Problem 3659: Partition Array Into K-Distinct Groups

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

Your task is to determine whether it is possible to partition all elements of

nums

into one or more groups such that:

Each group contains

exactly

k

elements.

All elements in each group are

distinct

.

Each element in

nums

must be assigned to

exactly

one group.

Return

true

if such a partition is possible, otherwise return

false

.

Example 1:

Input:

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

Output:

true

Explanation:

One possible partition is to have 2 groups:

Group 1:

[1, 2]

Group 2:

[3, 4]

Each group contains

k = 2

distinct elements, and all elements are used exactly once.

Example 2:

Input:

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

Output:

true

Explanation:

One possible partition is to have 2 groups:

Group 1:

[2, 3]

Group 2:

[2, 5]

Each group contains

k = 2

distinct elements, and all elements are used exactly once.

Example 3:

Input:

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

Output:

false

Explanation:

We cannot form groups of

k = 3

distinct elements using all values exactly once.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

1 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (partition-array nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Partition Array Into K-Distinct Groups
* 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:
bool partitionArray(vector<int>& nums, int k) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Partition Array Into K-Distinct Groups
* 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 boolean partitionArray(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Partition Array Into K-Distinct Groups
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 partitionArray(self, nums: List[int], k: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition Array Into K-Distinct Groups
* 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 {boolean}
*/
var partitionArray = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Partition Array Into K-Distinct Groups
* 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 partitionArray(nums: number[], k: number): boolean {
```

```
};
```

C# Solution:

```
/*
* Problem: Partition Array Into K-Distinct Groups
* 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 bool PartitionArray(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Partition Array Into K-Distinct Groups
* 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
*/

bool partitionArray(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Partition Array Into K-Distinct Groups
// 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 partitionArray(nums []int, k int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Partition Array Into K-Distinct Groups
// 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 partition_array(nums: Vec<i32>, k: i32) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (partition-array nums k)
(-> (listof exact-integer?) exact-integer? boolean?)
)
```