

. You can choose to change the value of

one

element of

nums

to

k

, or to leave the array

unchanged

.

Return

the

maximum

possible number of ways to

partition

nums

to satisfy both conditions after changing

at most

one element

.

Example 1:

Input:

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

Output:

1

Explanation:

One optimal approach is to change nums[0] to k. The array becomes [

3

,-1,2]. There is one way to partition the array: - For pivot = 2, we have the partition [3,-1 | 2]: 3 + -1 == 2.

Example 2:

Input:

nums = [0,0,0], k = 1

Output:

2

Explanation:

The optimal approach is to leave the array unchanged. There are two ways to partition the array: - For pivot = 1, we have the partition [0 | 0,0]: 0 == 0 + 0. - For pivot = 2, we have the

partition [0,0 | 0]: 0 + 0 == 0.

Example 3:

Input:

nums = [22,4,-25,-20,-15,15,-16,7,19,-10,0,-13,-14], k = -33

Output:

4

Explanation:

One optimal approach is to change nums[2] to k. The array becomes [22,4,

-33

,-20,-15,15,-16,7,19,-10,0,-13,-14]. There are four ways to partition the array.

Constraints:

n == nums.length

2 <= n <= 10

5

-10

5

<= k, nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (ways-to-partition nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Ways to Partition an Array
* Difficulty: Hard
* 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 waysToPartition(vector<int>& nums, int k) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Ways to Partition an Array
* Difficulty: Hard
* 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 waysToPartition(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Ways to Partition an Array
Difficulty: Hard
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 waysToPartition(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Ways to Partition an Array
* Difficulty: Hard
* 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 waysToPartition = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Ways to Partition an Array
* Difficulty: Hard
* 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 waysToPartition(nums: number[], k: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Maximum Number of Ways to Partition an Array
* Difficulty: Hard
* 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 WaysToPartition(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Ways to Partition an Array
* Difficulty: Hard
* 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 waysToPartition(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Number of Ways to Partition an Array
// Difficulty: Hard
```

```
// 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 waysToPartition(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Ways to Partition an Array
// Difficulty: Hard
// 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 ways_to_partition(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (ways-to-partition nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```